Simon Monk

Das Action-Buch für Maker

Bewegung, Licht und Sound mit Arduino und Raspberry Pi – Experimente und Projekte

edition Make:

dpunkt.verlag

Simon Monk

Das Action-Buch für Maker

Bewegung, Licht und Sound mit Arduino und Raspberry Pi – Experimente und Projekte

dpunkt.verlag

Simon Monk

Lektorat: Dr. Michael Barabas
Fachgutachter: Duncan Amos
Aktualisierungen: Maik Schmidt
Copy-Editing: Ursula Zimpfer
Übersetzung & Satz: G&U Language & Publishing Services GmbH, www.gundu.com
Herstellung: Nadine Thiele
Umschlaggestaltung: Helmut Kraus, www.exclam.de
nach der Originalvorlage von No Starch Press
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, Paderborn

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

ISBN:
Print 978-3-86490-385-4
PDF 978-3-96088-027-1
ePub 978-3-96088-028-8
mobi 978-3-96088-029-5

1. Auflage 2016

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Make: Action* ISBN 9781457187797

5 4 3 2 1 0

Inhalt

Der Autor

Simon Monk ist Vollzeit-Autor, der vor allem Bücher über Elektronik für Bastler schreibt. Zu seinen bekannteren Werken gehören *Programming Arduino: Getting Started with Sketches*, *Raspberry Pi Kochbuch* und *Elektronik-Hacks: Ein Do-It-Yourself-Guide für Einsteiger*. Außerdem hilft er seiner Frau Linda, der Geschäftsführerin von MonkMakes.com, Bausätze und andere Produkte rund um seine Bücher herzustellen und zu verkaufen. Sie können Simon auf Twitter folgen und auf *simonmonk.org* mehr über seine Bücher erfahren.

Der Fachgutachter

Duncan Amos hat den Großteil seiner mehr als 50 Arbeitsjahre als Rundfunkingenieur zugebracht, aber auch Satellitenteilsysteme entwickelt und gebaut, technische Handbücher und Bedienungsanleitungen verfasst, Nutzvieh künstlich besamt, Gartengeräte repariert sowie Möbel entworfen und gebaut. Als eingefleischter Bastler, der erst spät im Leben Bekanntschaft mit Mikrocontrollern geschlossen hat, kennt er den Wert klarer Erläuterungen von komplizierten Techniken.

1 Einleitung

Mit dem Arduino und dem Raspberry Pi ist es für Bastler leichter denn je, in die Welt der Elektronik einzusteigen, etwa um ein Heimautomatisierungssystem im Eigenbau zu realisieren, mit dem Sie die Beleuchtung und die Heizung über ein WLAN steuern können, oder einfach um Motoren zu regeln.

In diesem Buch zeige ich Ihnen, wie Sie diese beiden beliebten Plattformen einsetzen können, um mit Ihrem Raspberry Pi oder Arduino Bewegungen, Licht und Ton hervorzurufen und zu steuern.

Arduino und Pi

Sowohl der Arduino als auch der Raspberry Pi sind kleine Platinen ungefähr von der Größe einer Kreditkarte, aber im Grunde genommen handelt es sich bei ihnen um sehr unterschiedliche Geräte. Der Arduino ist eine sehr einfache Mikrocontroller-Platine ohne irgendeine Form von Betriebssystem, während es sich bei dem Raspberry Pi um einen Minicomputer mit Linux handelt, an den außerdem externe elektronische Geräte angeschlossen werden können.

Der Raspberry Pi

Wenn Sie noch keine Erfahrungen mit Elektronik haben, aber mit Computern vertraut sind, ist der Raspberry Pi (siehe Abb. 1–1) für Sie das geläufigere Gerät. Er stellt eine äußerst kleine Version eines normalen Linux-Computers dar und verfügt über USB-Anschlüsse für eine Tastatur und eine Maus, einen HDMI-Videoausgang für einen Monitor oder einen Fernseher und einen Audioausgang.

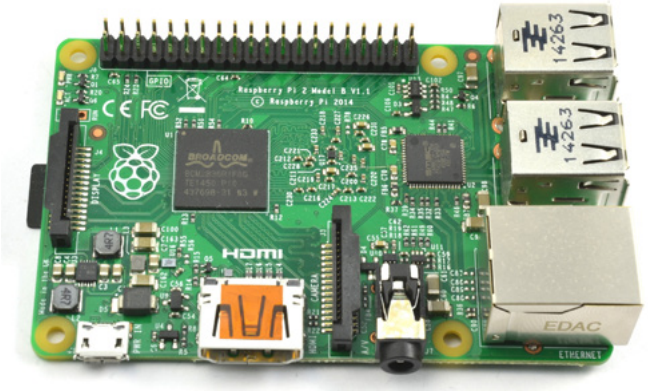

Abb. 1–1 *Ein Raspberry Pi 2*

Über seinen Ethernetanschluss können Sie den Raspberry Pi auch mit einem Netzwerk verbinden. Sie können aber auch USB-WLAN-Adapter anschließen. Die Stromversorgung erfolgt über eine Micro-USB-Buchse.

Für die Speicherung wird statt eines herkömmlichen Festplattenlaufwerks eine Micro-SD-Karte verwendet. Sie enthält sowohl das Betriebssystem als auch Ihre Dokumente und Programme.

Entwickelt wurde der Raspberry Pi in Großbritannien, hauptsächlich als kostengünstiger Computer, um Schulkindern die Grundlagen der Informatik beizubringen, insbesondere die Programmierung in Python. Der Name Pi soll sogar von *Python* abgeleitet sein.

Zwischen einem normalen Desktop- oder Laptop-Computer mit Linux und dem Raspberry Pi bestehen vor allem folgende Unterschiede:

- Der Pi kostet lediglich um die 40 €. (Ein abgespecktes Modell namens A+ ist für einen noch niedrigeren Preis erhältlich, und das Modell 0 kostet sogar noch weniger.)
- Der Pi hat eine Leistungsaufnahme von weniger als 5 W.
- Der Pi verfügt über eine Doppelreihe von GPIO-Pins (General Purpose Input/Output, also »Allzweck-E/A-Pins«), an die Sie direkt elektronische Geräte anschließen können. (In Abb. 1–1 sehen Sie diese Pins auf der linken Seite der oberen Kante.) Über diese Pins können Sie LEDs, Displays, Motoren und viele weitere Arten von Ausgabegeräten steuern, mit denen wir in diesem Buch noch arbeiten werden.

Außerdem kann der Raspberry Pi über ein WLAN oder ein LAN-Kabel mit dem Internet verbunden werden, was ihn für Projekte im Rahmen des »Internets der Dinge« (siehe Kapitel 16) geeignet macht.

