Harold Timmis

Know-how ist blau. Arduino™

Apress®

Arduino™ in der Praxis

> Arduino™-Projekte mit Bluetooth steuern

> Arduino™-Projekte mit Xbox-Controller steuern

> Mit Fehlermeldungen via GSM-Messaging arbeiten

Die wichtigsten Anleitungen zur Arduino™-Programmierung

FRANZIS

Harold Timmis

Arduino™ in der Praxis

Harold Timmis

Arduino™ in der Praxis

Mit 216 Abbildungen

Bibliografische Information der Deutschen Bibliothek

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

Alle Angaben in diesem Buch wurden vom Autor mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. Der Verlag und der Autor sehen sich deshalb gezwungen, darauf hinzuweisen, dass sie weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen können. Für die Mitteilung etwaiger Fehler sind Verlag und Autor jederzeit dankbar. Internetadressen oder Versionsnummern stellen den bei Redaktionsschluss verfügbaren Informationsstand dar. Verlag und Autor übernehmen keinerlei Verantwortung oder Haftung für Veränderungen, die sich aus nicht von ihnen zu vertretenden Umständen ergeben. Evtl. beigefügte oder zum Download angebotene Dateien und Informationen dienen ausschließlich der nicht gewerblichen Nutzung. Eine gewerbliche Nutzung ist nur mit Zustimmung des Lizenzinhabers möglich.

Original English-language edition published by Apress, Inc. © 2011.
German-language edition © 2012 Franzis Verlag GmbH, 85540 Haar bei München.
All rights reserved. Arduino™ ist ein eingetragenes Markenzeichen von Arduino LLC und den damit verbundenen Firmen.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien. Das Erstellen und Verbreiten von Kopien auf Papier, auf Datenträgern oder im Internet, insbesondere als PDF, ist nur mit ausdrücklicher Genehmigung des Verlags gestattet und wird widrigenfalls strafrechtlich verfolgt. Die meisten Produktbezeichnungen von Hard- und Software sowie Firmennamen und Firmenlogos, die in diesem Werk genannt werden sind in der Regel gleichzeitig auch eingetragene Warenzeichen und sollten als solche betrachtet werden. Der Verlag folgt bei den Produktbezeichnungen im Wesentlichen den Schreibweisen der Hersteller.

Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: GGP Media GmbH, Pößneck
Printed in Germany

ISBN 978-3-645-65132-5

Vorwort

Harold Timmis, der Autor des vorliegenden Werks, studierte technische Informatik am Florida Institute of Technology und sammelte dort seine ersten Erfahrungen mit LabVIEW und Arduino. Als Projektmitarbeiter im Bereich Zugtechnik hat er seine Kenntnisse in den Bereichen LabVIEW, Arduino, Datenerfassung und Kontrolltheorie weiter vertieft. Seit 3 Jahren beschäftigt er sich mit LabVIEW und Arduino. In dieser Zeit hat er beruflich an zahlreichen LabVIEW-Projekten mitgewirkt und diverse Arduino-Projekte in seiner Freizeit realisiert.

Das Buch richtet sich an Studenten, Hobbybastler und Ingenieure gleichermaßen. Anhand verschiedener Projekte wird der praktische Einstieg in die Arbeit mit Arduino ermöglicht. Jeder findet heraus, wie er/sie die Plattform am besten für sich nutzen kann. Für den gelungenen Einstieg in die Programmierung mit Arduino unter Verwendung spezifischer Hardware-Komponenten wurden mehrere Einführungsprojekte konzipiert.

Das Werk enthält umfassende Informationen zum Thema Arduino und bietet weit mehr als nur die üblichen Anwendungsbeispiele mit LEDs. Eine Vielzahl von Peripheriegeräten und Technologien wie ein Ultraschallsensor, ein Xbox®-Controller und ein Bluetooth-Modul kommen zum Einsatz. Auch der Laie lernt in diesem Buch, Prozesse nachzuvollziehen, die auch für zukünftige (Nicht-Arduino-)Projekte hilfreich sein können.

Dieses Buch ist für jeden geeignet, der etwas über Arduino und die praktische Arbeit damit lernen möchte. Vorausgesetzt werden grundlegende Kenntnisse im Breadboarding und Löten.

In diesem Buch lernen Sie, wie Sie verschiedene Hardware-Komponenten und Technologien (Bluetooth, GPS, GSM) mit Arduino für sich nutzen können. Sie lernen, den Ablauf Ihrer Arduino-Projekte zu optimieren und sparen so Zeit und Ressourcen.

Mein Dank gilt den Fachkorrektoren Coleman Sellers und Andreas Wischer für Ihr ausführliches Feedback.

Christian Schweinfurth

Inhaltsverzeichnis

1 Der Engineering-Prozess

In diesem Kapitel geht es um den eigentlichen Entwicklungsprozess und die Möglichkeiten der Prototypoptimierung. Vorrang hat vor allem die Vermeidung von Hard- und Software-Problemen sowie die Einhaltung eines festen Zeitplans. Die Realisierung der Projekte in diesem Buch erfolgt nach einem vorgegebenen Ablauf, der im Folgenden als *Engineering-Prozess* bezeichnet wird. Dieser setzt sich folgendermaßen zusammen:

- Zusammenstellen der Anforderungen
- Anlegen einer Checkliste
- Hardware
- Konfigurieren der Hardware
- Schreiben der Software
- Debuggen der Arduino-Software
- Hardware-Fehlerbehebung
- Fertiger Prototyp

Aus dieser Zusammenfassung lässt sich die Effektivität dieser Vorgehensweise für die Prototyperstellung bereits erahnen. Deshalb wird der Prozess für alle Arduino-Projekte in diesem Buch konsequent eingehalten.

Arduino ist ein frei konfigurierbarer Mikrocontroller und Open Source, d. h., der Quellcode ist frei verfügbar und die integrierte Entwicklungsumgebung (IDE) zum Schreiben der Software kostenlos. Das gilt auch für einen Großteil der verfügbaren Ressourcen. Lediglich der Arduino-Mikrocontroller selbst muss käuflich erworben werden. Für den Arduino-Mikrocontroller finden Sie zahlreiche Ressourcen im Web und in Büchern, z. B. Tutorials zu einzelnen Problemstellungen. Für den Einstieg sind die Seiten *www.arduino.cc* und *http://tronixstuff.wordpress.com/tutorials/* einen Besuch wert. Dieses Buch ist wesentlich mehr als nur eine Sammlung von Tutorials. Hier lernen Sie, den Engineering-Prozess konsequent anzuwenden, um Ihre Projekte übersichtlicher, effizienter und zuverlässiger zu gestalten.

1.1 Zusammenstellen der Hardware

Zunächst gilt es, sich mit den wichtigsten Einzelkomponenten und Materialien vertraut zu machen. Für den erfolgreichen Abschluss aller Projekte in diesem Buch benötigen Sie folgende Hardware-Komponenten:

- *Arduino Duemilanove* oder *UNO*: Beide Modelle verfügen über mehrere E/A-Ports für Sensoren und Motoren. Über diese E/As erfolgt die Steuerung und Programmablaufprotokollierung der verschiedenen Projekte in diesem Buch.

