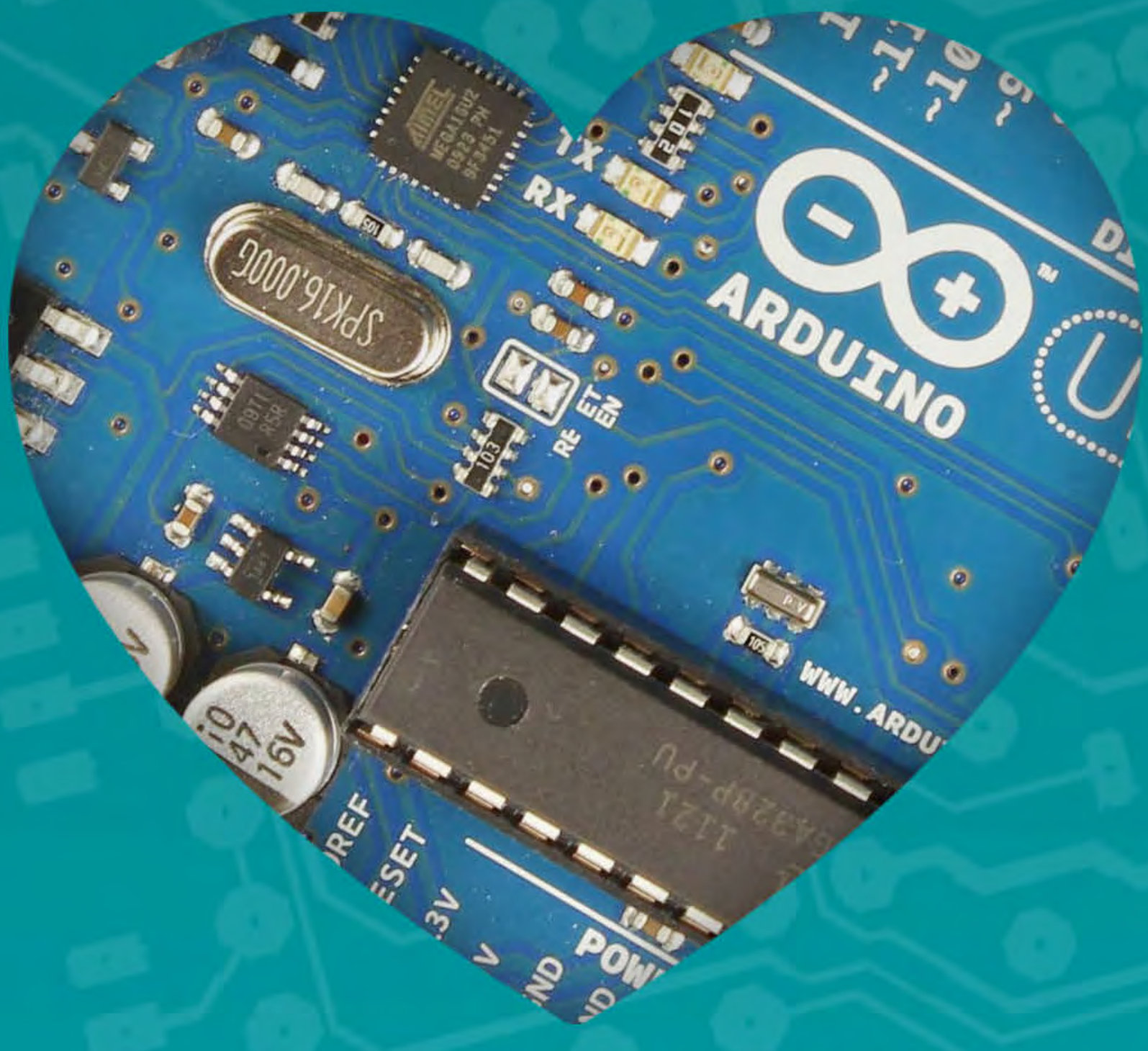
FRANZIS

I LOVE MY ARDUINO™



Spielerischer Umgang mit Arduino™
12 E-Books + Video-Tutorial

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.

© 2016 Franzis Verlag GmbH, 85540 Haar bei München

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.

Autoren: Christian Caroli, Philip Caroli, Manuel Di Cerbo, Fabian Kainka, Andreas Plötzeneder, Friedrich Plötzeneder, Andreas Rudolf, Mattias Schlenker, Ulli Sommer, Dr. Günter Spanner

ISBN 978-3-645-39075-0

INHALTSÜBERSICHT

Philip & Christian Caroli

Arduino™ Handbuch

Platinen, Shields, Elektronik und Programmieren: Roboter, SMS-Alarmanlage, Wetterstation, automatische Gießanlage und mehr als Treibstoff für eigene Projekte

Inhaltsverzeichnis

Teil IV
22 Projekte

5

Arduino™ im Praxiseinsatz

Gut gerüstet geht es jetzt an die praktische Umsetzung von 22 kleinen und großen Arduino-Projekten, die einen Querschnitt der vielen Einsatzmöglichkeiten des Arduino zeigen und dazu animieren, mit neuen eigenen Ideen daran anzuschließen.

5.1 Leonardo, der Kollegenschreck

Auf den ersten Blick kann man den Arduino Leonardo leicht mit einem Uno verwechseln, denn von den äußeren Ausmaßen und der Position der Steckleisten sind die beiden sehr gleich. Der große Vorteil des Leonardo gegenüber dem Arduino Uno ist jedoch, dass er auf sehr einfache Art und Weise mit dem PC oder einem Macintosh kommunizieren kann, denn durch eine besondere Programmierung simuliert der Mikrocontroller des Leonardo eine Tastatur oder eine Maus, die über USB an den entsprechenden Computer angeschlossen ist.

Damit ist der Leonardo problemlos in der Lage, den Mauszeiger des Computers zu verschieben oder Texte an den Computer zu senden, die dort genau so erscheinen, als ob sie über die Tastatur eingetragen worden wären. Der Leonardo dreht also als erster Arduino seiner Familie den Spieß um: Er bekommt nicht nur Befehle vom Computer, sondern kann genau solche auch an den Computer zurücksenden.

5.1.1 Motivation

Das Telefon klingelt. Eine von Fett und Chipskrümeln übersäte Hand greift zum Hörer. »Ja? … Ihr Mauszeiger dreht Kreise? Jetzt verarschen Sie mich nicht! … Oh Mann, muss man denn alles alleine machen? Ja, ich komme.« Mit einem abgrundtiefen Stöhnen landet der Hörer auf der Gabel. Ein massiger Körper wuchtet sich aus dem Drehstuhl, und das spärliche T-Shirt mit der Aufschrift »Ich Admin, du nix« wird in gekonnter Star-Trek-TNG-Manier glatt gezogen, was die bislang blank liegenden Speckröllchen gnädigerweise versteckt.

Bedeutungsschwere Schritte nähern sich einem klein gewachsenen Angestellten, der zwar nicht zitternd, so doch voller Sorge in seine Richtung schaut: »Ehrlich, ich habe nichts gemacht!« Der Administrator würdigt ihn keines Blickes, greift sich die Tastatur, plotzt auf den Sitz. Tatsächlich, der Mauszeiger beschreibt einen exakten Kreis und verhält sich dann wieder normal. Das ist doch bestimmt wieder ein Virus, den sich dieser Idiot beim Surfen auf illegalen Seiten eingefangen hat!

Der Task-Manager wird geöffnet, der eine oder andere verdächtige Prozess wird beendet. Der kleine Kollege zuckt nur einmal kurz auf, als die Arbeit des Nachmittags im digitalen Nirwana verschwindet – brav. Da! Schon wieder, ein runder Kreis! Aber wo ist denn jetzt der blöde Virus, er hat doch schon jedes in Frage kommende Programm gekillt! Und schon wieder! Erste Schweißperlen bilden sich – das gibt's doch nicht, bis jetzt hat er noch jeden Virus plattgemacht. Ah, ein Erfolg, keine Kreisbewegung mehr! Aber dafür läuft der Computer insgesamt nicht mehr. Also ein Neustart …

Zwei Stunden später entschwindet ein völlig entnervter Admin vorzeitig in den Feierabend, nachdem er dem Angestellten noch blumenreich mitgeteilt hat, dass er auf ein schwächeres Gerät wechseln muss und der Rechner auseinandergenommen wird – morgen.

Kurz vor Feierabend zieht ein dritter Kollege ein kleines USB-Kabel aus dem Rechner, das – unerreichbar für bückscheue Zeitgenossen – hinten am Rechner angebracht war. Sicherlich wird der kleine Kollege morgen melden, dass der Rechner jetzt wieder problemlos arbeitet. Und falls der Admin diesen Erfolg für sich verbuchen will, wird bestimmt die unschuldige Frage auftauchen, wie genau er das denn nur wieder geschafft hat und wie man das in Zukunft vermeiden kann.

5.1.2 Aufgabenstellung

Wir wollen ein kleines Programm schreiben, das einen Arduino Leonardo dazu veranlasst, den Mauszeiger einen Kreis ziehen zu lassen, nachdem eine Taste gedrückt wurde. Nach Beendigung der Kreisbewegung soll die Maus wieder freigegeben werden.

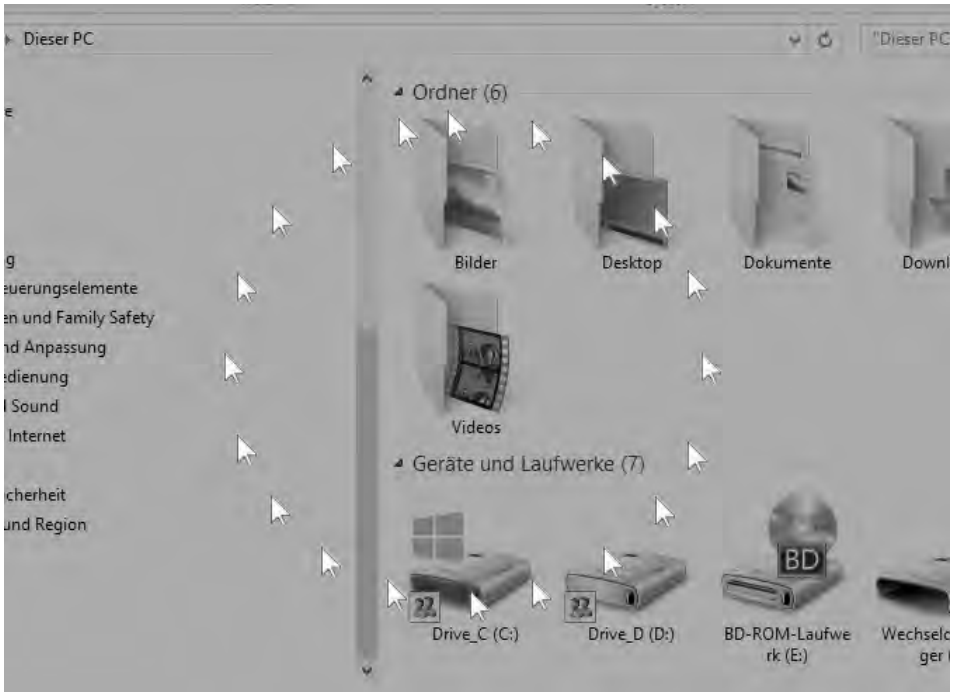


Bild 5.1: Der Arduino Leonardo bewegt den Mauszeiger im Kreis.