Der in diesem Buch hauptsächlich verwendete Raspberry Pi 2 weist folgende technische Daten auf:

- 900-MHz-Quad-Core-Prozessor ARM v7
- 1 GB DDR2-Speicher
- 100-BaseT-Ethernet
- 4 USB-2.0-Anschlüsse
- HDMI-Videoausgang
- Buchse mit Kameraschnittstelle
- 40-polige GPIO-Stiftleiste (alle Pins werden mit 3,3 V betrieben)

Nach Abfassung der Originalausgabe dieses Buches ist das neue Modell Raspberry Pi 3 auf den Markt gekommen.

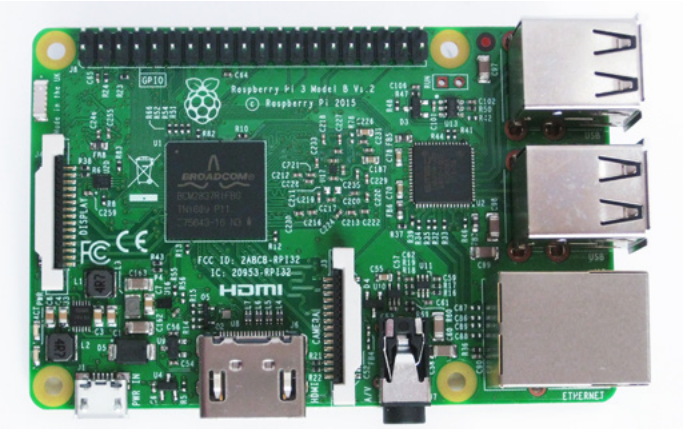

Abb. 1–2 *Optisch kaum vom Raspberry Pi 2 zu unterscheiden: das neue Modell 3*

Äußerlich sieht der Pi 3 praktisch genauso aus wie das Modell 2, und auch alle vertrauten Anschlüsse sind unverändert vorhanden. Es hat jedoch erhebliche »unsichtbare« Neuerungen gegeben, nämlich bei der Rechenleistung und bei den Anschlüssen. So ist der Raspberry Pi 3 jetzt mit einem 64-Bit-Vierkernprozessor vom Typ ARMv8 mit 1,2 GHz ausgestattet und verfügt über eine 802.11n-WLAN-Verbindung sowie über einen Bluetooth-Anschluss (Bluetooth 4.1 und Bluetooth Low Energy). Der in Kapitel 3 erwähnte WLAN-Adapter ist daher bei diesem Modell nicht mehr nötig.

Wenn der Raspberry Pi für Sie neu ist, können Sie sich mit dem Einführungskurs in Kapitel 3 sehr schnell mit der Hardware und der Programmiersprache Python vertraut machen.

Der Arduino

Es gibt eine breite Palette von unterschiedlichen Arduino-Modellen. In diesem Buch konzentrieren wir uns auf das am weitesten verbreitete, nämlich den Arduino Uno (siehe Abb. 1–3). Er ist noch ein bisschen billiger als der Raspberry Pi – Sie können ihn schon für 25 € bekommen.

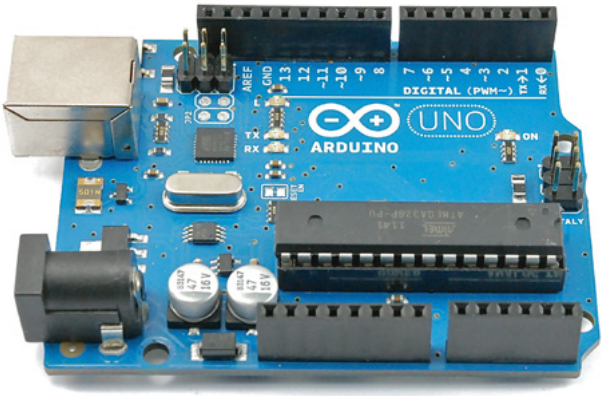

Abb. 1–3 *Ein Arduino Uno Revision 3*

Wenn Sie die Arbeit mit einem regulären Computer gewohnt sind, werden Ihnen die technischen Daten des Arduino höchst unzureichend vorkommen. Er hat lediglich einen Arbeitsspeicher (unterschiedlicher Art) von 34 KB. Das bedeutet, dass der Raspberry Pi einen etwa 30.000 Mal größeren Arbeitsspeicher aufweist, worin der Flash-Speicher der SD-Karte noch nicht einmal enthalten ist. Des Weiteren beträgt der Prozessortakt des Arduino Uno nur 16 MHz. Es ist auch nicht möglich, eine Tastatur, eine Maus oder einen Monitor an den Arduino anzuschließen oder auf ihm ein Betriebssystem auszuführen.

Vielleicht fragen Sie sich nun, wie dieses kleine Gerät überhaupt irgendetwas Sinnvolles tun kann. Das Geheimnis des Arduino liegt jedoch gerade in seiner Einfachheit. Es gibt kein Betriebssystem, das Sie erst starten müssten, und keine anderen Schnittstellen, die für ein Projekt vielleicht unnötig wären und nur Kosten verursachen und Strom verbrauchen würden.

Der Raspberry Pi ist ein Allzweckcomputer, der Arduino dagegen soll einzig und allein eine Sache gut machen, nämlich das Steuern elektronischer Geräte.

Um einen Arduino zu programmieren, brauchen Sie einen regulären Computer (Sie können dazu auch einen Raspberry Pi verwenden). Auf diesem Computer benötigen Sie eine integrierte Entwicklungsumgebung (Integrated Development

Environment, IDE). Darin schreiben Sie die Programme, die anschließend in den Flash-Speicher des Arduino heruntergeladen werden.

Der Arduino kann immer nur ein Programm auf einmal ausführen. Nachdem er programmiert wurde, merkt er sich das Programm und führt es automatisch aus, sobald er eingeschaltet wird.

Arduinos sind so gestaltet, dass Sie sogenannte *Shields* auf die E/A-Anschlüsse aufstecken können. Diese Shields tragen zusätzliche Hardware, z. B. verschiedene Arten von Anzeigen, Ethernet- oder WLAN-Adapter.

Zur Programmierung eines Arduino verwenden Sie die Programmiersprache C (mehr darüber erfahren Sie in Kapitel 2).

Welches Gerät – Arduino oder Pi?

In diesem Buch wird erklärt, wie Sie elektronische Geräte sowohl an den Arduino als auch an den Raspberry Pi anschließen können. Einer der Gründe dafür besteht darin, dass für einige Projekte der Pi besser geeignet ist, für andere der Arduino. Es gibt noch andere Platinen, die zwischen diesen beiden Extremen liegen. Im Allgemeinen ähneln sie entweder dem Arduino oder dem Raspberry Pi, sodass Ihnen dieses Buch auch dabei helfen kann, mit diesen Alternativen zu arbeiten.