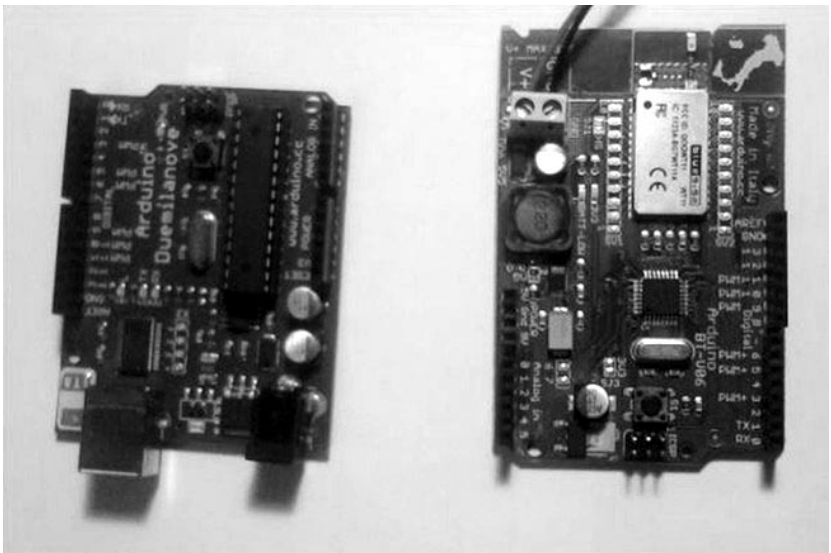

Bild 1.1: Arduino UNO (links) und Duemilanove (rechts)

- *ArduinoBT* oder *Bluetooth Mate Silver*: Es empfiehlt sich, hier das Bluetooth-Mate-Silver-Modul zu verwenden. So können Sie aus Ihrem Arduino Duemilanove oder UNO ein ArduinoBT machen – und das zum halben Preis. Außerdem fehlt dem ArduinoBT ein 3,3-V-Ausgang, d. h., Sie müssten für einen 3,3-V-Ausgang dem Arduino-Board weitere Schaltungen hinzufügen. Einen solchen Ausgang benötigen Sie für Kapitel 6 (Home-Engineering) in diesem Buch.

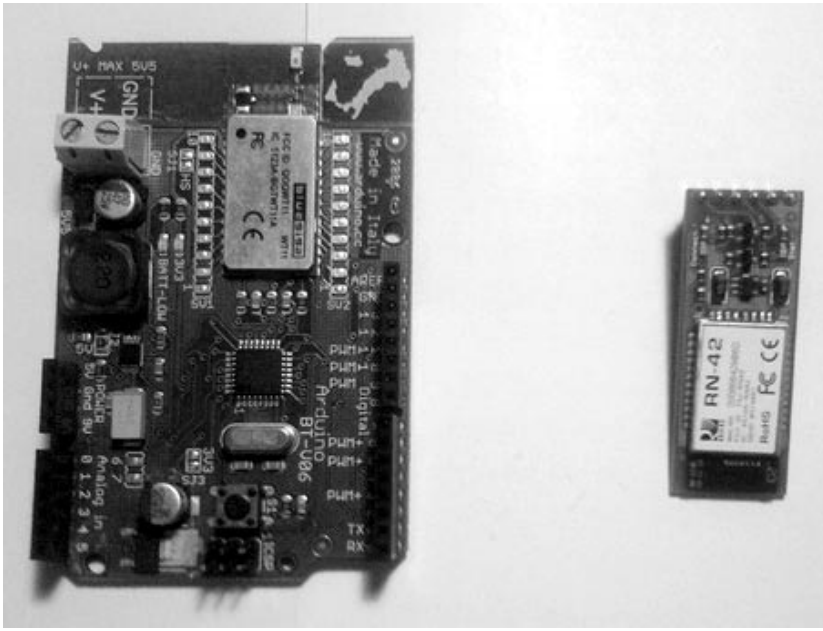

Bild 1.2: ArduinoBT (links) und Bluetooth-Mate-Silver-Modul (rechts)

- *Lötfreie Steckplatine*: Eine weitere wichtige Hardware-Komponente ist die lötfreie Steckplatine zur Umsetzung Ihrer Schaltungen. Für dieses Buch benötigen Sie eine mittelgroße lötfreie Steckplatine, die sowohl in der Entwurfs- als auch in der Fehlerbehebungsphase zum Einsatz kommt.

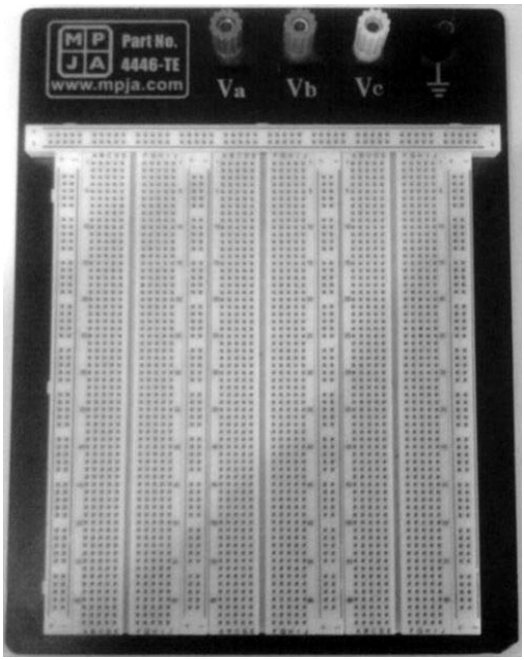

Bild 1.3: Lötfreie Steckplatine

- *Draht*: Für die Projekte in diesem Buch wird viel Draht benötigt. Drahtbrücken erhalten Sie in jedem gut sortierten Elektrofachgeschäft.
- *Arduino-Shields*: In diesem Buch werden verschiedene Shields, z. B. Motor-, GPS-, GSM- und LCD-Shields verwendet.

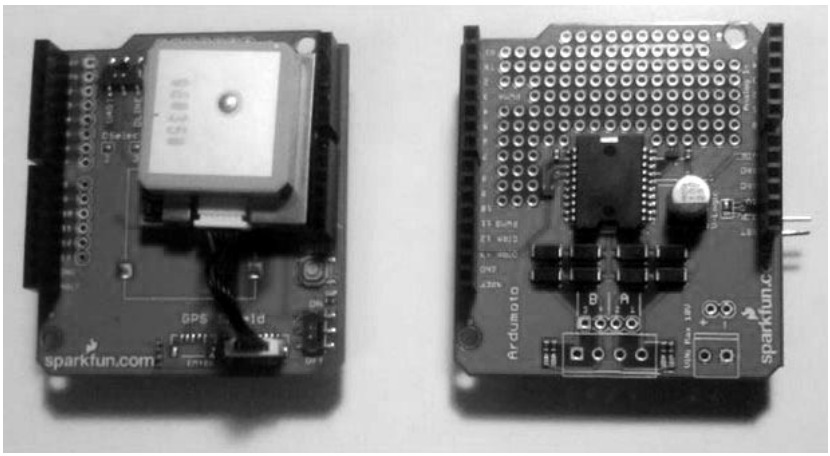

Bild 1.4: Arduino-Shields-: GPS-Shield (links), Motor-Shield (rechts)