5.1.3 Hintergrundwissen

Es ist ja normal, dass ein Arduino über USB mit dem Computer kommunizieren kann. Wann immer ein Programm auf den Arduino geladen werden muss, läuft dies über USB, und auch der serielle Monitor, über den man Informationen vom Arduino erhalten kann, muss diesen Weg gehen.

Kommunikation eines normalen Arduino mit dem Computer

Das geschieht unter Zuhilfenahme der RS232-Schnittstelle, die für einen Arduino deutlich leichter zu handhaben ist als die USB-Schnittstelle. Der Grund hierfür liegt hauptsächlich darin, dass das USB-Datenaustauschprotokoll zwischen Gerät und Computer um ein Vielfaches komplexer ist als das der RS232-Schnittstelle und man gerade in der Anfangszeit des Arduino diese komplexe Thematik einem eigens dafür konstruierten Chip überlassen hat. Dieser Chip ist es, der das komplizierte USB-Protokoll ganz für sich allein abhandelt und dem Arduino das deutlich vereinfachte RS232-Protokoll zur Verfügung stellt.

In den früheren Arduinos wurde hierfür ein spezieller Chip verwendet, der aber keine hohen Geschwindigkeiten zuließ und noch ein paar andere Nachteile hatte. Als dann der Arduino immer älter und fortschrittlicher wurde, ist man umgestiegen auf einen

eigenen Mikrocontroller, der das USB-Protokoll in höherer Geschwindigkeit und mit mehr Kompatibilität abhandelte. Dennoch ist es heute immer noch so, dass der eigentliche Mikroprozessor des Arduino über die RS232-Schnittstelle mit dem Computer kommuniziert.

Kommunikation eines Leonardo mit dem Computer

Natürlich soll ein Arduino Leonardo genauso programmiert werden können wie ein Uno, auch er muss also über die RS232-Schnittstelle mit dem Computer kommunizieren können. Gleichzeitig aber soll er noch auf einem anderen Weg mit dem Computer in Verbindung treten, nämlich als Tastatur und Maus.

Mäuse und Tastaturen für Computer gibt es mittlerweile am Markt unübersehbar viele. Da alle die gleiche Funktion innehaben, nämlich Mausbewegungen bzw. Tastendrücke zu übermitteln, ist es nicht sinnvoll, für jede einzelne Tastatur oder Maus einen eigenen Treiber für Macintosh, Linux und Windows zu schreiben. Daher hat man sich schon vor geraumer Zeit auf einen Standard geeinigt, mit dem Mäuse und Tastaturen mit den Computern kommunizieren müssen. Als »Human Interface Devices« oder auch HID-Geräte werden sie heute von allen Computern mit modernen Betriebssystemen automatisch erkannt.

Auch der Leonardo, der Micro und der Esplora aus der Arduino-Familie gehen diesen Weg und geben sich beim Computer als HID-Gerät aus, das sowohl Maus- als auch Tastatursignale abgeben kann.

Die Kommunikation mit dem Computer übernimmt in diesem Fall kein externer Chip, sondern ein spezieller Mikrocontroller (ATmega32U4), der neben den normalen Aufgaben eines Arduino auch noch in der Lage ist, die USB-Kommunikation zu übernehmen.

Doch jetzt gibt es ein Problem: Der Arduino selbst mit seiner Möglichkeit, Programme zu empfangen und den seriellen Monitor anzusprechen, ist aus Sicht des Computers ein Gerät, die Simulation einer Tastatur oder Maus ist aber ein anderes, zusätzliches USB-Gerät. Es ist nicht möglich, beide Funktionen in einem USB-Anschluss zu vereinen.

Natürlich gäbe es die Möglichkeit, den Arduino mit zwei USB-Kabeln an den Computer anzuschließen. Das wäre aber umständlich und natürlich auch nicht besonders billig, weshalb die Erschaffer des Arduino Leonardo einen anderen Weg gewählt haben: Ist der Arduino an den Computer angeschlossen, um ein Programm zu empfangen oder die serielle Schnittstelle zu bedienen, befindet sich der Arduino in dem einen USB-Modus, soll er hingegen Tastatur- oder Maussignale an den Computer senden, befindet er sich in einem anderen.

Man ist also nicht gezwungen, das USB-Kabel ständig zu wechseln oder gar mit zwei Kabeln zu hantieren, es gibt aber auch ein paar Nachteile. Aufgrund der besonderen Architektur des Leonardo bricht bei einem Reset die serielle Verbindung zum Computer zusammen. Viel nervenaufreibender ist jedoch der Umstand, dass es der

Arduino-Entwicklungssoftware nicht immer gelingt, ein selbst geschriebenes Programm auf den Arduino hochzuladen.

Wenn nämlich der Arduino mit seinem bisherigen Programm die Tastatur- oder Mausfunktion des Computers anspricht, kann das zum einen dazu führen, dass die Bedienung des Programms erschwert wird – weil zum Beispiel der Mauszeiger gerade in eine Kreisbahn gedrängt wird –, zum anderen ist aber der Arduino gar nicht bereit dafür, neue Programme vom Computer entgegenzunehmen.

Auch wenn es hierfür einen kleinen Trick gibt, ist es doch angeraten, bei jedem Programm, das die Tastatur- oder Mausfunktion des Leonardo verwendet, eine Möglichkeit vorzusehen, die HID-Funktion seitens des Arduino ordentlich zu beenden und somit den Weg freizumachen für die normale Kommunikation mit dem Arduino. Das geschieht üblicherweise über die Befehle `Mouse.end()` und `Keyboard.end()`.

Pull-up-Widerstände

Wenn ein Taster gedrückt wird, besteht eine elektrische Verbindung zwischen seinen beiden Anschlüssen, wird er losgelassen, besteht auch keine Verbindung mehr. Wenn ein Taster mit einem Anschluss an die Masse gelegt wird und mit dem anderen an einen digitalen Pin des Arduino, wird bei gedrücktem Taster mit dem Befehl `digitalRead(pin_taster)` immer ein `LOW` gemessen. Was passiert aber, wenn der Schalter nicht gedrückt wird?

Der Pin des Arduino ist über ein Kabel mit dem Taster verbunden, aber es besteht keine Verbindung zur gegenüberliegenden Seite des Tasters. In diesem Fall werden wir bei 100 Aufrufen von `digitalRead(pin_taster)` ungefähr 50-mal ein `LOW` und 50-mal ein `HIGH` messen. Das liegt daran, dass wir mit dem Arduino-Pin und dem Kabel zum Taster eine kleine Antenne aufgebaut haben, die auf viele unterschiedliche Störquellen reagiert.

Für einen Taster ist das natürlich weniger gut. Beheben kann man das, indem man den Arduino-Pin mit einer Spannung verbindet, beispielsweise der Versorgungsspannung. Wenn man dabei noch einen hochohmigen Widerstand benutzt, wird bei gedrücktem Taster trotzdem ein `LOW` gemessen.

Auf solche Widerstände trifft man in der digitalen Elektronik häufiger, man nennt sie auch Pull-up-Widerstände. Freundlicherweise sind im Mikrocontroller des Arduino solche Pull-ups bereits intern eingebaut, wir müssen sie nur anschalten. Das geschieht, indem der entsprechende Pin durch den Befehl `digitalWrite(Pinnummer, HIGH)` initialisiert wird.

Da nur Pull-up-Widerstände eingebaut sind, die den Pin mit der Versorgungsspannung verbinden, und keine Pull-down-Widerstände für eine Verbindung mit der Masse, können Taster nicht zwischen Arduino-Pin und der Versorgungsspannung geschaltet werden, da sonst sowohl bei gedrücktem als auch bei nicht gedrücktem Schalter ein `HIGH` gemessen werden würde.

Benötigte Bauteile
- 1 Arduino Leonardo
- 1 Breadboard
- 1 Mikrotaster
- 2 Drahtbrücken

5.1.4 Schaltplan

Um den Taster an den Arduino anzuschließen, müssen Sie eine Seite mit der Masse verbinden – auf dem Arduino mit GND bezeichnet – und die andere Seite mit dem digitalen Pin 2 des Arduino. Bei einem üblichen Taster mit vier Anschlusspins sind jeweils zwei Pins im Bauteil intern miteinander verbunden, weshalb er nicht beliebig angeschlossen werden kann. Sie müssen zwei auf gegenüberliegenden Seiten herausstehende Pins zum Anschluss benutzen. Sollte Ihre Schaltung nicht funktionieren, versuchen Sie einmal, den Taster um 90 Grad zu drehen.

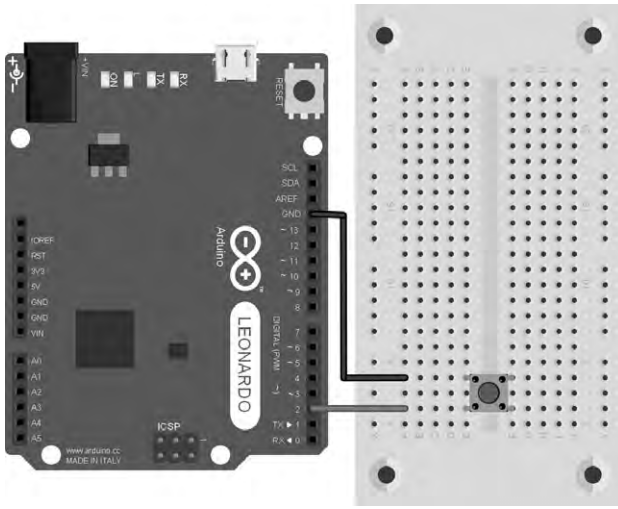


Bild 5.2: Der Breadboard-Aufbau der Schaltung.