Wenn ich mit einem neuen Projekt beginne, nehme ich im Allgemeinen einen Arduino. Stellt das Projekt aber eine der folgenden Anforderungen, dann ist ein Raspberry Pi wahrscheinlich die bessere Wahl:

- Internet- oder Netzwerkanschluss
- Großer Bildschirm
- Tastatur und Maus
- USB-Peripheriegeräte, z. B. eine Webcam

Mit einigen Kosten und Mühen ist es möglich, einen Arduino mithilfe von Shields so zu erweitern, dass er diese Anforderungen ebenfalls erfüllt. Allerdings ist es schwieriger, die Sachen auf diese Weise zum Laufen zu bekommen, da nichts davon zu den eigentlichen Funktionen des Arduino gehört.

Gute Gründe dafür, den Arduino gegenüber dem Raspberry Pi vorzuziehen, sind die folgenden:

- *Kosten* Ein Arduino Uno ist billiger als ein Raspberry Pi 2.
- *Startzeit* Ein Arduino muss nicht darauf warten, dass das Betriebssystem hochfährt. Es gibt eine kleine Verzögerung von etwa einer Sekunde, in der das Gerät prüft, ob ein neues Programm hochgeladen wird, aber danach ist der Arduino in Betrieb.

- *Zuverlässigkeit* Der Arduino ist ein viel einfacheres und robusteres Gerät als der Raspberry Pi und kommt ohne den Zusatzaufwand eines Betriebssystems aus.
- *Stromverbrauch* Ein Arduino verbraucht nur ein Zehntel so viel Strom wie ein Raspberry Pi. Wenn Sie Ihr Projekt mit Batterie- oder Solarstrom betreiben müssen, ist der Arduino die bessere Wahl.
- *GPIO-Ausgabestrom* Die GPIO-Pins eines Raspberry Pi können nur einen Strom von maximal 16 mA bereitstellen. Die Arduino-Pins dagegen sind für 40 mA ausgelegt. Daher lassen sich manche Dinge (z. B. helle LEDs), die sie auf diese Weise nicht mit einem Raspberry Pi verbinden können, direkt an einen Arduino anschließen.

Sowohl der Arduino als auch der Raspberry Pi sind bestens als Basis für Elektronikprojekte geeignet. Für welches Gerät Sie sich entscheiden, ist in gewissem Maße auch eine Frage des persönlichen Geschmacks.

Beim Anschluss von elektronischen Geräten an den Raspberry Pi müssen Sie unbedingt daran denken, dass er mit 3,3 V betrieben wird – nicht mit 5 V wie der Arduino. Wenn Sie an einen GPIO-Pin des Raspberry Pi 5 V anlegen, können Sie den Pin oder gar den ganzen Pi beschädigen oder zerstören.

Alternativen

Der Arduino und der Raspberry Pi befinden sich an den beiden Enden des Spektrums von Geräten, mit denen sich Geräte steuern lassen. Wie zu erwarten ist, hat der Markt eine große Zahl von anderen Geräten hervorgebracht, die zwischen diesen beiden Polen einzuordnen sind. Manche davon kombinieren die Vorteile der beiden Plattformen.

Neue Geräte kommen ständig auf den Markt. Die Open-Source-Natur des Arduino hat unzählige Varianten möglich gemacht, die auf bestimmte Nischen zugeschnitten sind, z. B. auf die Steuerung von Drohnen oder die Arbeit mit drahtlosen Sensoren.

Abbildung 1–4 zeigt ein Diagramm der am weitesten verbreiteten Geräte auf diesem Gebiet.

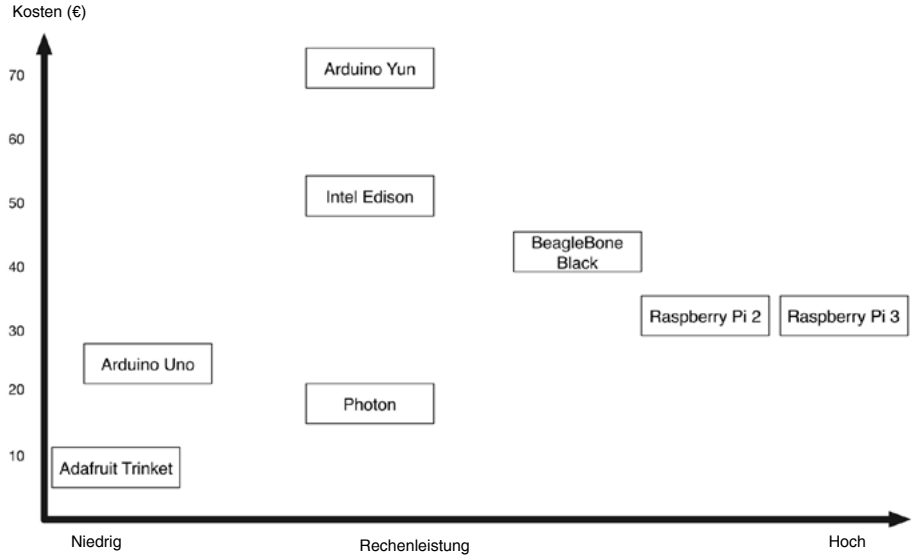

Abb. 1–4 *Plattformen für eingebettete Geräte*

Unterhalb des Arduino – sowohl was den Preis als auch die Leistung angeht – ist der Adafruit Trinket einzuordnen. Diese interessante Platine verfügt nur über wenige GPIO-Pins, ist ansonsten aber ziemlich gut mit dem Arduino kompatibel. Wenn Sie in einem Projekt nur ein oder zwei Eingänge oder Ausgänge benötigen, ist diese Platine eine Überlegung wert.

Im Mittelfeld liegen Produkte wie der Arduino Yun, der Intel Edison und der Photon. Alle verfügen über WLAN-Fähigkeiten und sind für Projekte im Internet der Dinge (Internet of Things, IoT) gedacht (siehe Kapitel 16). Von diesen Geräten bietet der Photon wahrscheinlich den größten Wert für das Geld. Alle drei Geräte werden in Arduino C programmiert. Was Sie also über die Programmierung des Arduino lernen, können Sie auch für diese Platinen anwenden.

Der BeagleBone Black ist ähnlich konzipiert wie der Raspberry Pi. Auch bei ihm handelt es sich um einen Einplatinencomputer. Was die Rohleistung angeht, fällt die aktuelle Version des BeagleBone Black zwar hinter dem Raspberry Pi zurück, doch dafür bietet er andere Vorteile: Er weist mehr GPIO-Pins auf und verfügt sogar über einige Pins, die als analoge Eingänge dienen können; ein Merkmal, das dem Raspberry Pi 2 fehlt. Den BeagleBone Black können Sie ebenso wie den Raspberry Pi in Python programmieren, aber auch in JavaScript.