- *Motor-Shield*: Dieses Shield dient der Steuerung von Motoren bis 18 V. Das Motor-Shield ist mit einer oberflächenmontierten H-Bridge zur Verwendung von Motoren mit einer höheren Versorgungsspannung/zur Steuerung von zwei Motoren ausgestattet. Weitere Informationen über H-Brücken finden Sie in Kapitel 3.
- *GPS-Shield*: Mit diesem Shield können Sie GPS-Positionsdaten abrufen. Es verwendet den NMEA(National Marine Electronics Association)-Standard, über den sich eine Vielzahl von Daten abrufen lässt, z. B. Längengrad, Breitengrad, GPS-Fix-Verfügbarkeit, GPS-Fix-Typ, Zeitstempel und Signal-Rausch-Verhältnis. Weitere Informationen über GPS-Systeme finden Sie in Kapitel 5.
- *GSM-Shield*: Mit diesem Shield können Sie die Leistungsfähigkeit des GSM-Standards nutzen, um Textnachrichten über große Distanzen hinweg zu versenden und zu empfangen. Das Shield verwendet als Standardprotokoll das GSM-Protokoll.
- *LCD-Shield*: Mit diesem Shield erweitern wir unseren Roboter um ein bildgebendes Verfahren und hauchen ihm dadurch Leben ein. Mit dem LCD-Shield können Sie auch eine eigene Benutzerschnittstelle für Ihren Roboter oder für jedes beliebige andere Projekt erstellen.
- *Sensoren*: Sensoren machen Ihre Projekte »lebendig«. Beispiele sind Bewegungsmelder, Ultraschall- und Temperatursensoren.

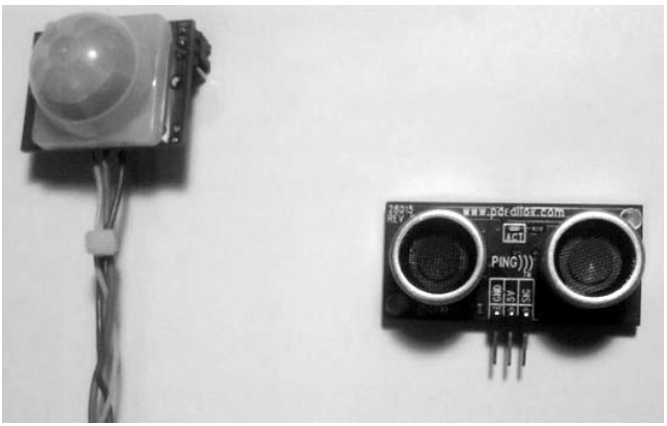

Bild 1.5: Bewegungsmelder (links) und Ultraschallsensor (rechts)

- *Bewegungsmelder*: Dieser Sensor eignet sich hervorragend für die Erkennung von Infrarotlicht- und Temperaturschwankungen, lässt sich aber auch gut für die Bewegungserkennung einsetzen.
- *Ultraschallsensor*: Ultraschallsensoren eignen sich für das Erkennen von Objekten in der unmittelbaren Umgebung. Der hier eingesetzte Ultraschallsensor ermittelt die Entfernung zu einem bestimmten Objekt über ein digitales Ping-Signal.
- *Temperatursensor*: Mit diesen Sensoren können Sie die Temperatur ermitteln. Dazu müssen Sie zuerst eine Zuordnung der Spannungs- und Temperaturwerte (Skalierung) vornehmen, die Sie aufzeichnen möchten. Weitere Informationen finden Sie in Kapitel 6.
- *Servos und Motoren:* Motoren und Servos bieten vielfältige Steuerungsmöglichkeiten.

Bild 1.6: Auswahl von Motoren

- *Diverse Bauelemente*: gängigste Komponenten elektronischer Schaltungen wie Widerstände, Kondensatoren, LEDs, Dioden und Transistoren.

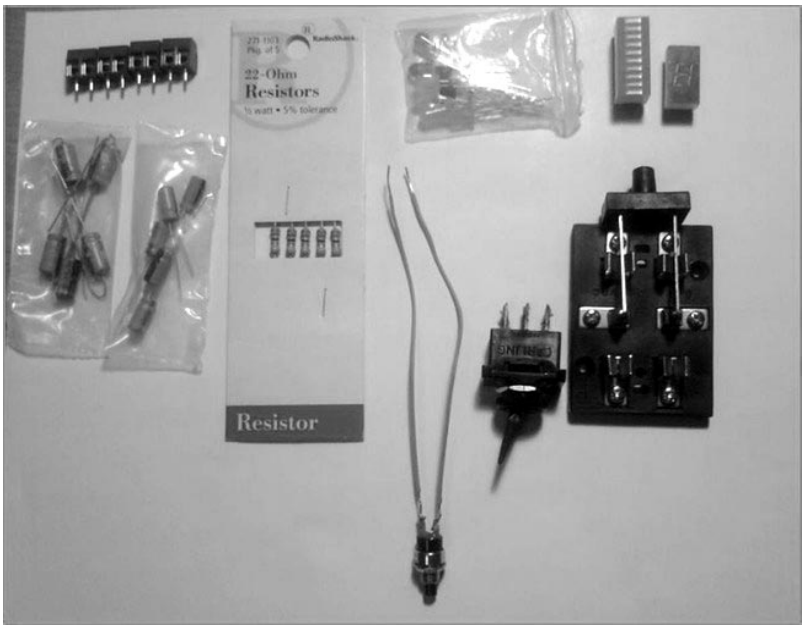

Bild 1.7: Diverse Bauelemente (Reihenklemmen, Kondensatoren, Widerstände, LEDs und Schalter)

1.2 Zusammenstellen der Werkzeuge

Sie benötigen eine Reihe von Werkzeugen, die nachfolgend aufgeführt sind.

- *Lötkolben*: Hiermit lassen sich Schaltungen miteinander verbinden.

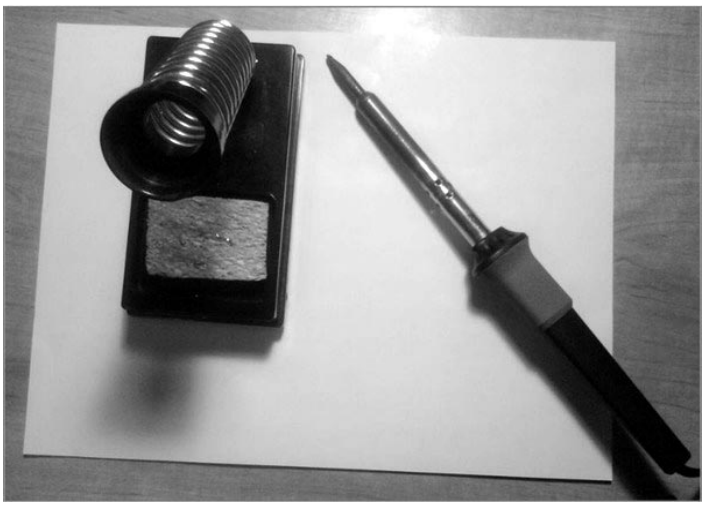

Bild 1.8: Lötkolben mit Halter

- *Lot*: Lot (Metalllegierung) ist der Werkstoff zum Verbinden von Schaltkreisen mithilfe eines Lötkolbens. Lot hat einen sehr geringen Schmelzpunkt.
- *Spitzzange*: Mit einer Spitzzange kann man Draht und Schaltungen in Position halten, Draht um Schaltungen wickeln usw.
- *Lupe*: Mit diesem Werkzeug hat man eine bessere Sicht auf die Schaltungen.
- *Seitenschneider*: Er dient dem Abtrennen von Drähten.
- *Abisolierzange*: Mit diesem Werkzeug entfernt man die Drahtisolierung.
- *Multimeter*: Hiermit kann man die Spannung (Wechsel- und Gleichstrom), die Stromstärke (Ampere) und den Widerstand (Ω) ablesen.
- *Wissenschaftlicher Taschenrechner*: Damit kann man verschiedene Berechnungen vornehmen (Widerstände nach ohmschem Gesetz, Spannungsteilung usw.)