Download Source Code
`Mauswandern.ino` und `MauswandernEsplora.ino`

- *buch.cd mit dem Code **60316-4***

5.1.5 Source Code

```
001 void setup() {
002   pinMode(2, INPUT);            //Pin 2 auf Ausgang setzen
003   digitalWrite(2, HIGH);        //Pull-up-Widerstand einschalten
```

```
004   Mouse.begin();                //Mauskontrolle übernehmen
005 }
006
007 bool StartKreis = false;
008 float GradZahl = 0;             //Aktuelle Position (Grad)
009 float SchrittWeite = 0.1;       //in Grad
010 double KreisDurchmesser = 100; //300 Pixel Durchmesser
011 double XPos = 0;                //Aktuelle absolute Position (in Pixel)
012 double YPos = 0;
013 double AlteXPos = 0;            //Vorherige absolute Position (in Pixel)
014 double AlteYPos = 0;
015
016 void loop() {
017   if (digitalRead(2) == LOW)    //Bei Knopfdruck Vorgang starten
018   {
019     StartKreis = true;
020   }
021
022   if (StartKreis) {             //Wenn Vorgang gestartet ist
023     //Berechnung der neuen Position
024     GradZahl = GradZahl + SchrittWeite;
025     XPos = round(sin(GradZahl) * KreisDurchmesser);
026     YPos = round(cos(GradZahl) * KreisDurchmesser);
027     // Maus an neue Position setzen
028     Mouse.move(XPos - AlteXPos, YPos - AlteYPos, 0);
029     AlteXPos = XPos;            //Aktuelle Position merken
030     AlteYPos = YPos;
031   }
032
033   if (GradZahl > 2 * PI) {      //Nach einer Umdrehung anhalten
034     GradZahl = 0;               //Position auf 0
035     StartKreis = false;         //Vorgang beenden
036     Mouse.end();                //Mauskontrolle wieder abgeben
037   }
038
039   delay(20);                    //Einige Millisekunden warten
040 }
```

In der `setup()`-Routine des Programms wird der Modus des zweiten Pins des Arduino auf Eingang geschaltet, und durch den `digitalWrite`-Befehl wird der im Prozessor vorhandene Pull-up-Widerstand aktiviert. Außerdem wird die Leonardo-spezifische Klasse `Mouse` durch den `begin()`-Befehl initialisiert, und damit wird auch die Kommunikation zum Computer als HID-Gerät aufgebaut.

Anschließend werden etliche Variablen definiert, die wir im weiteren Verlauf des Programms benutzen und über die man auch einige Einstellungen vornehmen kann. Interessant ist hierbei vor allem die Variable `KreisDurchmesser`, über die die Größe des Kreises bestimmt wird, die der Mauszeiger abfahren soll.

In der `loop`-Anweisung wird dann zuerst einmal abgefragt, ob Pin 2 auf `LOW` steht, was gleichbedeutend mit einem Tastendruck ist. Wurde die Taste gedrückt, wird die Variable `StartKreis` auf den Wert `true` gesetzt.

Eben diese Variable wird anschließend auf den Wert `true` geprüft, um sicherzugehen, dass auch immer ein Kreis abgeschlossen wird, egal wann die Taste wieder losgelassen wird.

Ist dies also der Fall, beginnt die Berechnung der aktuellen Position des Mauszeigers, die vor allem von der Variablen `GradZahl` abhängt, die den derzeitigen Winkel im Kreis angibt und die bei jedem Durchlauf des `loop`-Bereichs um eine `SchrittWeite` erhöht wird.

Bei der Berechnung selbst wird eine virtuelle Position des Mauszeigers berechnet. Virtuell deshalb, weil wir die tatsächliche Position des Mauszeigers nicht wissen, denn der Computer nimmt zwar Mauspositionierungsdaten vom Leonardo an, gibt aber keinerlei Informationen über die aktuelle Position zurück. Aus der virtuellen Position berechnen wir eine relative, denn nur das akzeptiert der Computer als Mauseingabe: Es ist also nicht möglich, dem Computer zu befehlen, an die Position x = 100 und y = 100 zu gehen, vielmehr akzeptiert er lediglich das Kommando, die Maus vom aktuellen Standpunkt aus um beispielsweise fünf Pixel nach oben und sieben nach rechts zu bewegen.

Die relative Position errechnen wir aus der virtuellen ganz einfach, indem wir die im vorherigen `loop`-Durchgang (`AlteXPos` bzw. `AlteYPos`) gespeicherte Position abziehen.

Nun benötigen wir noch eine Möglichkeit, die Kontrolle der Maus wieder abzugeben und damit die Kommunikation mit dem Arduino zum Upload eines neuen Programms wieder freizugeben. In unserem Fall wird einfach geprüft, ob die Variable `GradZahl` den Wert `2 * PI` angenommen hat, also die Kreisbewegung vollendet ist.

Im letzten Befehl des Programms lassen wir den Leonardo noch 20 ms warten, sodass der Mauszeiger insgesamt ca. 50 Mal in der Sekunde neu gesetzt wird.

5.1.6 Tipps und Tricks

Man kann dieses Projekt übrigens auch auf einem Arduino Esplora zum Laufen bringen. Zwar hat dieser keine Möglichkeit, einen Schalter anzubringen – dafür bringt er aber selbst mehr als genug davon mit.

Verwendung eines Arduino Esplora

Wenn Sie über einen Esplora verfügen, brauchen Sie also keinen externen Schalter anzuschließen und müssen lediglich den Source Code etwas abwandeln. Vor den gesamten Code schreiben Sie dazu die Zeile `#include <Esplora.h>`, die die Entwicklungsumgebung dazu veranlasst, die Esplora-Bibliothek zu verwenden. Anschließend müssen Sie nur noch die erste Zeile des `loop`-Bereichs

```
017 if (digitalRead(2) == LOW)    //Bei Knopfdruck Vorgang starten
```

durch diese hier ersetzen:

Philip & Christian Caroli

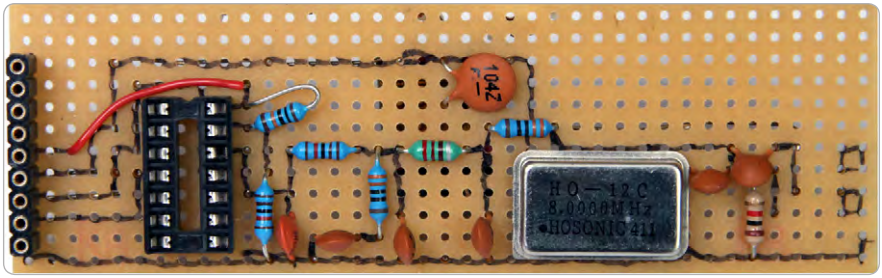
Schnelleinstieg Arduino™

Alles über Boards, Shields,
Installation, Programmierung
und Elektronik für die Praxis

INHALT

4 LÖTEN WIE DIE PROFIS

Wenn man ein gelungenes Selbstbauprojekt über einen längeren Zeitraum verwenden möchte, sind Breadboards nicht mehr die optimale Wahl. Die Bauteile können sich viel zu leicht lösen, mechanischen Belastungen schon kleinster Art halten die Kontakte kaum stand, und außerdem sind die Breadboards oft viel zu groß und unhandlich. Eine Lösung des Problems kann die Verwendung von sogenannten Lochrasterplatinen sein, auf denen man die Bauteile mit einem Lötkolben anbringt. Eine so gestaltete Platine ist oftmals viel kleiner, stabiler und auch langlebiger. Allerdings erfordert sie einen größeren Aufwand bei der Herstellung und sollte gut geplant sein, damit die Freude am Bastelprojekt nicht in Frust umschlägt.



Lochrasterplatinen sind universell verwendbar und langlebiger als ein Breadboard-Aufbau.

4.1 Werkzeug, das man zum Löten braucht

Wenn Sie mit einem Lötkolben arbeiten möchten, benötigen Sie natürlich ein paar Werkzeuge, die Ihnen die Arbeit ermöglichen und erleichtern.

4.1.1 Lötkolben und Elektroniklötzinn

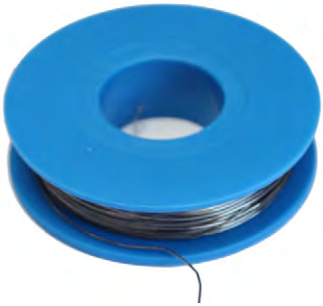
An allererster Stelle benötigen Sie einen **Lötkolben**. Sie sollten beim Kauf darauf achten, dass er für Elektronikarbeiten ausgelegt ist – 100-Watt-Lötpistolen oder Lötkolben für Tiffany-Glaskunst eignen sich nicht, es sei denn, Sie möchten Ihren Frust an der Platine abarbeiten. Einfache Elektroniklötkolben sollten eine möglichst feine und auswechselbare Lötspitze haben und ca. 30 W benötigen. Derartige Lötkolben sind recht günstig zu haben, einfache Modelle bekommen Sie bereits ab 15 Euro.

Einfache Lötkolben reichen oft für die ersten Lötarbeiten aus.

Wenn Sie aber planen, etwas tiefer in die Elektronik einzusteigen, empfiehlt sich der Kauf einer **Lötstation**, für die Sie ca. 60 bis 100 Euro ausgeben sollten. Diese Stationen bieten neben dem eigentlichen Lötkolben auch eine Temperatursteuerung, mit der Sie die Temperatur des Kolbens auf Werte zwischen 200 und 400 °C einstellen können.

Bei einer Lötstation kann man die Temperatur genau einstellen.

Neben dem Lötkolben benötigen Sie Elektroniklötzinn. Das **Lötzinn** ist eine spezielle Legierung aus verschiedenen Metallen, die durch den Lötkolben erhitzt wird und im flüssigen Zustand eine Verbindung zwischen dem jeweiligen Bauteil und der kupferbeschichteten Platine schafft. Lötzinn wird üblicherweise als Draht ausgeliefert, der im Kern ein Flussmittel beinhaltet, das die Aufgabe hat, das Bauteil und die Lötstelle auf der Platine besser zu benetzen, indem Fette und Oxidflächen auf den Metallen weitestgehend entfernt werden.



Beim Lötzinn sollte man nicht sparen, sondern eines in guter Qualität kaufen.

LOTE OHNE BLEI KONTRA BLEIHALTIGE LOTE

Da Blei die unangenehme Eigenschaft hat, sich im Körper anzulagern, und dort zu schweren Folgeschäden führen kann, steht das Metall zu Recht auf der Liste der gefährlichen Substanzen (RoHs). Daher darf die Elektronikindustrie im Allgemeinen nur Lote ohne Blei verwenden und hat andere Legierungen entwickelt, die ähnliche Eigenschaften haben. Wenn man aber lediglich einen einfachen Lötkolben besitzt und zudem wenig Löterfahrung hat, sind bleihaltige Lote für den Anfänger besser geeignet, denn sie schmelzen bei deutlich niedrigeren Temperaturen und lassen sich viel leichter verarbeiten als Lötzinn ohne Blei. Natürlich empfiehlt sich ein intensives Reinigen der Hände nach getaner Arbeit, und auch Handfood wie Gummibärchen und Nüsschen sollte man während des Arbeitens nicht zu sich nehmen.

GEEIGNETE LÖTZINNSTÄRKEN FÜR ANFÄNGER

Lötzinn gibt es in unterschiedlichen Stärken, als Anfänger sollten Sie einen Draht wählen, der zwischen 0,5 und 1mm liegt.