Zusammenfassung

In diesem Kapitel haben Sie eine kurze Einführung in den Arduino und den Raspberry Pi erhalten. Wir haben die Vor- und Nachteile der beiden Platinen betrachtet und uns auch einige Alternativen angeschaut. In den nächsten beiden Kapiteln erhalten Sie eine Einführung, wie Sie den Arduino und den Raspberry Pi verwenden und programmieren können.

Wenn Sie bereits Erfahrungen mit dem Arduino und dem Raspberry Pi haben, können Sie auch gleich zu Kapitel 4 vorblättern und einige praktische Projekte damit bauen. Zum Nachschlagen können Sie jederzeit zu Kapitel 2 bzw. 3 zurückblättern.

2 Der Arduino

Dieses Kapitel basiert auf dem »Grundkurs Arduino«, der sich als Anhang in meinem Buch *Der Maker-Guide für die Zombie-Apokalypse* befindet.

Wenn der Arduino für Sie noch neu ist, können Sie sich in diesem Kapitel mit dieser großartigen kleinen Platine vertraut machen.

Was ist ein Arduino?

Es gibt verschiedene Arduino-Modelle. Das gebräuchlichste ist der Arduino Uno, der auch für alle Arduino-Projekte in diesem Buch verwendet wird (siehe Abb. 2–1).

Der Arduino Uno ist einer Reihe von Überarbeitungen unterzogen worden. In Abbildung 2–1 sehen Sie das Modell R3 (Revision 3), das zum Zeitpunkt der Entstehung dieses Buches das neueste Modell war.

Wir beginnen unsere Beschreibung mit der USB-Buchse. Sie dient gleich mehreren Zwecken, nämlich um den Arduino mit Strom zu versorgen, um ihn zum Programmieren an einen Computer anzuschließen oder als Kommunikationsverbindung mit anderen Rechnern.

Die kleine rote Taste neben der USB-Buchse ist der Reset-Knopf. Wenn Sie darauf drücken, wird das auf dem Arduino installierte Programm neu gestartet.

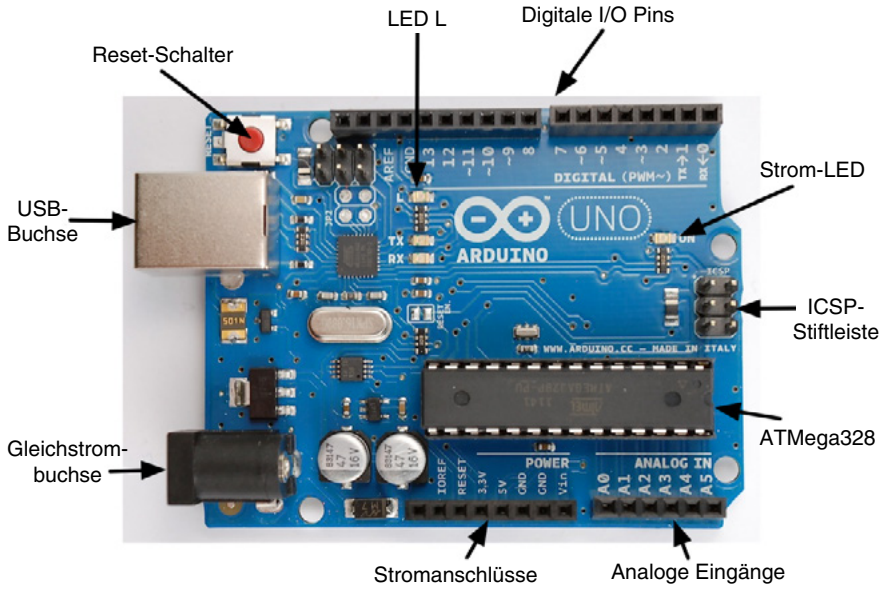

Abb. 2–1 *Ein Arduino Uno R3*

Über die Anschlussleisten am oberen und unteren Rand des Arduino können Sie elektronische Geräte an die Platine anschließen. Oben in Abbildung 2–1 befinden sich die digitalen I/O-Pins, die von 0 bis 13 durchnummeriert sind und einzeln als Eingänge oder Ausgänge eingerichtet werden können. Wenn Sie beispielsweise eine Taste mit einem digitalen Eingang verbinden, kann der Eingang erkennen, ob die Taste gedrückt wurde. Ausgänge dagegen senden Informationen oder Strom. Wenn Sie eine LED an einen digitalen Ausgang anschließen, können Sie sie einschalten, indem Sie den Ausgang von `low` auf `high` schalten. Eine LED mit der Bezeichnung L ist fest auf der Platine montiert und mit Digitalpin 13 verbunden.

Die Strom-LED unter den digitalen I/O-Pins zeigt an, ob die Platine mit Strom versorgt wird. Die ICSP-Stiftleiste (In-Circuit Serial Programming) ist nur für professionelle Programmierverfahren erforderlich, die die meisten Hobbybenutzer des Arduino nie anwenden werden.

Der ATMega328 ist ein Mikrocontroller, ein integrierter Schaltkreis (Integrated Circuit, IC), der das Gehirn des Arduino bildet. Er enthält 32 KB Flash-Speicher, in denen die Programme gespeichert werden, die Sie auf dem Arduino ausführen.

Unter dem ATMega328 sehen Sie eine Reihe von analogen Eingangspins, die von A0 bis A5 durchnummeriert sind. Digitale Eingänge können Ihnen nur mitteilen, ob irgendetwas ein- oder ausgeschaltet ist, wohingegen analoge Eingänge die am Pin anliegende Spannung messen (sofern sie zwischen 0 V und 5 V liegt).

Eine solche Spannung kann beispielsweise von einem Sensor stammen. Wenn Ihnen für ein Projekt die digitalen Ein- und Ausgänge nicht ausreichen, können Sie diese anlogen Eingangspins auch als digitale Ein- und Ausgänge verwenden.

Daneben befindet sich eine Reihe von Anschlüssen für verschiedene Arten der Stromversorgung für den Arduino. Damit können Sie auch von Ihnen gebaute elektronische Geräte mit Strom versorgen

Darüber hinaus verfügt der Arduino auch über eine Gleichstrombuchse. Hier können Sie Gleichspannungen zwischen 7 V und 12 V anschließen. Ein eingebauter Spannungsregler stellt die vom Arduino benötigten 5 V bereit. Der Arduino nimmt automatisch Strom an, der ihm über die USB-Buchse oder die Gleichstrombuchse zugeführt wird, je nachdem, welche an eine Stromquelle angeschlossen ist.