Bild 1.9: Wichtige Werkzeuge von links nach rechts: Multimeter, Spitzzange, Seitenschneider (oben), Abisolierzange (unten)

1.3 Projekt: Blinkende LED

Der Engineering-Prozess dient der umfassenden Optimierung des gesamten Entwicklungsprozesses.

1.3.1 Zusammenstellen der Anforderungen

Wir gehen von dem Fall aus, dass Sie bei einem Kunden die Anforderungen für ein bestimmtes Projekt analysieren sollen. Dieser Teil des Engineering-Prozesses ist von entscheidender Bedeutung: Alle weiteren Schritte hängen davon ab, welche Entscheidungen Sie beim ersten Meeting treffen. Ihr Kunde möchte z. B. eine LED mit einer bestimmten Frequenz aufleuchten lassen, und Sie einigen sich darauf, für diesen Zweck den Arduino-Mikroprozessor zu verwenden. Das Intervall soll 100 ms betragen.

1.3.2 Anlegen einer Checkliste

Aus den Kundenanforderungen und der von Ihnen vorgeschlagenen Lösung erstellen Sie z. B. folgende einfache Checkliste:

- Hardware
  - Arduino-Board
  - LED
  - 9-V-Batterie
  - 9-V-Batterieanschluss
  - Widerstand (22 Ω)
- Software
  - Programm, das eine LED im Abstand von 100 ms aufleuchten lässt

Auch wenn es sich in diesem Fall um ein sehr einfaches Beispiel handelt, wird das Prinzip dieser Vorgehensweise für den Rest des Buchs beibehalten. Mit einer solchen Checkliste vermeiden Sie z. B., dass Ihr Kunde fortlaufend das Hinzufügen neuer Funktionen erwartet. Das könnte zum Problem werden, weil Sie zusätzliche Arbeitszeit ohne entsprechende Bezahlung auf ein Projekt verwenden würden, das sich möglicherweise endlos hinzieht. Dank der Checkliste wissen Sie und Ihr Kunde genau, was zu tun ist und auf was Sie sich geeinigt haben. Nach dem Anlegen der Checkliste können Sie ein Flussdiagramm erstellen, das Ihnen später dabei hilft, die Software zu debuggen.

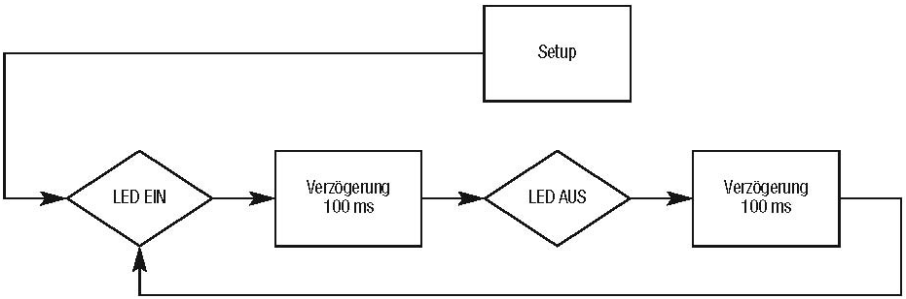

Bild 1.10: Programmablauf des Projekts »Blinkende LED«

1.3.3 Hardware

Als nächster Schritt des Engineering-Prozesses müssen Sie sicherstellen, dass Sie über die richtige Hardware verfügen. Welche Hardware Sie benötigen, sollten Sie beim Zusammenstellen der Anforderungen entscheiden. Alle Hardware-Komponenten müssen kompatibel sein, andernfalls müssen Sie ggf. einzelne Kompo-

nenten austauschen. Stimmen Sie sich dabei immer mit dem jeweiligen Unternehmen ab, für das Sie arbeiten.

1.3.4 Konfigurieren der Hardware

Je nach den Hardware-Anforderungen des jeweiligen Projekts kann die Konfiguration ganz unterschiedlich ausfallen. Als Beispiel dient hier die Hardware-Konfiguration für das Projekt »Blinkende LED«.

- Arduino-Board
- LED
- 9-V-Batterie
- 9-V-Anschluss

Für die Einrichtung der Hardware muss die LED an Digital-Pin 13 und den Masse-Anschluss des Arduino-Boards angeschlossen werden.