4.1.2 Lötschwamm zum Säubern des Kolbens

Wenn Sie Ihr erstes Lötzinn am Lötkolben schmelzen, werden Sie schon bald auf ein weiteres wichtiges Utensil zurückgreifen wollen, den **Lötschwamm**. Dieser Schwamm wird mit Wasser getränkt, sodass man in der Lage ist, den heißen Lötkolben samt dem daran haftenden Lötzinn auf diesem Schwamm

abzustreifen, bis die Lötspitze wieder sauber ist. Jedem Elektroniklötkolben sollte ein solcher Schwamm beiliegen, ansonsten bekommen Sie ihn für 1 bis 2 Euro im einschlägigen Elektronikfachhandel.

An Lötschwämmen kann man überschüssiges Lötzinn abstreifen.

4.1.3 Eine Spitzzange gegen Brandblasen

Wer mit einem heißen Lötkolben hantiert, verbrennt sich leicht die Finger. Um das zu vermeiden, sollten Sie sich eine kleine, nach vorne hin spitz zulaufende **Zange** anschaffen, mit der man Bauteile sicher festhalten kann.

Eine kleine Zange kann Ihre Finger vor Brandblasen schützen.

4.1.4 Ein Seitenschneider und eine Löthilfe

Sehr häufig müssen bei Lötarbeiten Drähte abgeschnitten werden. Hierzu ist ein **Seitenschneider** bestens geeignet, der die Kraft der Hand genau wie bei einer Zange verstärkt, aber zwei Schneideflächen bietet. Diese pressen sich genau aufeinander, wodurch der Draht abgezwickt wird. Da es bei Lötarbeiten von Vorteil ist, wenn der Draht ganz dicht an der Platine abgetrennt wird, empfiehlt sich ein Präzisionsseitenschneider, dessen Schneiden an nur einer Seite geschliffen sind.

Mit einem Seitenschneider trennt man Drähte durch.

Wer schon einmal gelötet hat, wird schnell feststellen, dass die Natur ihn leider nur mit zwei Armen ausgestattet hat. Die dritte Hand fehlt häufig, und auch eine herbeigerufene Person hat nicht unendlich viel Geduld und ist nicht immer bereit, ihre Finger in die Nähe eines 300 °C heißen Lötkolbens zu halten. Eine gute, wenn auch nicht perfekte Lösung ist die **helfende Hand**, die in den meisten Fällen zwei kleine Zangen sowie eine Lupe auf einem mehr oder weniger festen Standbein bietet. So kann man die zu lötenden Bauteile einspannen und mit den eigenen zwei Händen normal arbeiten.

Die Löthilfe ist die dritte Hand des Elektronikers.

4.1.5 Stahlwolle, um Korrosion zu entfernen

Ein ganz einfaches Utensil ist handelsübliche **Stahlwolle**. Mit ihr kann man sehr leicht eventuelle Korrosionen auf einer Lochrasterplatine entfernen, sodass das blanke Kupfer frei liegt und sich leicht löten lässt. Achten Sie aber beim Kauf darauf, dass Sie reine Stahlwolle kaufen, Wollen mit Seifenzusätzen eignen sich nicht für unsere Zwecke.

Mit Haushaltsstahlwolle kann man Platinen leicht von Kupferoxid befreien.

4.1.6 Kabel, Kupferlitze und Silberdraht

Natürlich möchten Sie auch in der Lage sein, Bauteile miteinander zu verbinden. Dafür benötigen Sie ein **Kabel**, das den elektrischen Strom leiten kann und an das Sie Bauteile anlöten können. Während früher solche Kabel aus einem einzelnen Kupferdraht bestanden, der entsprechend steif war und nicht selten bei mechanischer Belastung brach, wird heute meistens sogenannte **Kupferlitze** verwendet. Diese Kabel haben in ihrem Inneren nicht nur einen Kupferdraht, sondern viele kleine. Bricht einmal aufgrund mechanischer Belastung ein solches kleines Drähtchen, fällt das kaum ins Gewicht, solange es noch weitere Drähte gibt, die den elektrischen Strom leiten können.

Kabel sind heutzutage aus vielen Kupferdrähtchen zusammengesetzt und dadurch flexibler und haltbarer.

Üblicherweise ist die Kupferlitze mit einem Plastiküberzug isoliert, sodass sich die Kabel auch berühren können, ohne dass dies zu einem Kurzschluss führt. Vor allem aber haben diese Plastikisolierungen den Vorteil, dass sie in verschiedenen Farben erhältlich sind, sodass Sie beispielsweise die Versorgungsspannung immer mit einem roten Kabel versehen können und gleich wissen, welches Kabel welche Funktion hat.

Mit Silberdraht kann man Lötpunkte auf Lochrasterplatinen gut verbinden.

Wenn Sie mit Lochrasterplatinen arbeiten, kann es auch sinnvoll sein, sich ein paar Meter **Silberdraht** zu besorgen. Dieser Draht besteht aus einem einzelnen Strang und besitzt keine Isolierung. Vor allem wenn Sie mit Lochrasterplatinen arbeiten, kann Ihnen dieser Draht viel Arbeit ersparen.

4.1.7 Schrumpfschlauch gegen blanke Kabel

Schrumpfschlauch gibt es in verschiedenen Durchmessern und Längen. Er wird über das Kabel gestreift, das verlötet werden soll, und nach dem Verbinden über die offen liegenden blanken Stellen des Kabels geschoben. Anschließend kann man mit einem Heißluftföhn oder einem Feuerzeug den Schlauch erwärmen, der sich daraufhin zusammenzieht und so fest mit dem Kabel verbunden wird. So wird die Lötstelle professionell isoliert.

Die beiden Autoren dieses Buchs sind begeisterte und bekennende Schrumpfschlauch-Enthusiasten! Schrumpfschlauch wird bei uns überall und wo immer möglich eingesetzt. Wir haben Schrumpfschlauch in verschiedenen Farben und Stärken und isolieren damit nicht nur offene Kabelverbindungen, sondern auch die Lötfahnen an Schaltern oder anderen Elementen.

Schrumpfschlauch gibt es in verschiedenen Größen – meistens benötigt man aber nur die kleinste.

Unser 3-D-Drucker Heidi, dessen Elektronik wir schon mehrfach überarbeitet haben, ist sicherlich einer der am besten »geschrumpfschlauchten« Geräte überhaupt. Von Freunden und Bekannten werden wir, manchmal zu Recht, aufgrund unserer obsessiven Verwendung des Materials gehänselt – einige unserer Schaltungen sind fast komplett schwarz. Ungewollte Kurzschlüsse sind dafür bei uns noch nie aufgetreten – und wenn, würden wir das nie zugeben.

VORSICHTSMASSNAHMEN BEIM LÖTEN

Bevor Sie mit dem Löten beginnen, sollten Sie einige Vorsichtsmaßnahmen ergreifen:

- Vorsicht, heiß: Es mag ja dumm klingen, aber stellen Sie sich vor, Sie können sich mit einem Lötkolben tatsächlich die Finger verbrennen! Wer schon einmal eine Brandwunde hatte, weiß genau, wie lästig, nervend und vor allem lang anhaltend so ein Schmerz sein kann. Entgegen vereinzelten Meinungen schadet es auch dem Ansehen des größten Machos nicht, wenn er geeignete Gegenmaßnahmen ergreift. Arbeiten Sie ruhig und konzentriert, legen Sie Ihren Lötkolben immer wieder in die Halterung und ziehen Sie sich keine kurzen Hosen an – denn wohin fällt üblicherweise ein fallen gelassener Lötkolben?
- Schalten Sie Ihren Lötkolben in längeren Pausen immer wieder ab. Es wäre nicht das erste Mal, dass ein eingeschalteter Lötkolben vergessen wird und sich eine lieb gewonnene Person aus dem Haushalt daran verbrennt oder ein besonders leicht brennbarer Stoff auf der Werkzeugbank landet. Weitere Beispiele aus diesem Bereich können Sie bei Ihrer freundlichen Feuerwehr erfragen, der Zufall ist sehr kreativ, wenn es um die Schadensmaximierung bei der Verwendung heißer Gegenstände im Haushalt geht. ▸

VORSICHTSMASSNAHMEN BEIM LÖTEN *(FORTS.)*

- Lüften Sie Ihre Räumlichkeiten und vermeiden Sie direkten Kontakt mit dem Rauchgas, das beim Löten entsteht. Zum einen sind die Ausdünstungen nicht besonders gesund, das Einatmen und auch der Kontakt mit den Augen können zu Reizungen oder gar zu Schlimmeren führen. Zum anderen verbreiten sie auch keinen Rosenduft, und Ihre Mitmenschen werden es Ihnen sicherlich danken, wenn Sie das eine oder andere Mal das Fenster öffnen. Noch besser sind Absaugeinrichtungen, die das Rauchgas durch einen Aktivkohlefilter ziehen, der die darin enthaltenen Giftstoffe neutralisiert.

4.2 Tipps für fachgerechtes Löten

Das Thema des fachgerechten Lötens kann ganze Bücher füllen. Im Endeffekt läuft es aber häufig darauf hinaus, dass Sie viel üben und Ihre eigenen Erfahrungen machen. Hier einige Tipps, die Ihnen vielleicht das eine oder andere Mal weiterhelfen:

- **Wärmebelastung**: Verschiedene elektronische Bauteile sind unterschiedlich empfindlich. Während Widerstände relativ viel Wärme vertragen können, ist dies bei Leuchtdioden, Transistoren oder ICs anders. Falls Sie aus irgendeinem Grund einmal Arbeiten vornehmen müssen, die länger als einige Sekunden dauern, empfiehlt es sich, Pausen zum Abkühlen der Bauteile einzuplanen. Hält man Geräte mit einer Zange fest, kann diese Teile der Wärme aufnehmen und das Bauteil selbst kühlen.
- **Kalte Lötstellen**: Wenn beim Verbinden von zwei Bauelementen die Lötstellen nicht ausreichend erwärmt wurden, können sich sogenannte kalte Lötstellen bilden. Diese können eine Zeit lang den Strom mehr oder weniger gut leiten, durch mechanische Belastung oder altersbedingte Korrosion kann sich die elektrische Verbindung jedoch jederzeit wieder lösen und Fehler verursachen. Kalte Lötstellen kann man optisch erkennen, denn sie sind vom Erscheinungsbild matter als gute Lötstellen. Die Reparatur ist denkbar einfach: Man muss den Lötkolben lediglich einige Sekunden lang an die fehlerhafte Lötstelle halten, bis sich das Lötzinn korrekt mit den Bauteilen verbunden hat.