Die Arduino-IDE installieren

Der Arduino ist nicht gerade das, was man sich unter einem Computer vorstellt. Er hat kein Betriebssystem, keine Tastatur, keinen Bildschirm und keine Maus. Er führt immer nur ein Programm auf einmal aus, und Sie benötigen einen »richtigen« Computer, um ein Programm in den Flash-Speicher des Arduino zu laden. Allerdings können Sie den Arduino so oft umprogrammieren, wie Sie wollen (viele tausend Male).

Zur Programmierung des Arduino müssen Sie die Arduino-IDE auf Ihrem Computer installieren. Die plattformübergreifende Natur dieser Software – sie läuft auf Windows, Mac und Linux – ist einer der Hauptgründe für die große Beliebtheit des Arduino. Außerdem können Sie den Arduino über eine USB-Verbindung programmieren. Besondere Programmierhardware ist nicht erforderlich.

Um die Arduino-IDE für Ihre Plattform einzurichten, laden Sie die Software von der Arduino-Website herunter und folgen dann den Anweisungen auf der Website (*http://arduino.cc/en/Guide/HomePage*).

Windows- und Mac-Benutzer müssen außerdem die USB-Treiber für die Arduino-IDE installieren, damit diese mit dem Arduino kommunizieren kann.

Starten Sie die IDE, nachdem alles installiert ist. Abbildung 2–2 zeigt das Fenster der IDE.

Mit der Schaltfläche *Hochladen* (*Upload*) laden Sie den aktuellen Sketch auf den Arduino hoch. Dabei wird der in Textform vorliegende Programmcode in ausführbaren Code für den Arduino umgewandelt. Jegliche Fehler werden im Protokollbereich aufgeführt. Über die Schaltfläche *Prüfen* (*Verify*) untersuchen Sie den Code nur auf Fehler, ohne das Programm auf die Platine hochzuladen.

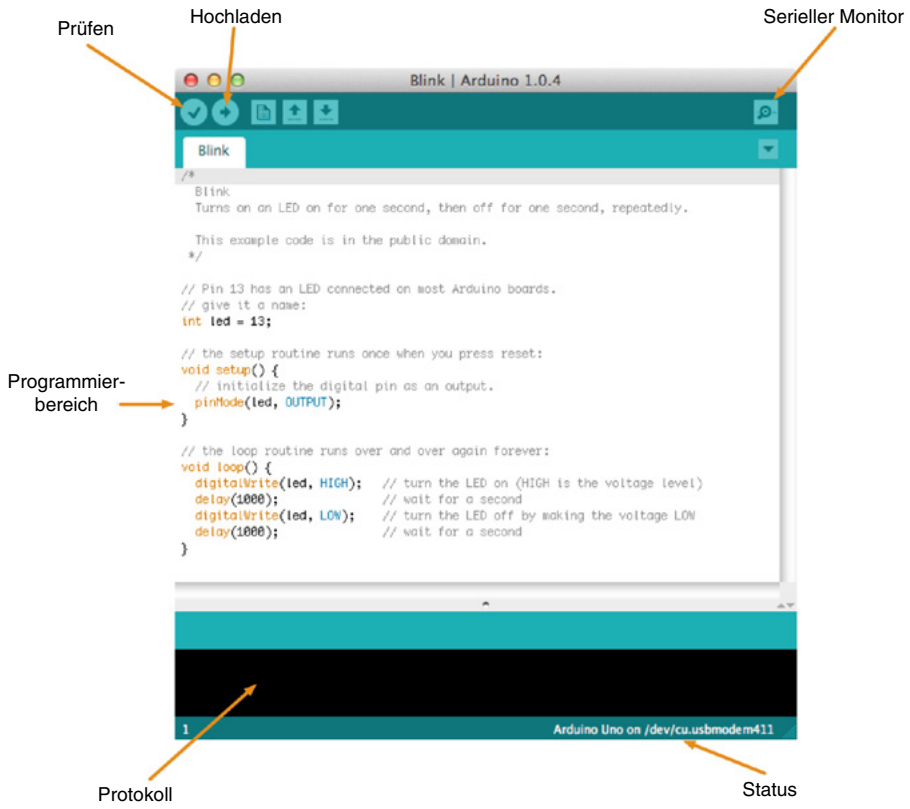

Abb. 2–2 *Die Arduino-IDE*

Mit der Schaltfläche oben rechts öffnen Sie das Fenster des seriellen Monitors, der eine Kommunikation mit dem Arduino ermöglicht. Den seriellen Monitor werden Sie in vielen Experimenten in diesem Buch verwenden, da er eine hervorragende Möglichkeit bietet, um Befehle vom Computer an den Arduino zu senden. Die Kommunikation ist beidseitig, was bedeutet, dass Sie nicht nur Textnachrichten an den Arduino senden, sondern auch Antworten von ihm empfangen können.

Der Statusbereich am unteren Bildschirmrand zeigt, welchen Arduino-Typ Sie verwenden und welcher serielle Port zur Übertragung eingesetzt wird, wenn Sie auf *Hochladen* klicken. In dem Statusbereich in Abbildung 2–2 ist die Art von Port zu sehen, die auf einem Mac oder Linux-Computer verwendet wird (`/dev/cu.usbmodem411`). Bei einem Windows-Computer wird hier `COM` gefolgt von einer Nummer angezeigt, die Windows beim Anschluss des Arduino zugewiesen hat.

Der Hauptteil der IDE ist der Programmierbereich, in dem Sie den Programmcode eingeben, den Sie auf den Arduino hochladen wollen.

Arduino-Programme werden *Sketche* genannt. Im Menü *Datei* können Sie Sketche öffnen und speichern wie die Dokumente in einer Textverarbeitung. Außerdem verfügt es über das Untermenü *Beispiele*, aus dem Sie die mitgelieferten Beispielsketche laden können.

Sketche hochladen

Um den Arduino zu testen und sicherzustellen, dass die Arduino-IDE korrekt installiert ist, klicken Sie auf *Datei* > *Beispiele* > *01. Basics* (in Abb. 2–2 ist der Beispielsketch *Blink* geladen).

Schließen Sie den Arduino über ein USB-Kabel an Ihren Computer an. Die Strom-LED des Arduino sollte aufleuchten, sobald die Verbindung hergestellt ist, und auch einige andere LEDs beginnen jetzt zu flackern.

Als Nächstes müssen Sie der IDE mitteilen, welches Platinenmodell programmiert werden soll (Arduino Uno) und über welchen seriellen Port die Verbindung läuft. Zur Angabe der Platine klicken Sie im Menü auf *Werkzeuge* > *Platine* und wählen dann den Arduino Uno aus der Liste aus.