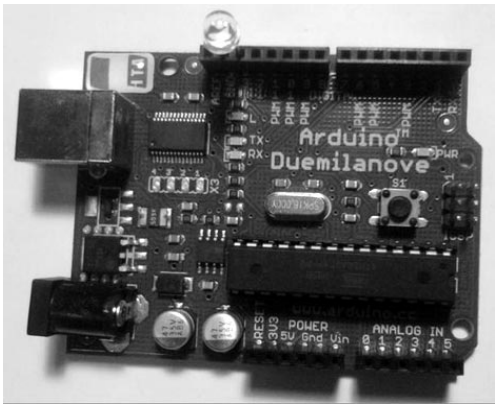

Bild 1.11: Einrichten der Hardware für das Projekt »Blinkende LED

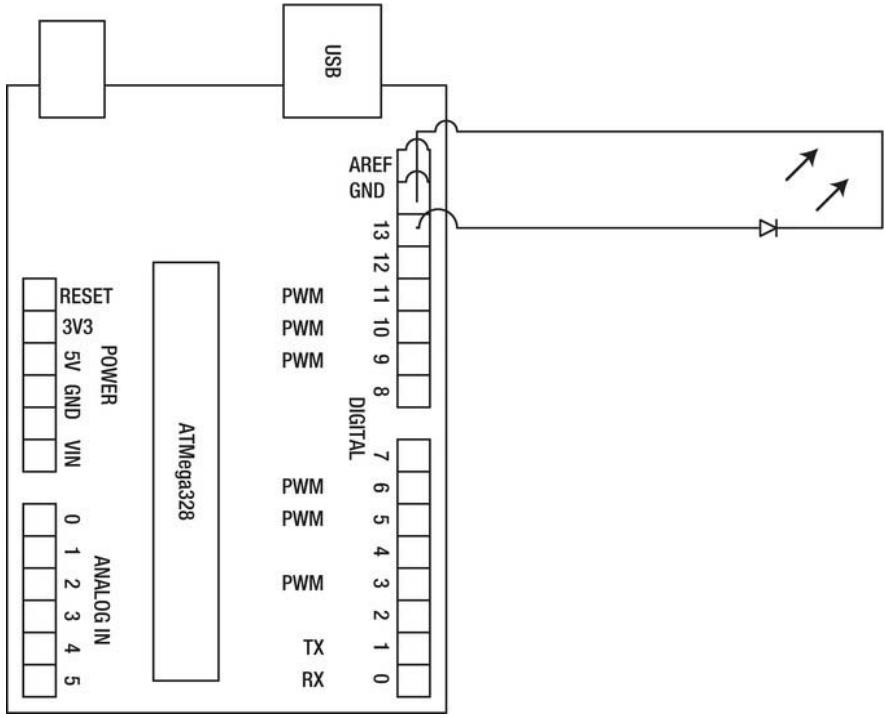

Bild 1.12: Schaltplan des Projekts »Blinkende LED«

Zuerst muss allerdings die Arduino-IDE installiert werden. Rufen Sie dazu die Adresse *www.arduino.cc/en/main/software* auf. Die Arduino-IDE ist unter Windows Vista, Windows 7, Mac OS X und Linux lauffähig. Nach dem Herunterladen der Arduino-IDE auf den Desktop liegen die Dateien im ZIP-Format vor und müssen entpackt werden. Nach dem Entpacken ist die Arduino-IDE installiert.

Nachdem Sie die Arduino-IDE auf dem Computer installiert haben, gilt es sicherzustellen, dass die IDE richtig konfiguriert ist. Öffnen Sie dazu die Arduino-IDE, und navigieren Sie zu *Tools/Serial Port*. Wählen Sie den seriellen Port aus, mit dem Ihr Arduino-Board verbunden ist. Wählen Sie anschließend *Tools/Boards*, und wählen Sie Ihr Arduino-Board aus (in diesem Buch verwenden wir das Modell Arduino Duemilanove ATmega328). Nachdem Sie die Hardware konfiguriert haben, können Sie die Software schreiben.

1.3.5 Schreiben der Software

Dieser Teil des Engineering-Prozesses ist von entscheidender Bedeutung. Ein Blick auf die Checkliste für das Projekt »Blinkende LED« hilft, um über den Funktionsumfang der Software zu entscheiden: Die LED soll in einem Intervall

von 100 ms aufleuchten. Die Software könnte demnach folgendermaßen aussehen:

```
// Code für blinkende LED (Intervall: 100 ms)

const int LEDdelay = 100;  // Verzögerungszeit

void setup()
{
     pinMode(13, OUTPUT);  // definiert Pin 13 als Ausgang
}

void loop()
{
     digitalWrite(13, HIGH);   // gibt HIGH-Bit an Pin 13 aus
     delay(LEDdelay);          // Verzögerung 100 ms
     digitalWrite(13, LOW);
     delay(LEDdelay)           // löst Syntaxfehler wegen eines
                               // fehlenden Strichpunkts aus

}
```

Hinweis
Wenn Sie versuchen, dieses Programm für Ihr Arduino-Board zu kompilieren, erhalten Sie eine Fehlermeldung. Das liegt an einem Syntaxfehler, den wir im nächsten Abschnitt beheben werden.

1.3.6 Debuggen der Arduino-Software

Das letzte Programm konnte wegen eines Syntaxfehlers nicht kompiliert werden. Ursache ist falsch formatierter Code, z. B. ein fehlender Strichpunkt wie in unserem Beispiel. Der überarbeitete Code sieht folgendermaßen aus:

```
// Code für blinkende LED (Intervall: 100 ms)

const int LEDdelay = 100;  // Verzögerungszeit

void setup()
{
     pinMode(13, OUTPUT);  // definiert Pin 13 als Ausgang
}

void loop()
{
     digitalWrite(13, HIGH);   // gibt HIGH-Bit an Pin 13 aus
     delay(LEDdelay);          // Verzögerung 100 ms
     digitalWrite(13, LOW);
```

```
    delay(LEDdelay);          // Strichpunkt jetzt vorhanden,
                              // Code kann kompiliert werden
}
```

Syntaxfehler gehören noch zu den harmloseren Fehlern. Die problematischsten Fehler sind *logische Fehler*. Hier wird der Code zwar kompiliert, aber Sie erhalten ein unerwartetes Ergebnis. Ein logischer Fehler liegt z. B. dann vor, wenn Sie ein Größer-als-Symbol (>) anstelle eines Kleiner-als-Symbols (<) verwenden. Bei einem Projekt mit mehreren Tausend Zeilen Code kann die Behebung eines solchen Fehlers ein nahezu unmögliches Unterfangen darstellen.

Hinweis
Im Projekt »Blinkende LED« läge ein logischer Fehler z. B. dann vor, wenn Sie für beide digitalWrite-Funktionen *digitalWrite(13, HIGH);* schreiben würden.

Logische Fehler in einem Arduino-Programm lassen sich am besten mithilfe eines Flussdiagramms ausfindig machen. Damit können Sie genau nachvollziehen, an welcher Stelle Werte abweichen.

1.3.7 Hardware-Fehlerbehebung

Erste Wahl zum Beheben von Hardware-Fehlern ist das Multimeter. Mit diesem Werkzeug lassen sich Hardware-Schäden bereits im Vorfeld verhindern. Wenn Sie z. B. mithilfe eines Multimeters feststellen, dass für das Projekt »Blinkende LED« die Stromversorgung den gewünschten Wert überschreitet, können Sie einen 22-Ω-Widerstand einbauen, um das Ausbrennen der LED zu verhindern.

1.3.8 Fertiger Prototyp

Nach dem Software Debugging und der Hardware-Fehlerbehebung sollten Sie jetzt einen fertigen Prototyp vorliegen haben, der unter normalen Bedingungen einwandfrei funktioniert. Das Projekt aus in diesem Kapitel war sehr einfach konzipiert. In den folgenden Kapiteln wird die Komplexität stetig zunehmen und der Engineering-Prozess eine immer wichtigere Rolle bei der Code-Optimierung spielen.

2 Arduino-Software-Entwicklung

Nachfolgend werden die verschiedenen Programmierelemente vorgestellt, auf die im Verlauf dieses Buchs immer wieder zurückgegriffen wird. Wenn Sie bereits Erfahrung mit der Programmierung in C haben, wird Ihnen das Programmieren mit Arduino dank der vielen Gemeinsamkeiten nicht schwerfallen. Falls Programmierung neu für Sie ist, erhalten Sie in diesem Kapitel eine Einführung in die wichtigsten Konzepte. Nur wenn Sie die Grundlagen der Arduino-Programmierung beherrschen, bleibt Ihr Code übersichtlich und nachvollziehbar. Außerdem werden Sie nachfolgend die wichtigsten Programmstrukturen erarbeiten, damit Sie sich später auf die Bibliotheken konzentrieren können. Bibliotheken sind Sammlungen von Klassen, Typen oder Funktionen, die über Schlüsselwörter aufgerufen werden können. Mit Bibliotheken können Sie Ihrem Programm vorgefertigte Funktionalität hinzufügen. Damit ermöglichen Bibliotheken das Prinzip der modularen Wiederverwendbarkeit von Code. Hier werden wir uns schwerpunktmäßig mit den Bibliotheken NewSoftSerial, LCD Library und TinyGPS befassen.