- **Flussmittel**: Wenn Sie am Lötkolben bereits Lötzinn hängen haben, das von Ihrer vorherigen Lötstelle stammt, eignet sich dieses oftmals nicht so sehr dazu, eine neue Lötstelle zu verbinden. Der Grund sind die im Lötzinn vorhandenen Flussmittel, die bei Lötzinn, das bereits einige Sekunden alt ist, schon verdampft oder verbrannt sind. Wenn Sie aber Ihrem bisherigen Tropfen ein winziges Stück neues Lötzinn hinzufügen, wird das darin enthaltene Flussmittel auch auf das alte Lötzinn übergehen und dieses wieder benutzbar machen.
- **Lötwasser** und **Lötfett**: Es gibt im Fachhandel auch Lötwasser und Lötfett, die die Flusseigenschaften von Lötzinn verbessern können. Anders als das Flussmittel im Lötzinn bilden diese Stoffe aber oft auf Isolationsmaterial Rückstände, die elektrisch leitend sind und Wasser anziehen. Damit eignen sich diese Stoffe nicht für das Elektronik-Löten.
- **Bleifreies Löten**: Wenn man RoHS-konformes Lötzinn verwenden möchte, benötigt man in erster Linie eine Lötstation, bei der man die Temperatur einstellen kann. Man braucht mit 350 bis 370 °C eine höhere Temperatur als bei normalem Lötzinn (300 bis 330 °C). Da die Fließeigenschaften von bleifreiem Lot schlechter sind, muss ausreichend Flussmittel verwendet werden, das zudem oft noch aggressiver als normales Flussmittel ist. Es wird also empfohlen, für eine besonders gute Lüftung bzw. eine Absaugvorrichtung für die Lötdämpfe zu sorgen. An bleifreies Löten sollten Sie sich erst dann heranwagen, wenn Sie bereits einige Erfahrungen gesammelt haben.

Mattias Schlenker

Sensoren am Arduino™

Hören, Sehen, Fühlen, Riechen: Zeigen Sie dem Arduino in über 20 Projekten mit analogen und digitalen Sensoren die Welt

Inhaltsverzeichnis

5

Digitale Sensoren

Viele Sensoren gibt es als Digitalbauteile. Sie sind natürlich aufgrund der verwendeten ICs (*Integrated Circuits*) oder sogar manchmal benötigter Mikrocontroller in der Regel etwas teurer als analoge Gegenstücke, die nur einen veränderlichen Widerstand zur Verfügung stellen. Im Gegenzug sind sie meist fertig kalibriert und liefern zuverlässige Informationen darüber, ob sie im korrekten Messbereich betrieben werden.

Protokolle gibt es viele. Manche verwenden die standardisierten Bus-Protokolle SPI oder I^2C, andere bauen auf proprietäre Protokolle, die aber aufgrund der niedrigen Datenraten leicht nachzuvollziehen und zu implementieren sind. Wiederum andere setzen auf eine Mischform aus digital und analog, beispielsweise indem der zeitliche Abstand zweier kurzer HIGH-Pegel den Messwert angibt.

Wie bei anderen Sensoren gilt, dass ein Studium des Datenblatts vor dem Kauf eine gute Entscheidung ist. Unterstützt der Sensor I^2C oder SPI und das Datenblatt spezifiziert das Nachrichtenformat präzise, stehen die Chancen gut, dass bereits ein anderer Tüftler eine Bibliothek programmiert hat. Ist dies nicht der Fall, ist immerhin bei diesen beiden Bus-Systemen eine Eigenentwicklung in der Regel mit wenig Aufwand möglich:

Meist sind nur sehr kurze Nachrichten auf dem Bus erforderlich (oft nur 1 Byte), um dem Sensor einen Messwert zu entlocken (oft ebenfalls nur 1 oder 2 Byte). Verwendet der Sensor ein selbst gestricktes Protokoll, achten die Hersteller meist darauf, bei mehrbittigen Nachrichten sehr lange HIGH- und LOW-Pegel zu verwenden — in solchen Fällen ist es wichtiger, das Protokoll mit geringem Aufwand implementieren zu können, als hohe Übertragungsraten zu erreichen. Und schließlich gibt es noch

Sensoren, bei denen genau ein Bit genügt: Ein dauernder HIGH- oder LOW-Pegel ist am einfachsten auszuwerten.

5.1 Temperaturmessung mit DHT11 und DHT22

In vielen Fällen genügt es nicht, die Temperatur allein zu messen. Geht es um ein Raumklima, bei dem man sich wohlfühlt, oder wird im Produktionsbereich (beispielsweise bei der Holzbearbeitung) ein relativ konstantes Klima gefordert, muss auch die relative Luftfeuchte berücksichtigt werden.

Für diesen Zweck gibt es mit der DHT-Familie (*Digital Humidity and Temperature*) Sensoren, die beide Werte messen und die analoge Messung als digitales Signal ausgeben. Konkret sind die beiden Baumuster »DHT11« und »DHT22« erhältlich. DHT11 kostet je nach Quelle 2 bis 6 Euro, DHT22 ist mit 5 bis 12 Euro rund doppelt so teuer.

5.1.1 Unterschiede und bevorzugter Einsatzzweck

DHT11 deckt einen Temperaturbereich von 0 bis 50 °C ab und erreicht dabei nur eine Genauigkeit von +/-2 °C. Die Luftfeuchtemessung ist im Bereich von 20 bis 80 % relativer Feuchte auf fünf Prozentpunkte genau. DHT22 dagegen deckt den Temperaturbereich von –40 bis +125 °C mit 0,5 °C Genauigkeit ab und soll auch bei der Luftfeuchte mit zwei bis fünf Prozentpunkten Abweichung (in der vollen Spanne 0 bis 100 %) genauer sein.

Damit dürfte klar sein, welcher Sensor wo sinnvoll einzusetzen ist. DHT11 ist akkurat genug, um ein Gewächshaus zu überwachen, bei Frost Alarm zu schlagen und auch sonst ausreichend genaue Werte für Beschattung und Belüftung zu liefern, damit Tomaten und andere Nutzpflanzen (Hanf für die Seilerei etc.) gedeihen.

Für den Einsatz in Wohnräumen, in denen die Wohlfühltemperatur selten größere Schwankungen als 0,5 °C zulässt und die Luftfeuchte nahe bei 50 % liegen sollte (60 % im Badezimmer), ist DHT22 die wesentlich bessere Wahl.

Benötigte Bauteile:

- DHT11 oder DHT22
- Widerstand 22 kOhm

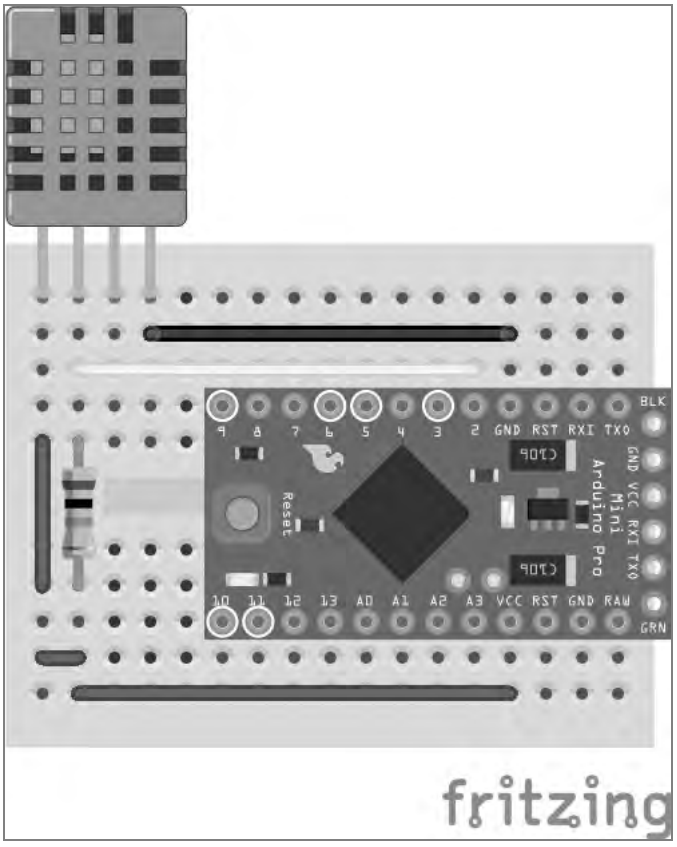


Bild 5.1: Der Data-PIN des DHT11/22 sollte einen Pull-up-Widerstand (10 kOhm oder 22 kOhm) erhalten, um zuverlässiges Auslesen zu ermöglichen.

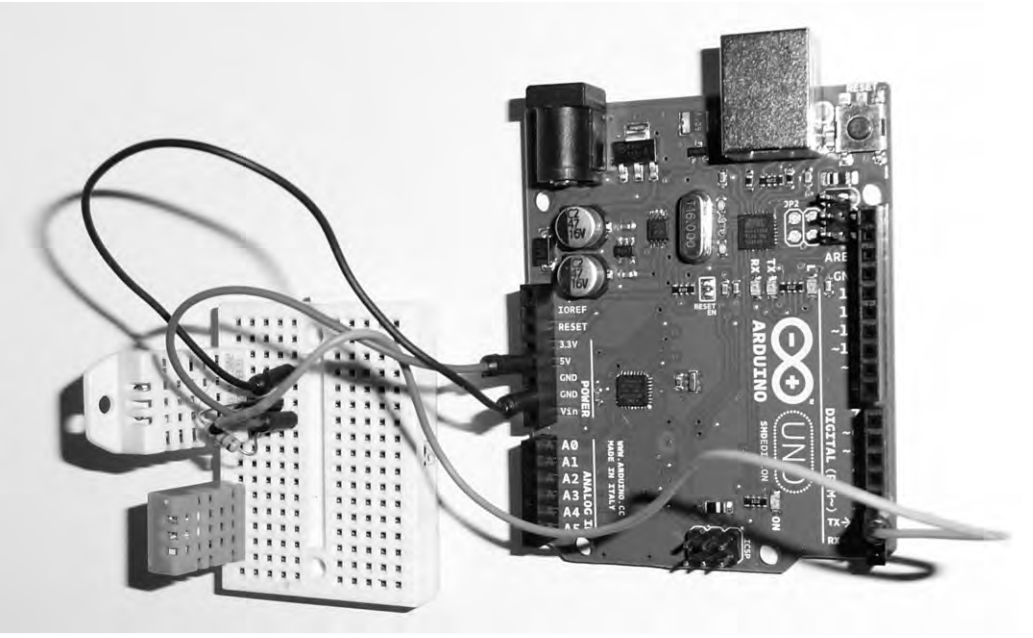


Bild 5.2: Für den Größenvergleich haben wir einen DHT11 (links, nicht angeschlossen) und einen DHT22 (rechts mit Schrauböse) auf ein Breadboard gepackt.

Die Anschlüsse von links nach rechts auf der Gitterseite sind: Versorgungsspannung (3,3 bis 5 Volt), Data, der dritte Pin wird nicht verbunden, Masse. Der Data-Pin wird über den Widerstand (als Pull-up) zur Versorgungsspannung verbunden.