Zur Angabe des seriellen Ports wählen Sie *Werkzeuge* > *Port*. Wenn Sie einen Windows-Computer verwenden, haben Sie wahrscheinlich nicht viele Auswahlmöglichkeiten, sondern sehen nur die Option *COM4*. Auf einem Mac oder Linux-Rechner wird im Allgemeinen eine Vielzahl von USB-Geräten aufgeführt. Es kann schwierig werden, den Anschluss für den Arduino herauszufinden.

Gewöhnlich beginnt der Name des richtigen Ports mit `dev/ttyusbmodem`*NNNN*, wobei *NNNN* eine Zahl ist. In Abbildung 2–3 sehen Sie die Portauswahl für den an meinen Mac angeschlossenen Arduino.

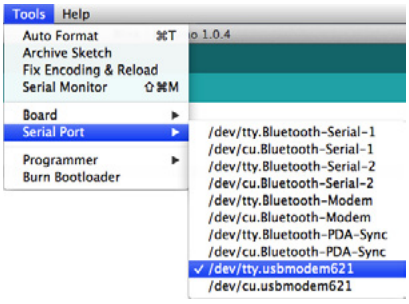

Abb. 2–3 *Den seriellen Anschluss des Arduino auswählen*

Wenn Ihr Arduino in der Liste nicht auftaucht, liegt das meistens an einem Problem mit den USB-Treibern, weshalb Sie versuchen sollten, sie neu zu installieren.

Jetzt sollte alles bereit sein, um den Sketch auf den Arduino hochzuladen. Klicken Sie auf die Schaltfläche *Hochladen*. Im Protokollbereich erscheint eine

Meldung, und die LEDs TX und RX auf der Platine flackern, während das Programm hochgeladen wird.

Wenn der Vorgang abgeschlossen ist, sehen Sie eine Meldung wie die in Abbildung 2–4.

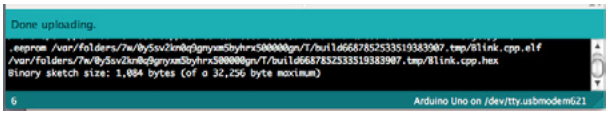

Abb. 2–4 *Ein Sketch wurde erfolgreich hochgeladen.*

Diese Meldung besagt, dass der Sketch hochgeladen wurde und 1084 Bytes der auf dem Arduino verfügbaren 32.256 Bytes belegt.

Nachdem der Sketch hochgeladen wurde, beginnt die eingebaute LED L auf dem Arduino langsam zu blinken. Das ist genau das, was das Programm *Blink* tun soll.

Der Code zu diesem Buch

Die gesamte Software für die Projekte – sowohl die Arduino-Sketche als auch die Python-Programme für den Raspberry Pi – steht auf der GitHub-Seite zu diesem Buch auf *https://github.com/simonmonk/make_action* zur Verfügung.

Um die Dateien auf Ihren Mac, Linux- oder Windows-Computer herunterzuladen, klicken Sie auf den Link *Download ZIP* unten rechts auf der GitHub-Seite.

Dadurch laden Sie eine ZIP-Datei herunter, die Sie auf dem Desktop oder an einem anderen gut zugänglichen Speicherort ablegen können. Wenn Sie das Archiv entkomprimieren, wird es in das Verzeichnis *make_action-master/* entpackt. Den Arduino-Code finden Sie im Verzeichnis *arduino/*. Darin gibt es wiederum die beiden Unterverzeichnisse *experiments/* und *projects/*.

Jedes Experiment oder Projekt hat wiederum ein eigenes Verzeichnis. Darin befindet sich gewöhnlich jeweils eine einzige Datei, bei der es sich um das eigentliche Programm handelt. Beispielsweise gibt es in dem Verzeichnis *experiments/* das Verzeichnis *ex_01_basic_motor_control/* mit der Datei *basic_motor_control.ino*. Wenn Sie die Arduino-IDE bereits installiert haben und diese Datei aufrufen, wird sie in der IDE geöffnet.

Alternativ können Sie auch die Ordner *experiments* und *projects* in Ihren Arduino-Sketchbook-Ordner kopieren, der den Namen *Arduino/* trägt und sich in Ihrem üblichen Dokumentenordner befindet (also *Eigene Dokumente* auf Windows oder *Dokumente* auf dem Mac).

Wenn Sie die Dateien in das Verzeichnis *sketchbook/* kopiert haben, können Sie sie öffnen, indem Sie in der Arduino-IDE auf *Datei* > *Sketchbook* klicken.

Programmierleitfaden

Dieser Abschnitt gibt Ihnen einen Überblick über die wichtigsten Befehle, damit Sie die in diesem Buch verwendeten Sketche verstehen können. Wenn Sie die Programmiersprache Arduino C erlernen wollen, sollten Sie sich mein Buch *Programming Arduino: Getting Started with Sketches* (Tab Books, 2012) zulegen.

Setup und loop

Sämtliche Arduino-Sketche müssen zwei grundlegende *Funktionen* enthalten (Programmcodeeinheiten, die eine bestimmte Aufgabe ausführen): `setup()` und `loop()`. Um zu sehen, wozu sie benötigt werden, schauen wir uns das Beispiel *Blink*, das wir auf den Arduino hochgeladen haben, genauer an.

```
int led = 13;
// Einrichtungsroutine, die einmal ausgeführt wird, wenn Sie Reset drücken
void setup() {
   // Initialisiert den LED-Pin als Ausgang
   pinMode(led, OUTPUT);
}

// Die Schleife wird endlos wiederholt
void loop() {
   digitalWrite(led, HIGH);   // Schaltet die LED ein (Spannung HIGH)
   delay(1000);               // Wartet eine Sekunde lang
   digitalWrite(led, LOW);    // Schaltet die LED aus (Spannung LOW)
   delay(1000);               // Wartet eine Sekunde lang
}
```

Sie sehen hier eine ganze Menge Text, dem ein doppelter Schrägstrich vorausgeht (//). Dies sind *Kommentare*. Dabei handelt es sich nicht um ausführbaren Programmcode, sondern nur um eine Beschreibung dessen, was an der betreffenden Stelle im Sketch passiert.

Wie Sie schon in den Kommentaren lesen können, werden die Codezeilen in der Funktion `setup()` nur einmal ausgeführt (genauer gesagt, jedes Mal, wenn der Arduino an eine Stromquelle angeschlossen oder die Reset-Taste gedrückt wird). In dieser Funktion geben Sie alle Aufgaben an, die zum Programmstart einmal erledigt werden müssen. Beim Programm *Blink* hat der Code in `setup()` nur eine Aufgabe, nämlich den LED-Pin als Ausgang festzulegen.

Die Befehle in der Funktion `loop()` dagegen werden immer und immer wieder ausgeführt. Wenn die letzte Zeile in `loop()` abgearbeitet ist, wird die Ausführung wieder mit der ersten Zeile fortgesetzt.