2.1 Erste Schritte mit setup und loop()

Ohne *setup()* und *loop()* kann kein Arduino-Programm ordnungsgemäß ausgeführt werden. Die Implementierung der beiden Funktionen erfolgt nach diesem Schema:

```
// Einfaches Arduino-Programm

void setup()
{
    // E/As hier einrichten
}

void loop()
{
    // Funktionalität
}
```

Unter *setup()* richten Sie E/A-Ports wie LEDs, Sensoren, Motoren und serielle Ports ein. Dieser Vorgang ist wichtig, denn um die Pins auf dem Arduino-Board nutzen zu können, müssen Sie die gewünschten Pins vorher »reservieren«.

loop() enthält den gesamten Code zur Steuerung der E/A-Ports. Hier legen Sie z. B. eine bestimmte Geschwindigkeit für Ihren Motor fest. Das Einrichten und Steuern der E/As wird in den nächsten Abschnitten näher erläutert.

Arduino-Programme enthalten außerdem diverse Unterfunktionen. Dabei handelt es sich um nützliche Sonderfunktionen, die Sie innerhalb von *loop()*-Anweisungen oder in den entsprechenden Unterfunktionen aufrufen können. Um eine Unterfunktion verwenden zu können, müssen Sie die Funktion zuerst am Anfang des Programms initialisieren. Diese erste Deklaration nennt sich *Funktionsprototyp*.

Beispiel:

```
// Funktionsprototyp
void delayLED();

void setup()
{

}

void loop()
{

}

 // Unterfunktion

void delayLED()
{
   // Code, der im Anschluss an die loop()-Struktur ausgeführt wird
}
```

2.1.1 Initialisieren von Variablen

Variablen sind die grundlegendsten Programmierbausteine. Mit ihnen lassen sich Daten zwischen einzelnen Programmbestandteilen übergeben. Alle Programme in diesem Buch verwenden Variablen. In der Arduino-Programmiersprache stehen verschiedene Variablentypen zur Verfügung.

Tabelle 2.1: Variablentypen

Name	*Wert*	*Wertebereich*
`char`	`'a'`	–128 bis 127
`byte`	`1011`	0 bis 255
`int`	`-1`	–32.768 bis 32.767
`unsigned int`	`5`	0 bis 65.535

Name	Wert	Wertebereich
long	512	-2.147.483.648 bis 2.147.483.647
unsigned long	3.000.000	0 bis 4.294.967.295
float	2,513	–3,4028235E+38 bis 3,4028235E+38
double	2,513	-3,4028235E+38 bis 3,4028235E+38

Nachdem Sie nun wissen, welche Variablentypen es gibt, gilt es, diese Variablen zu deklarieren. Für die Deklaration der Variablen müssen Sie wissen, in welchem *Bereich* sie verwendet werden können. Anschließend müssen Sie den Bereich angeben (*deklarieren*), der Ihren Anforderungen entspricht. Hier wird zwischen zwei grundsätzlichen Bereichen für Variablen unterschieden: lokal und global. Eine *lokale Variable* lässt sich nur innerhalb des für sie definierten Bereichs verwenden. Z. B. werden in einer Schleife die dort deklarierten Variablen nur innerhalb der Klammern verwendet, die Variablen sind also für diese Schleife lokal. Eine globale Variable kann von jedem beliebigen Ort des Programms aufgerufen werden. Zur Definition einer globalen Variable müssen Sie die Variable zu Beginn des Programms initialisieren. Das folgende Programm zeigt, wie Sie lokale und globale Variablen initialisieren:

```
// Variable initialisieren

int x;  // Diese Variable wird global deklariert und kann im
        // gesamten Programm aufgerufen werden.

void setup()
{

}

void loop()
{
    x = 1 + 2;  // weist x den Wert 3 zu
    for(int i; i <= 100; i++)
    {
       // i ist eine lokale Variable und kann nur innerhalb
       // dieser Schleife aufgerufen werden.
    }

}
```

Alle anderen Deklarationen erfolgen auf die gleiche Weise, solange Sie keine *Arrays* verwenden. Mit Arrays können Sie mehrere Werte vom selben Typ übergeben. Wenn Sie z. B. mehrere digitale Pins übergeben möchten, müssen Sie nicht jeden Pin einzeln deklarieren:

```
int pins[] = {13,9,8};
```

Sie sollten auch die Größe des Arrays deklarieren, wie im folgenden Beispiel:

```
const int NumOfPins = 3;
int pins[NumOfPins] = {13,9,8};
```

So können Sie auf die Informationen im Array zugreifen und anschließend an einen digitalen Pin oder eine andere Instanz übergeben.

Hinweis
Whitespacing bedeutet das Hinzufügen von Leerzeilen und Leerzeichen, um den Code lesbarer zu gestalten.

2.1.2 Bedingte Anweisungen

Mit bedingten Anweisungen steuern Sie den Programmfluss. Wenn z. B. ein Motor nur eingeschaltet werden soll, nachdem ein Knopf gedrückt wurde, können Sie das mit einer bedingten Anweisung umsetzen. Folgende bedingte Anweisungen sind für uns von Interesse: *if*, *if-elseif*, *if-else* und *switch*.

if-Anweisungen sind wichtige bedingte Anweisungen, die für nahezu jede boolesche Operation und verschiedene andere Zwecke verwendet werden können, z. B. zur Grenzwertbestimmung. Im Folgenden ein Beispiel für eine *if*-Anweisung:

```
int i;
if (i < 10)
{
    i++;
}
```

Sie können am Ende Ihrer *if*-Anweisung weitere *elseif*-Anweisungen hinzufügen, um weitere Bedingungen hinzuzufügen, oder Sie erstellen eine *if-elseif*-Anweisung:

```
int i;
if (i < 10)
{
     i++;
}
else if (i > 10)
{
     i--;
}
```

Ein praktisches Anwendungsbeispiel für eine bedingte Anweisung ist z. B. das Auslesen eines Potenziometerwerts:

```
potValue = analogRead(potPin);

if (potValue <= 500)
{
     digitalWrite(motorpin, 1);
}
else
{
     digitalWrite(motorpin, 0);
}
```

Hinweis
Sie müssen die Arduino-Pins im Vorfeld festlegen, bevor Sie die Pins in einer Schleife aufrufen.

Eine *switch*-Anweisung macht den Code lesbarer – für den Fall, dass Sie mehrere Bedingungen verwenden. Nachfolgend ein Beispiel für eine *switch*-Anweisung:

```
switch (potValue){
case 500;
     digitalWrite(motorPin,1);
case 501;
     digitalWrite(ledPin,1);
     break;
default:
     digitalWrite(motorPin,0);
     digitalWrite(ledPin,0);
```

Wenn in diesem Beispiel *potValue* 500 beträgt, wird der Motor eingeschaltet. Wenn *potValue* 501 beträgt, wird eine LED eingeschaltet. Der Standardfall (*default*-Anweisung) tritt ein, wenn *potValue* weder 500 noch 501 beträgt. In diesem Fall bleiben Motor und LED ausgeschaltet.