Wir haben uns für die DHT-Bibliothek von Adafruit (auf *buch.cd* oder im GitHub-Repository) entschieden, die leider keinen gleichzeitigen Betrieb von DHT11 neben DHT22 an einem Arduino zulässt. In der Praxis geht das in Ordnung, während des Testens und Kalibrierens stört es. Die Bibliothek von Adafruit liefert unter *Datei/Beispiele/DHT/DHTtester* ein Beispiel mit. Aufs Wesentliche reduziert, sieht der Sketch wie folgt aus:

```
001 #include "DHT.h"
002 #define DHTPIN 2
003 //#define DHTTYPE DHT11   // DHT 11
004 #define DHTTYPE DHT22   // DHT 22  (AM2302)
005 //#define DHTTYPE DHT21   // DHT 21 (AM2301)
006 DHT dht(DHTPIN, DHTTYPE);
007
008 void setup() {
009   Serial.begin(9600);
010   Serial.println("DHTxx test!");
011   dht.begin();
```

```
012 }
013
014 void loop() {
015   delay(3000);
016   float h = dht.readHumidity();
017   float t = dht.readTemperature();
018   float f = dht.readTemperature(true);
019   // Check if any reads failed and exit early (to try again).
020   if (isnan(h) || isnan(t) || isnan(f)) {
021     Serial.println("Failed to read from DHT sensor!");
022     return;
023   }
024   float hi = dht.computeHeatIndex(f, h);
025   Serial.print("Humidity: ");
026   Serial.print(h);
027   Serial.print(" %\t");
028   Serial.print("Temperature: ");
029   Serial.print(t);
030   Serial.print(" *C ");
031   Serial.print(f);
032   Serial.print(" *F\t");
033   Serial.print("Heat index: ");
034   Serial.print(hi);
035   Serial.println(" *F");
036 }
```

Interessant sind die Funktionen `isnan()` und `computeHeatIndex()`. Erstere prüft auf fehlerhafte Rückgaben des Sensors, Letztere berechnet eine »gefühlte Temperatur«.

Beim Test mit dem DHT22 fiel übrigens auf, dass er am Anfang viel zu hohe Werte lieferte und erst nach einigen Stunden Betrieb die versprochenen 0,5 °C maximaler Abweichung erreichte. Offenbar benötigt der verwendete Thermistor einige Stunden, bis er seine endgültigen elektrischen Eigenschaften erreicht hat.

5.1.2 DS3231 Real Time Clock

Eigentlich handelt es sich bei der DS3231 um eine Echtzeituhr, »Real Time Clock«, mit Quarz und Temperaturkompensation. Die Preise für ein fertig verlötetes Modul mit sechspoliger Sockelleiste und Knopfzelle liegt zwischen 2 Euro bei günstigen Versandhändlern und 8 Euro. Mittlerweile sind auch Exemplare mit fünfpoliger weiblicher Steckerleiste im Umlauf, die zum Aufstecken auf den Raspberry Pi gedacht sind. Sie unterscheiden sich nur mechanisch, also in Form und Anordnung, von den Modulen mit männlicher Sockelleiste und können ebenso verwendet werden. Prüfen Sie in diesem Fall jedoch die Bezeichnung der Pins doppelt.

DS3231 verwendet den I²C-Bus, der auch dazu benutzt werden kann, zwei Arduinos miteinander oder einen Arduino mit einem Raspberry Pi kommunizieren zu lassen. Bei Betrieb von mehr als zwei Kommunikationsteilnehmern auf einem Bus ist strikt darauf zu achten, dass alle im gleichen Spannungsbereich arbeiten — im Zweifel verwenden Sie gemeinsame 3,3 Volt. Sind nur zwei Kommunikationsteilnehmer auf einem Bus vorhanden, ist das Spannungsniveau zweitrangig. Beim Kauf eines DS3231-Moduls sollten Sie darauf achten, dass es den gesamten Spannungsbereich von 3,0 bis 5,5 Volt abdeckt, das macht den Einsatz leichter.

Der I²C-Bus des Arduino liegt auf den Pins A4 (SDA) und A5 (SCL), daneben müssen noch Masse und Versorgungsspannung verbunden werden. Bei Verwendung eines Pro Mini oder einer davon abgeleiteten Platine ist darauf zu achten, dass A4 und A5 entweder auf nach innen versetzten Pins oder an der Stirnseite liegen.

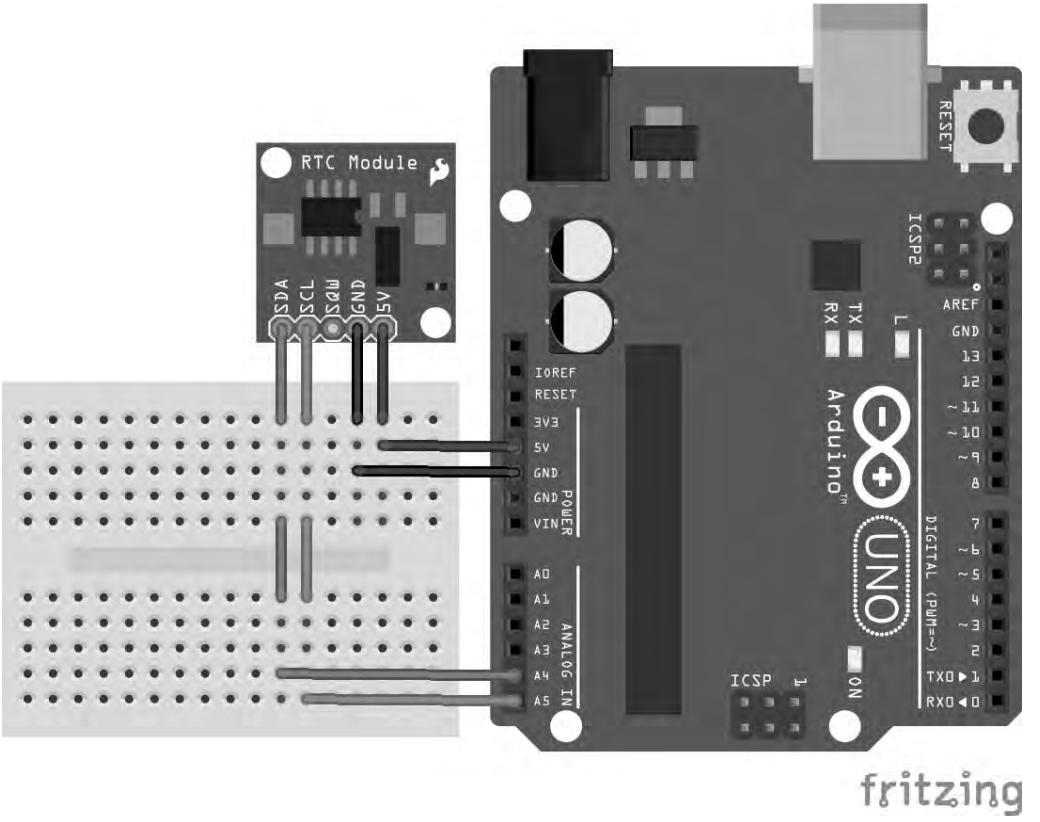


Bild 5.3: Es genügt, Masse und Versorgungsspannung sowie SDA (A4 am Arduino) und SCL (A5) anzuschließen.

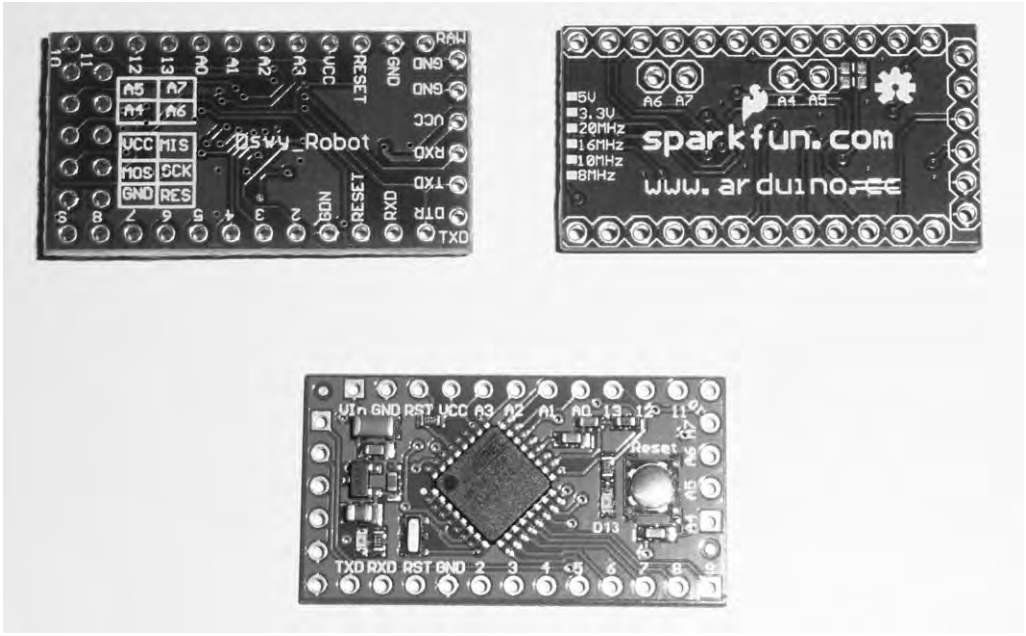


Bild 5.4: Bei Pro Mini und Derivaten hilft ein Blick auf die Platine, A4 und A5 zu finden — drei Platinen, drei unterschiedliche Layouts.

Zur Ansteuerung der DS3231 haben wir uns für die Bibliothek von Petre Rodan entschieden. Sie ist auf GitHub erhältlich (*https://github.com/rodan/ds3231*), zudem finden Sie in unserem GitHub-Repository und auf *www.buch.cd* einen Abzug. Es gibt auch Bibliotheken für die kleine Schwester, DS1307, mit Erweiterungen für DS3231, doch diese lassen oft kein Auslesen des Temperaturfühlers zu. Öffnen Sie nach dem Anschluss den Beispielsketch *Beispiele/DS3231/rtc_ds3231* und spielen Sie ihn auf den Arduino.