Wir haben bisher übersprungen, was die Befehle in den Funktionen `setup()` und `loop()` im Einzelnen tun, aber keine Sorge, das sehen wir uns gleich an.

Variablen

Variablen sind eine Möglichkeit, um Werten Namen zu geben. In der ersten Zeile von *Blink* wird der Pin 13 mit der Bezeichnung `led` versehen:

```
int led = 13;
```

Dadurch wird eine `int`-Variable namens `led` erstellt und ihr ein Anfangswert von 13 zugewiesen, da der Arduino-Pin, mit dem die LED L verbunden ist, die Nummer 13 trägt. Das Wort `int` ist eine Abkürzung für *Integer* und bedeutet eine ganze Zahl (ohne Dezimalstellen.

Es ist zwar nicht unbedingt erforderlich, für jeden verwendeten Pin einen Variablennamen zu vergeben, doch es ist eine gute Vorgehensweise, denn dadurch kann man leichter erkennen, wozu die einzelnen Pins dienen. Wenn Sie später einen anderen Pin nutzen möchten, müssen Sie dann außerdem nur den Wert der Variablen an einer einzigen Stelle ändern, nämlich bei ihrer Definition.

In einigen anderen Sketchen in diesem Buch werden Variablen, die einen bestimmten Pin festlegen, mit dem Schlüsselwort `const` eingeleitet:

```
const int led = 13;
```

Dadurch wird der Arduino-IDE mitgeteilt, dass die Variable in Wirklichkeit gar keine Variable ist, sondern eine Konstante: Der Wert von `led` beträgt 13 und kann sich nicht ändern. Durch die Zuweisung von Werten auf diese Weise können Sketche ein wenig kleiner gestaltet und schneller ausgeführt werden. Diese Vorgehensweise wird allgemein als eine gute Angewohnheit angesehen.

Digitale Ausgänge

Der Sketch *Blink* ist auch ein gutes Beispiel dafür, wie Sie einen Pin als *digitalen Ausgang* festlegen. Nach der Definition als `led` wird Pin 13 in der folgenden Zeile der Funktion `setup()` als Ausgang eingerichtet:

```
pinMode(led, OUTPUT);
```

Da dies nur einmal geschehen muss, steht dieser Befehl in der Funktion `setup()`. Wenn ein Pin erst einmal als Ausgang eingerichtet ist, bleibt er ein Ausgang, bis wir etwas anderes befehlen.

Damit die LED blinkt, muss sie wiederholt ein- und ausgeschaltet werden. Daher muss der Code für diesen Vorgang in der Schleife `loop()` platziert werden:

```
digitalWrite(led, HIGH);   // Schaltet die LED ein (Spannung HIGH)
delay(1000);               // Wartet eine Sekunde lang
digitalWrite(led, LOW);    // Schaltet die LED aus (Spannung LOW)
delay(1000);               // Wartet eine Sekunde lang
```

Die Funktion `digitalWrite()` nimmt zwei *Parameter* entgegen (in Klammern und durch ein Komma getrennt). Der erste legt fest, auf welchen Arduino-Pin geschrieben wird, und der zweite nennt den zu schreibenden Wert. Der Wert `HIGH` setzt den Ausgang auf 5 V, wodurch die LED eingeschaltet wird, und der Wert `LOW` setzt ihn auf 0 V und schaltet die LED damit aus.

Die Funktion `delay()` legt eine Pause von so vielen Millisekunden ein, wie ihr Parameter angibt. Hier wird durch den Wert `1000` festgelegt, dass die beiden `delay()`-Funktionen das Programm jeweils für eine Sekunde anhalten sollen.

Im Abschnitt »Experiment: Eine LED steuern« in Kapitel 4 lassen Sie nicht die eingebaute LED blinken, sondern eine externe LED, die an einen der digitalen Ausgänge angeschlossen ist.

Digitale Eingänge

Da es in diesem Buch hauptsächlich um Ausgänge geht, verwenden wir meistens die Funktion `digitalWrite()`. Allerdings sollten Sie sich auch mit den digitalen Eingängen auskennen, über die Sie Schalter und Sensoren an den Arduino anschließen können.

Mit der Funktion `pinMode()` können Arduino-Pins auch als Eingänge eingerichtet werden. Im folgenden Beispiel zeige ich Ihnen das anhand von Pin 7. Natürlich können Sie statt `7` auch einen Variablennamen verwenden.

Wie bei den Ausgängen müssen Sie auch den Modus eines Eingangspins nur in seltenen Fällen ändern, während der Sketch läuft, weshalb auch die Definition von Eingangspins gewöhnlich in der Funktion `setup()` erfolgt:

```
pinMode(7, INPUT)
```

Wenn ein Pin als Eingang festgelegt ist, können Sie messen, ob die an ihm anliegende Spannung näher bei 5 V (`HIGH`) oder näher bei 0 V (`LOW`) liegt. In dem folgenden Beispiel wird die LED eingeschaltet, wenn der Eingang zum Zeitpunkt der Messung `HIGH` ist. (Da es in dem Code keinen Befehl gibt, der die LED wieder ausschaltet, leuchtet sie nach dem Einschalten fortgesetzt weiter.)

```
void loop()
{
   if (digitalRead(7) == HIGH)
   {
      digitalWrite(led, HIGH)
   }
}
```

Jetzt wird der Code etwas komplizierter. Gehen wir ihn daher Zeile für Zeile durch.

In der zweiten Zeile steht eine öffnende geschweifte Klammer {. Manchmal wird sie in dieselbe Zeile geschrieben wie `loop()`, manchmal in die darauf folgende Zeile. Das ist Geschmackssache und hat keinerlei Auswirkungen auf die Codeausführung. Diese Klammer kennzeichnet den Beginn eines Codeblocks, der wiederum mit einer schließenden geschweiften Klammer endet, also mit }. Dadurch werden alle Codezeilen gruppiert, die zur Funktion `loop()` gehören.

In der ersten dieser Zeilen steht eine `if`-Anweisung. Unmittelbar auf das `if` folgt die Bedingung, in diesem Fall `(digitalRead(7) == HIGH)`. Das doppelte Gleichheitszeichen bedeutet, dass die Werte auf den beiden Seiten verglichen werden sollen. Wenn Pin 7 `HIGH` ist, dann wird der Codeblock in den geschweiften Klammern hinter dem `if` ausgeführt, anderenfalls nicht. Die Einrückung der zusammengehörigen öffnenden und geschweiften Klammern um denselben Wert macht es leichter zu erkennen, welche schließende Klammer } zu welcher öffnenden { gehört.

Den Code, der ausgeführt werden soll, wenn die Bedingung wahr ist, kennen Sie bereits: Es ist der `digitalWrite`-Befehl, mit dem die LED eingeschaltet wird.