2.2 Arbeiten mit Schleifen

Schleifen lassen sich vielfältig einsetzen, z. B. um redundanten Code zu entfernen oder Arrays zu durchlaufen. Für uns von Interesse sind die Schleifen *for*, *while* und *do…while*. Mit diesen Schleifen kann Code durchlaufen werden, während eine Bedingung wahr ist (oder falsch, unter bestimmten Umständen).

for-Schleife: Mit dieser Schleife können Sie einen Code-Block mit einer bestimmten Anzahl von Wiederholungen ausführen.

Eine *for*-Schleife weist folgende allgemeine Struktur auf:

```
for(int i = 0; i <= 10; i++)
{
     // Anweisungen hier einfügen
}
```

Ein praktisches Beispiel für die Anwendung einer *for*-Schleife ist die Aktualisierung mehrerer *pinMode*-Einstellungen:

```
int pins[] = {13,9,8};

void setup()
{
     for(int i = 0; i<=2;i++)  // Einrichten der Pins
     {
          pinMode(pin[i], OUTPUT);
     }
}
void loop()
{
     // Code hier einfügen
}
```

Hinweis
Mit *pinMode* können Sie die E/A-Pins auf dem Arduino-Board definieren.

while-Schleife: Diese Schleife wird so lange ausgeführt, bis eine Bedingung erfüllt ist. Trifft die erste Bedingung nicht zu, wird die Schleife überhaupt nicht ausgeführt. Mit einer *while*-Schleife können Sie z. B. Code ausführen, bis ein bestimmter Sensorwert erreich ist. Das folgende Beispiel veranschaulicht dieses Prinzip:

```
int potPin = A1;
int motorPin = 9;
int potVal;

void setup()
{
     pinMode(motorPin,OUTPUT);
     pinMode(potPin,INPUT);

}
void loop()
{
     potVal = analogRead(potPin);
```

```
     while(potVal <= 100)  // Wird so lange ausgeführt, bis
                           // potVal größer ist als 100
     {
          digitalWrite(motorPin,1);
     }
}
```

Mit diesem Code werden zuerst der Potenziometer und die Motor-Pins initialisiert. Anschließend wird *potVal* deklariert – unsere Variable, die den Potenziometerwert enthält. Danach wird *motorPin* als Ausgang und *potPin* als Eingang festgelegt. Schließlich verwenden wir eine *while*-Schleife mit der Bedingung *potVal <= 100*. Solange diese Bedingung wahr ist, läuft der Motor.

do…while-Schleife: funktioniert wie eine *while*-Schleife, mit dem einzigen Unterschied, dass die bedingte Anweisung erst am Ende der Schleife geprüft wird. D. h. die Schleife wird mindestens einmal ausgeführt. Beispiel:

```
do
{
     i++;  // i inkrementieren
}while(i <= 100);
```

2.3 Digitale Kommunikation

Im gesamten Buch werden wir verschiedene Arten von Ein- und Ausgaben über digitale Pins vornehmen. Ein grundlegendes Verständnis für diese Art von Kommunikation ist also unerlässlich. Wir verwenden insbesondere die Befehle *digitalWrite(pin,HIGH/LOW)* und *digitalRead(pin)*, um mit den digitalen Pins zu kommunizieren.

```
int button = 12;
int led = 13;
int buttonState;

void setup()
{
     pinMode(button,INPUT);
     pinMode(led,OUTPUT);

}
void loop()
{
     buttonState = digitalRead(button);  // Zuordnung von button
                                         // zu buttonState
     if(buttonState == 1)
     {
```

```
          digitalWrite(led,HIGH);  // legt für Pin 13 den
                                   // logischen Wert 1 fest
     }
     Else
     {
          digitalWrite(led,LOW);  // legt für Pin 13 den
                                  // logischen Wert 0 fest
     }
}
```

Das Programm verwendet die Funktionen *digitalWrite()* und *digitalRead()*, um den Wert des button-Pins abzurufen und einen neuen Wert festzulegen (in diesem Fall *HIGH* oder *LOW*).

Hinweis
Verwenden Sie die digitalen PWM-Pins, um die Motorgeschwindigkeit und die LED-Helligkeit zu steuern.

2.4 Analoge Kommunikation

Sie können auch auf analogem Weg mit Sensoren und Motoren kommunizieren, d. h., Sie können ein Potenziometer zur Steuerung der Motorgeschwindigkeit über einen PWM(Pulsweitenmodulations)-Pin auf dem Arduino-Board anschließen. Für die analoge Kommunikation stehen die Funktionen *analogRead(value)* und *analogWrite(pin,value)* zur Verfügung. Dabei gilt es zu beachten, dass ein Potenziometer einen Wert zwischen 0 und 1.024 zurückgibt. D. h., Sie müssen für *analogWrite* eine Skalierung für den Bereich 0 bis 255 vornehmen. Beispiel:

```
analogWrite(LED,ledValue/4); // 1.024/4 = 256
```

2.5 Serielle Kommunikation

Auch die serielle Kommunikation wird Sie durch das gesamte Buch begleiten. Mittels serieller Kommunikation lassen sich Computer, LCDs und viele andere Geräte ansteuern. Befehle für die serielle Kommunikation sind z. B. *Serial.begin(Baudrate)*, *Serial.println*(*Legen Sie einen beliebigen Wert für den seriellen Pin fest*), *Serial.read()*, *Serial.write*(*Binärdaten*), *Serial.available()* und *Serial.end()*. Mit diesen Befehlen können Sie Werte an beliebige serielle Peripheriegeräte ausgeben oder diese auslesen. Im Folgenden eine kurze Beschreibung der Befehle für die serielle Kommunikation.

Serial.begin(baud): Sie können diesen Befehl innerhalb der *setup()*-Struktur verwenden. Dazu müssen Sie die Baudrate des Geräts angeben, mit dem Sie kommunizieren möchten, z. B.:

```
void setup()
{
     Serial.begin(9600); // Baudrate = 9600 für Kommunikation
                         // mit einem Computer
}
```

Serial.println(): Mit diesem Befehl geben Sie Werte an den seriellen Port aus, z. B.:

```
void loop()
{
     Serial.println("Hallo Welt");   // Sendet "Hallo Welt" an
                                     // den seriellen Port
}
```

oder …

```
void loop()
{
     Serial.println(potVal);  // Sendet potVal an den seriellen
                              // Port
}
```

Serial.read(): Liest einen Wert am seriellen Port ein; Sie können mit diesem Befehl z. B. einen Wert von Ihrem Computer einlesen, den Sie anschließend auf einem LCD am Arduino-Board ausgeben.

```
void loop()
{
     char var = Serial.read();  // Lesen des eingehenden Bytes
                                // am seriellen Port
}
```

Serial.write(): Mit diesem Befehl geben Sie binäre Daten an einen seriellen Port aus, z. B.:

```
void loop()
{
while(Serial.available() > 0)
     {
          char var = Serial.read();  // Lesen des eingehenden
                                     // Bytes am seriellen Port
```

```
          Serial.write(var);  // Ausgeben binärer Daten an
                              // seriellen Port
     }
}
```

Hinweis
In diesem Buch verwenden wir in den meisten Fällen *Serial.println()*, um *int*- oder *string*-Werte in Serial Monitor auszugeben. Mit der Funktion *Serial.write()* können Sie binäre Daten an Serial Monitor oder an ein anderes Programm, das einen seriellen Port verwendet, senden.