Der Sketch verlangt eine serielle Verbindung und gibt über diese im Fünfsekundentakt die Zeit aus. Über die serielle Verbindung ist auch eine Interaktion mit der Uhr möglich. Zum Setzen der Zeit verwenden Sie den Befehl

```
001 TssmmhhWDDMMYYYY
```

also beispielsweise

```
001 T301512W17052015
```

um das Datum auf den 17. Mai 2015, 12:15:30 Uhr, einzustellen. Mit dem Befehl `c` lesen Sie den integrierten Temperaturfühler aus. Erstellen Sie dann den folgenden Sketch, um alle 5 s Temperatur und Uhrzeit auslesen und anzeigen zu lassen:

```
001 #include <Wire.h>
002 #include "ds3231.h"
003
004 #define BUFF_MAX 128
005
006 uint8_t time[8];
007 unsigned int recv_size = 0;
008 unsigned long prev, interval = 5000;
009
010 void setup() {
011     Serial.begin(9600);
012     Wire.begin();
013     DS3231_init(DS3231_INTCN);
014     Serial.println("GET time");
015 }
016
017 void loop() {
018     char in;
019     char buff[BUFF_MAX];
020     unsigned long now = millis();
021     struct ts t;
022     if ((now – prev > interval) && (Serial.available() <= 0)) {
023         DS3231_get(&t);
024         snprintf(buff, BUFF_MAX, "%d.%02d.%02d %02d:%02d:%02d", t.year,
025              t.mon, t.mday, t.hour, t.min, t.sec);
026         Serial.print(buff);
027         Serial.print("\t");
028         Serial.print(DS3231_get_treg(), DEC);
029         Serial.println("°C");
030         prev = now;
031     }
032 }
```

In diesem Sketch wird zunächst der I²C-Bus initialisiert (`Wire.begin()`) und dann die Kommunikation mit der Uhr (`DS3231_init()`). Im Loop wird eine in `ds3231.h` definierte Struktur `ts` (Zeitstempel) als Zeiger übergeben, damit die Funktion `DS3231_get()` sie mit Werten füllt. Die Temperatur liefert die Funktion `DS3231_get_treg()` als Fließkommazahl zurück.

Bei unseren Experimenten haben wir konstante Abweichungen von 1 bis 2 °C ermittelt — der Hersteller gibt bis zu 3 °C Abweichung an. Soll die Temperatur tatsächlich ausgelesen und angezeigt werden, empfehlen wir, zunächst eine Weile einen DHT22 zur Kontrolle mitlaufen zu lassen und dann Korrekturwerte einzuführen. So kalibriert, dürfte es über einen weiten Bereich möglich sein, die Ungenauigkeit in den Bereich von ca. 0,5 °C zu drücken.

Die relativ präzise Messung der Temperatur und die exakte Zeitmessung macht die DS3231 besonders attraktiv für Sensoren, die exakte Zeitstempel speichern sollen, beispielsweise wenn über einen längeren Zeitraum Bodenfeuchte und Lichteinfall auf eine SD-Karte geschrieben werden sollen.

5.1.3 One-Wire-Temperatursensor DS18D20

Eine günstige Alternative zu DHT11 und DHT22 ist der One-Wire-Temperatursensor DS18D20, oft einfach als Dallas-Sensor bezeichnet. Je nach Quelle kostet er zwischen 1 und 3 Euro und liegt damit kaum über dem Preis analoger Sensoren (NTC oder PTC). Er deckt einen Temperaturbereich von –55 °C bis 125 °C ab — damit eignet er sich weder für den Wetterballon noch fürs Barbecue, wohl aber für den typischen Haus- und Gartengebrauch.

Im Bereich zwischen –10 °C und +85 °C wirbt der Hersteller mit einer Genauigkeit von 0,5 °C. Da jeder Sensor über eine eindeutige Seriennummer verfügt, über die er adressierbar ist, können mehrere DS18D20 an einem Bus betrieben werden, beispielsweise um mit lediglich einem Arduino-Pin Innen- und Außentemperatur oder die Temperatur an verschiedenen Stellen in einem Raum zu messen.

Ein Clou des DS18D20 ist der parasitäre Betrieb — hier dient eine Leitung gleichzeitig der Stromversorgung und als Signalleitung, es genügt folglich ein zweiadriges Kabel. Um den DS18D20 parasitär einzusetzen, verbinden Sie die beiden äußeren Pins miteinander und mit Masse. Der mittlere Pin wird mit einem Pull-up-Widerstand (3,3 oder 5 kOhm) an einen digitalen Pin des Arduino angeschlossen — im Beispiel verwenden wir Pin 2.

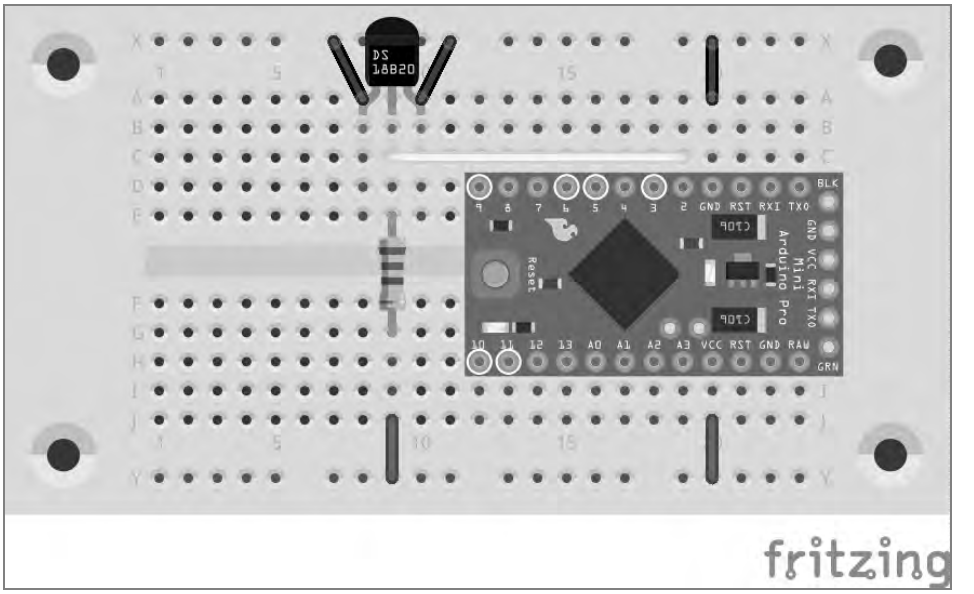


Bild 5.5: Eine Besonderheit ist die hier eingesetzte parasitäre Stromversorgung — beide äußeren Pins werden mit Masse verbunden.

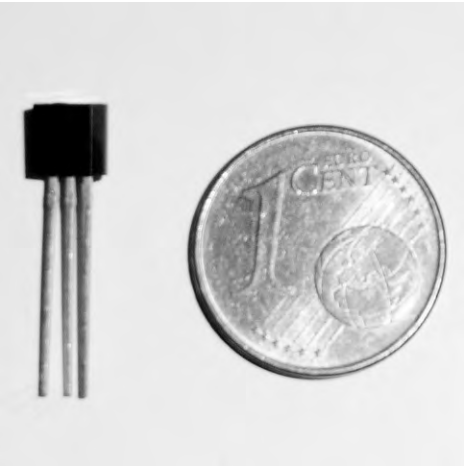


Bild 5.6: Die DS18x20-Sensoren sind sehr kompakt und bereits für gut 1 Euro erhältlich.

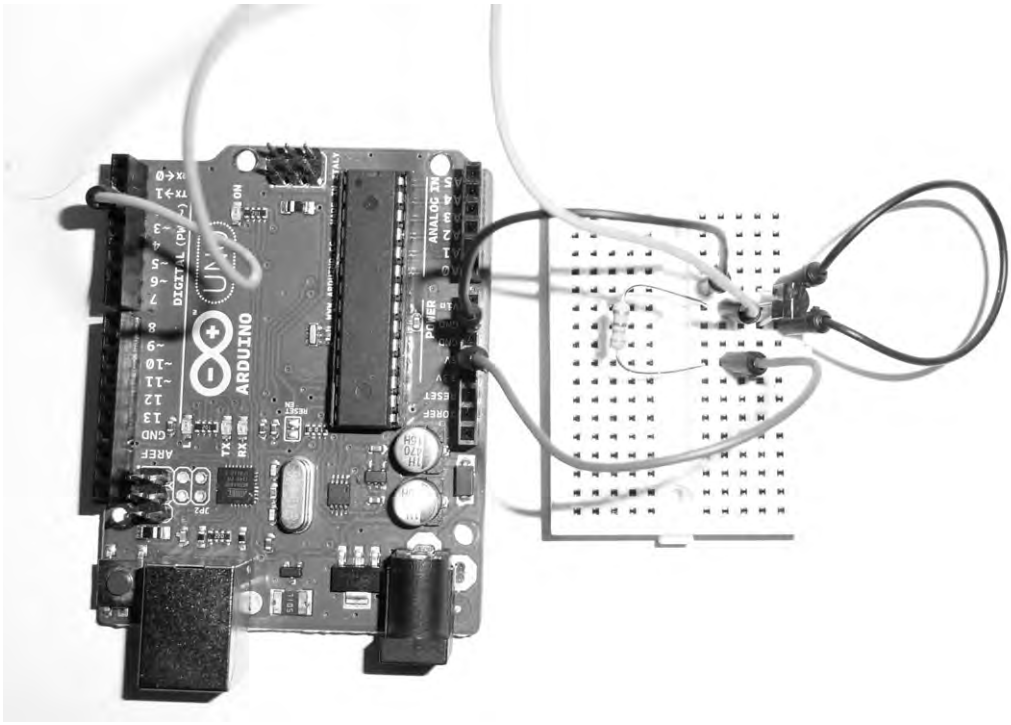


Bild 5.7: Bei parasitärer Energieversorgung kommt ein Pull-up-Widerstand mit 3,3 oder 5 kOhm zum Einsatz.

Als Bibliothek verwenden wir TCL (`Temperature Control Library`) von Miles Burton (*www.milesburton.com/?title=Dallas_Temperature_Control_Library*). Diese wiederum setzt Paul Stoffregens One-Wire-Bibliothek voraus (*www.pjrc.com/teensy/arduino_libraries/OneWire.zip*).

```
001 #include <OneWire.h>
002 #include <DallasTemperature.h>
003
004 #define ONE_WIRE_PIN 2
005
006 OneWire oneWire(ONE_WIRE_PIN);
007 DallasTemperature sensors(&oneWire);
008
009 void setup(void) {
010   Serial.begin(9600);
011   Serial.println("Dallas Temperature IC Control Library Demo");
012   sensors.begin();
013   sensors.setResolution(12); // 9 bis 12 Bit
014 }
015
```

```
016 void loop(void) {
017   Serial.print("Requesting temperatures...");
018   sensors.requestTemperatures();
019   Serial.println("DONE");
020   Serial.print("Temperature for Device 1 is: ");
021   Serial.println(sensors.getTempCByIndex(0));
022   delay(2000);
023 }
```

Interessant sind zwei Aspekte: Zunächst setzen wir gleich nach `sensors.begin()` die Auflösung der Messung auf 12 Bit. Dies führt zu deutlich präziseren Messungen als die voreingestellten 9 Bit, allerdings benötigt der Sensor dann länger zur Berechnung des Rückgabewerts. Das zweite Kuriosum ist `getTempCByIndex()`. Diese Funktion liefert die Celsius-Temperatur des ersten Dallas-Sensors. Wie eingangs erwähnt, können sich mehrere Sensoren einen Bus teilen. Bei zwei oder drei Sensoren ist es machbar, mit einem Eiswürfel den Index der einzelnen Sensoren zu ermitteln.