Der Beispielcode aus dem vorhergehenden Abschnitt setzt voraus, dass der digitale Eingang definitiv `HIGH` oder `LOW` ist. Wenn Sie einen Schalter mit einem digitalen Eingang verbinden, kann er nichts anderes, als eine Verbindung schließen. Gewöhnlich werden Schalter so angebracht, dass der digitale Eingang beim Schließen des Kontakts mit Masse (0 V, GND) verbunden wird. Ist der Schalter geöffnet, so befindet sich der digitale Eingang in einem *schwebenden* Zustand, was bedeutet, dass er mit keiner Komponente elektrisch verbunden ist. Allerdings kann ein schwebender Eingang immer noch elektrisches Rauschen aus der umgebenden Schaltung aufnehmen, weshalb die Spannung am Pin zwischen `HIGH` und `LOW` schwankt. Um dieses unerwünschte Verhalten zu verhindern, wird normalerweise ein sogenannter Pullup-Widerstand verwendet (siehe Abb. 2–5).

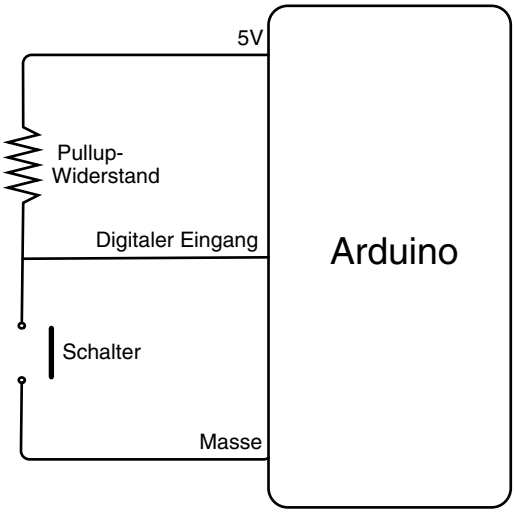

Abb. 2–5 *Verwendung eines Pullup-Widerstands an einem digitalen Eingang*

Wenn der Schalter geöffnet ist (wie in Abb. 2–5), verbindet der Widerstand den Eingangspin mit der Spannungsquelle und zieht die Spannung an dem Pin auf 5 V hoch. Wird die Taste gedrückt und damit der Schalter geschlossen, so wird das schwache Hochziehen des Eingangs außer Kraft gesetzt und der Eingang stattdessen mit Masse verbunden.

Sie könnten zwar auch selbst einen Widerstand anbringen, aber Arduino-Eingänge verfügen bereits über eingebaute Pullup-Widerstände von etwa 40 kΩ, die aktiviert werden, wenn Sie den Pin statt auf `INPUT` auf `INPUT_PULLUP` setzen. Der folgende Code zeigt, wie Sie den Pinmodus einrichten müssen, wenn Sie einen Schalter an einem digitalen Eingang anbringen und keinen externen Pullup-Widerstand wie in Abbildung 2–5 verwenden möchten:

```
pinMode(switchPin, INPUT_PULLUP);
```

Analoge Eingänge

An den analogen Eingangspins A0 bis A5 des Arduino können Sie Spannungen zwischen 0 V und 5 V messen. Anders als bei den digitalen Ein- und Ausgängen brauchen Sie für die analogen Eingänge keinen `pinMode`-Befehl in der Funktion `setup()`.

Um den Wert an einem analogen Eingang zu lesen, verwenden Sie `analogRead()`, wobei Sie den Namen des gewünschten Pins als Parameter übergeben. Im Gegensatz zu `digitalRead` gibt `analogRead` nicht `true` oder `false` zurück, sondern einen Zahlenwert zwischen 0 (0 V) und 1023 (5 V). Um die Zahlen in die anliegende Spannung umzurechnen, müssen Sie sie mit 5 multiplizieren und das Ergebnis

durch 1023 teilen oder die Zahl gleich durch 204,6 dividieren. Das folgende Beispiel zeigt, wie Sie in Arduino-Code einen Analogwert lesen und umrechnen:

```
int raw = analogRead(A0);
float volts = raw / 204.6;
```

Die Variable `raw` ist vom Typ `int` (Ganzzahl), da der Wert an einem analogen Eingang immer eine ganze Zahl ist. Zur Umrechnung des Rohwerts in eine Dezimalzahl brauchen wir jedoch eine Variable vom Typ `float` (Fließkommazahl).

An analoge Eingänge können Sie verschiedene Sensoren und Regler anschließen. In den Experimenten dieses Buches verwenden wir beispielsweise einen Fotowiderstand als Lichtsensor (siehe Abschnitt »Projekt: Arduino-Bewässerungsanlage für Zimmerpflanzen« in Kapitel 7) und einen variablen Widerstand (für den Getränkekühler mit Thermostat in Kapitel 12).

Analoge Ausgänge

Mit digitalen Ausgängen können Sie Bauteile (z. B. eine LED) nur ein- und ausschalten, aber analoge Ausgänge bieten Ihnen die Möglichkeit, inkrementell zu steuern, wie viel Leistung einer Komponente zugeführt werden soll. Dadurch können Sie beispielsweise die Helligkeit einer LED oder die Drehzahl eines Motors regeln. Diese Möglichkeit werden wir in den Experimenten in diesem Buch noch häufig nutzen.

Auf dem Arduino Uno können nur die Pins D3, D5, D6, D9, D10 und D11 als analoge Ausgänge verwendet werden. Sie sind auf dem Arduino mit einer kleinen Tilde (~) neben der Nummer gekennzeichnet.

Um einen analogen Ausgang zu steuern, verwenden Sie die Funktion `analogWrite()` mit einer Zahl zwischen 0 und 255 als Parameter. Der Wert 0 steht für 0 V, also eine völlige Abschaltung. Bei 255 dagegen ist der Pin voll eingeschaltet.

Es drängt sich die Vorstellung auf, dass ein analoger Ausgang in der Lage wäre, unterschiedliche Spannungen zwischen 0 V und 5 V zu erzeugen. Wenn Sie ein Voltmeter zwischen einem analogen Ausgang und Masse anlegen und den Parameter von `analogWrite()` ändern, scheint sich die Spannung tatsächlich zu ändern. In Wirklichkeit aber läuft hier ein komplizierterer Vorgang ab. Diese Ausgänge greifen auf die *Pulsweitenmodulation* (PWM) zurück. Abbildung 2–6 zeigt, was wirklich vor sich geht.

Ein analoger Ausgang generiert 490 Pulse pro Sekunde mit variabler Pulsweite (mit Ausnahme von D5 und D6, die 980 Pulse pro Sekunde liefern). Je länger der Puls auf `HIGH` bleibt, umso mehr Leistung wird an den Ausgang geliefert und umso heller leuchtet die LED bzw. umso schneller dreht sich der Motor.