Serial.available(): Diese Funktion prüft, ob eingehende Bytes am seriellen Port vorliegen, z. B.:

```
void loop()
{
     while(Serial.available() > 0)  // Stellt sicher, dass mind.
                                    // 1 Byte am seriellen
                                    // Port vorliegt
     {
          // Code hier einfügen
     }
}
```

Serial.end(): Mit dieser Funktion wird die serielle Kommunikation deaktiviert.

Nach dieser allgemeinen Einführung in die Befehle für die serielle Kommunikation ist es jetzt an der Zeit für ein etwas konkreteres Beispiel. Im folgenden Programm kommt ein Großteil der bisher vorgestellten Funktionen zum Einsatz.

```
int incomingByte;
const int ledPin = 13;
void setup() {

     Serial.begin(9600);     // Öffnet den seriellen Port und
                             // setzt die Datenübertragungsrate
                              // auf 9600 bit/s

     pinMode(ledPin, OUTPUT);

}

void loop()
{
  while(Serial.available() > 0)
  {
```

```
    incomingByte = Serial.read();      // Liest eingehendes Byte
    Serial.println(incomingByte, BYTE);// Gibt eingehendes Byte
                                       // an seriellen Port aus
    digitalWrite(ledPin, incomingByte);// Schreibt eingehendes
                                       // Byte in ledPin
  }
}
```

Dieses Programm bildet die Grundlagen der seriellen Kommunikation ab. *incomingByte* und *ledPin* werden initialisiert. Als Nächstes wird in der Setup-Struktur die Baudrate auf 9.600 gesetzt. Sobald wir uns innerhalb der Loop-Struktur befinden, prüft die *while*-Schleife, ob Daten am seriellen Port vorliegen. Ist dies der Fall, werden die Informationen am seriellen Port *incomingByte* zugeordnet. Schließlich werden die Daten am seriellen Port ausgegeben und in *ledPin* geschrieben (in diesem Fall 1 oder 0).

2.6 Arduino-Bibliotheken

Die Bibliotheken, die in den folgenden Kapiteln zum Einsatz kommen, sind in erster Linie die Bibliotheken NewSoftSerial, TinyGPS und LCD sowie einige andere, auf die wir zu einem späteren Zeitpunkt eingehen werden. Zuerst müssen die Bibliotheken jedoch heruntergeladen und im Ordner *Libraries* im Verzeichnis *Arduino-022* entpackt werden. Danach können Sie jede Bibliothek nutzen, die mit dem Arduino-Mikrocontroller kompatibel ist.

2.6.1 NewSoftSerial

Mit der Bibliothek NewSoftSerial können Sie Daten an mehrere serielle Softwareschnittstellen ausgeben. Einziger Nachteil ist, dass das Lesen und Schreiben nicht gleichzeitig möglich ist, sondern beide Vorgänge hintereinander ausgeführt werden müssen. Um die Bibliothek verwenden zu können, müssen Sie sie zuerst aus dem Internet herunterladen (*http://arduiniana.org/libraries/newsoftserial*).

Nachdem Sie die Bibliothek NewSoftSerial dem Arduino-Verzeichnis hinzugefügt haben, müssen Sie NewSoftSerial nur noch in dem Programm aufrufen, in dem Sie die Bibliothek verwenden möchten, z. B. so:

```
#include <NewSoftSerial.h>

NewSoftSerial gps(2,3);   // Erzeugt eine neue Instanz von
                          // NewSoftSerial mit dem
                          // Namen "gps".
```

```
                                // die Angaben in den Klammern
                                // stehen für (rx,tx), also
                                // Empfänger und Sender
```

Diese umfasst eine serielle gps-Schnittstelle an den Pins 2 und 3. In der Setup-Struktur müssen Sie die Baudrate der neuen seriellen Schnittstelle festlegen:

```
void setup()
{
     gps.begin(4800);

}
```

Danach können Sie die Funktionen der NewSoftSerial-Bibliothek über das zuvor erzeugte gps-Objekt aufrufen, anstatt die Serial-Bibliothek zu verwenden:

```
Serial.read();
```

wird zu

```
gps.read();
```

2.6.2 TinyGPS

Mit dieser Bibliothek wandeln Sie NMEA-Daten wie Längengrad, Breitengrad, Höhe über dem Meeresspiegel und Geschwindigkeit in ein benutzerfreundliches Format um. Weitere Informationen über die TinyGPS-Bibliothek finden Sie in Kapitel 5 (Integration eines GPS-Moduls). Für den Moment genügt es, wenn Sie die TinyGPS-Bibliothek unter der Adresse *http://arduiniana.org/libraries/tinygps* herunterladen.

2.6.3 ColorLCDShield-Bibliothek

Mit dieser LCD-Bibliothek können Sie Daten an einen LCD-Bildschirm ausgeben oder Daten von dort auslesen. Die LCD-Bibliothek wird ausführlich in Kapitel 4 (Arbeiten mit LCDs) behandelt. Laden Sie für den Moment einfach die LCD-Bibliothek unter *www.franzis.de/download/elo/65132-5_Arduino_in_der_Praxis.zip* herunter.

Stichwortverzeichnis

Harold Timmis

Apress®

Arduino™ in der Praxis

Mit Arduino™ in der Praxis setzen Sie Ihre Ideen in die Realität um. Lernen Sie, solide technische Grundsätze auf all Ihre Arduino™-Projekte anzuwenden – egal, ob Sie nur zum Spaß Geräte bauen oder sie verkaufen oder den Code veröffentlichen wollen. Harold Timmis zeigt Ihnen, wie Sie einen einwandfreien Entwurf für Ihr Arduino™-Projekt erstellen und gründliche Tests durchführen, bevor Sie sich auf einen speziellen Prototyp festlegen.

Arduino™ in der Praxis führt Sie in den gesamten Entwicklungsprozess ein, von den grundlegenden Voraussetzungen und vorläufigen Entwürfen bis hin zu Prototypen und Testverfahren. Ihr Wissen über Arduino™ und wesentliche Prinzipien der Entwicklung wird mit jedem Kapitel umfangreicher. Von der Arbeit mit einzelnen Komponenten wie LCDs, Potenziometern und GPS-Modulen schreiten Sie fort zur Integration mehrerer Module in größere Projekte und entwickeln letztendlich beispielsweise drahtlose Temperaturmesssysteme und sogar einen ganzen Roboter.

Ob Sie Arduino™ als Hobby betreiben, ein Entwickler sind, der sich mit Hardware-Hacking beschäftigt, oder ein Arduino-Hacker, der den nächsten Level erreichen will: Arduino™ in der Praxis ist die richtige Wahl, um Ihre Projekte mit Arduino™ und zusätzlicher Software zu perfektionieren.

Stärken Sie Ihr Wissen über Arduino™ und erfahren Sie, wie Sie:

- Ihre fest installierten und mobilen Arduino™-Projekte mit Bluetooth steuern
- grundlegende Entwicklungen mit Bluetooth Mate Silver auf Arduino™ anwenden
- mit Fehlermeldungen via GSM-Messaging arbeiten
- Ihre Projekte mit dem Xbox-Controller steuern
- Arduino™-Projekte mit LabVIEW debuggen

9 783645 651325

39,95 EUR [D]
ISBN 978-3-645-65132-5

Besuchen Sie unsere Website
www.franzis.de