Bei mehr Sensoren an einem Bus werden Sie zunächst die Funktion `getAddress (index)` nutzen, um eine Adresse zu ermitteln, und schließlich `requestTemperatures-ByAddress()` verwenden, um auf einen bestimmten Sensor zuzugreifen.

Übrigens: Die letzten Bytes der Sensoradresse finden Sie in den Sensor eingelasert. Das macht die Zuordnung leichter.

5.2 Passive Infrarotsensoren

Passive Infrarotsensoren zur Bewegungserkennung werden gern und häufig zum Schalten von Lampen oder in Alarmanlagen eingesetzt. Die Sensoren sind ab 2 Euro pro Stück erhältlich und erkennen recht präzise Menschen, Haustiere und Kraftfahrzeuge. Bei PIR-Sensoren handelt es sich um echte digitale Sensoren, bei denen ein HIGH-Pegel Bewegung anzeigt und ein LOW-Pegel Ruhe. Ausgeliefert werden die Sensoren zumeist mit zwei Potenziometern.

Mit Sx stellen Sie die Empfindlichkeit ein, Tx bestimmt, wie lang ein HIGH-Pegel gehalten wird — typischerweise fünf Sekunden bis drei Minuten. In der Regel sind die Sensoren auf fünf bis zehn Sekunden eingestellt, die Leuchtdauer einer angeschlossenen (oder per Funksteckdose angesteuerten Lampe) können wir als glückliche Arduino-Nutzer in Software festlegen.

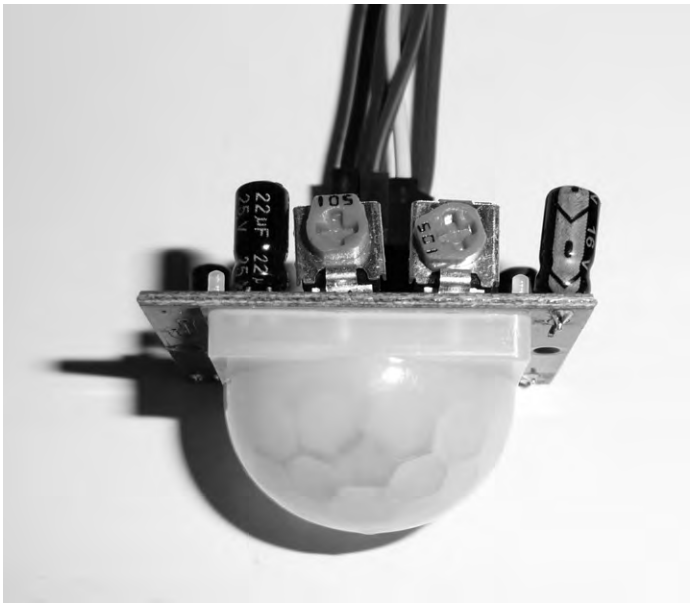


Bild 5.8: Passive Infrarotsensoren besitzen meist zwei Potis — interessant für uns ist Sx zur Einstellung der Sensitivität.

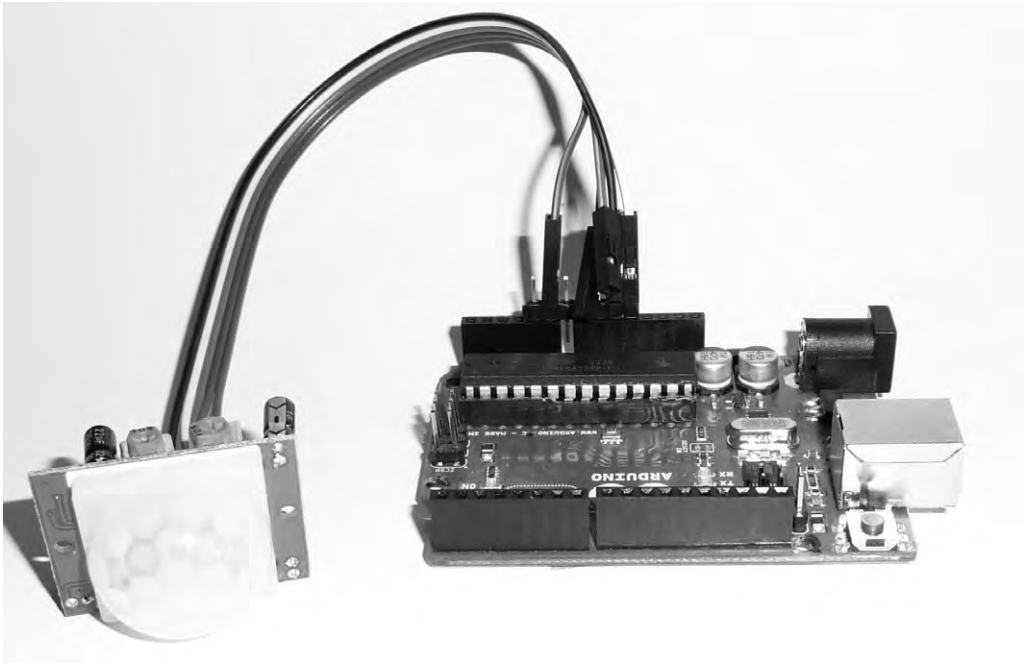


Bild 5.9: Der Anschluss an den Arduino erfolgt mit Masse, Versorgungsspannung und einem Daten-Pin — eine spezielle Bibliothek wird nicht benötigt.

Angeschlossen werden die drei Pins des PIR an Masse, 5 Volt und einen beliebigen Daten-Pin — aus Platzgründen haben wir für unser Beispiel Pin A0 gewählt. Für die Ansteuerung ist keine spezielle Bibliothek nötig, wir verwenden `digitalRead()`.

```
#define pirPin A0
#define calTime 30

void setup() {
  pinMode(pirPin, INPUT);
  Serial.begin(9600);
  digitalWrite(pirPin, LOW);
  Serial.print("Waiting for sensor to settle");
  for (int i=0; i<calTime; i++) {
    Serial.print(".");
    delay(1000);
  }
  Serial.println();
}

void loop() {
  if (digitalRead(pirPin) == HIGH) {
    Serial.println("Motion!");
  } else {
    Serial.println(".");
  }
}
```

Wie bereits erwähnt, schafft ein Arduino mehr Flexibilität als eine recht grobe Potenziometereinstellung. Soll beispielsweise erreicht werden, dass eine Lampe 30 s nachleuchtet, können Sie dies dadurch realisieren, dass Sie im Sketch bei nicht erkannter Bewegung vergleichen, wann die letzte Bewegung war. In Code umgesetzt, sieht das wie folgt aus — als Lampe dient hier die LED an Pin 13.

```
#define pirPin A0
#define calTime 30
#define ldelay 30000
#define ledPin 13

unsigned long lastMovement = 0;

void setup() {
  pinMode(pirPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  digitalWrite(pirPin, LOW);
```

Mattias Schlenker

HAUSAUTOMATION MIT ARDUINO™

Inhaltsverzeichnis

Einstieg in die Hausautomation

1.1 Begriffsklärung

»Haus- bzw. Heimautomation« kennzeichnet die automatisierte Steuerung von Heizung, Klimatisierung, Beleuchtung und Beschattung. In den letzten Jahren ist noch der Aspekt der Einlass- oder Zugangskontrolle hinzugekommen. Erreichen möchte die moderne Hausautomation Energieeinsparung, Komfortgewinn und eine Erhöhung der Sicherheit. Die Ziele sind von Anwendungsfall zu Anwendungsfall und von Nutzer zu Nutzer unterschiedlich gewichtet und häufig sogar konträr. Auch kulturelle und wirtschaftliche Aspekte spielen eine große Rolle: In Ländern mit hohen Energiepreisen, wie in Westeuropa üblich, genießt Energieersparnis einen hohen Stellenwert, während der typische kalifornische Nutzer eher den Komfortgewinn durch die perfekte Klimaanlagensteuerung zu schätzen wissen dürfte. In Ländern mit hoher Kriminalität wie Südafrika werden viele Anwender der Sicherheit ein größeres Augenmerk schenken.

Die Wege, diese Ziele zu erreichen, unterliegen einem zeitlichen Wandel, und einfache Lösungen wurden bereits lange vor Entstehung des Begriffs praktiziert. So sind Zentralheizungen mit Nachtabsenkung seit mindestens 40 Jahren üblich (auch wenn der Einsatz der Nachtabsenkung in modernen Gebäuden nur noch wenig Sinn ergibt). Und ich erinnere mich daran, dass mein Vater vor rund 30 Jahren das ganze Haus mit Heizkörperthermostaten ausstattete, die einen Heizwiderstand in der Nähe des Bimetalls enthielten: 1 V (ein Volt) Spannung sorgte für die Absenkung um ein Grad

Celsius. Zusammen mit einem kleinen Trafo und einer zentral aufgestellten Schaltuhr wurde so die Temperatur zwischen den Niveaus 17 °C und 21 °C reguliert. Dafür mussten allerdings zweiadrige Leitungen durchs ganze Haus gelegt werden.

Mit der Verfügbarkeit günstiger Mikrocontroller kamen vor rund 15 Jahren programmierbare Heizkörperthermostate auf den Markt. Die letzten zehn Jahre sahen das Aufkommen einfacher Funklösungen zur zentralen Steuerung von Heizung und Licht bzw. Steckdosen. Dann hielten lokale Netzwerke und das Internet Einzug. Heute geht der Weg hin zu selbst lernenden Systemen, einer Vernetzung per Bluetooth und WLAN und der Kontrolle über die »Cloud«.

Und der technische Fortschritt geht weiter: Wer heute neu baut, nutzt zur Heizung möglicherweise keine Warmwasserzentralheizung mehr, sondern eine Wärmepumpe, die im Sommer als Klimaanlage fungiert. Sensoren registrieren den Kohlendioxid- und Feuchtegehalt in der Luft und lüften selbsttätig – Gegenstromwärmetauscher sorgen in diesem Fall dafür, dass wenig Energie verloren geht. Derart moderne Häuser sind meist über Bussysteme vernetzt, über die auch andere Geräte wie Rollos, einzelne Steckdosen oder die Beleuchtung gesteuert werden. Momentan sind das Alter und der Bestand solcher per Bus vernetzter Häuser noch überschaubar – in zehn Jahren wird die Erweiterung und Anpassung in vielen Fällen eine echte Herausforderung werden, wenn wichtige zentrale Komponenten nicht mehr erhältlich sind oder die Dokumentation von Protokollen nach Pleiten oder Übernahmen nicht mehr auffindbar sind.

Mein Fokus liegt daher auf der Nachrüstung von Altbeständen. Auf Gedanken zum Neubau und die Bussysteme der teuren kommerziellen Lösungen für die Erstausrüstung gehe ich nur am Rande ein.

1.2 Kommerzielle Produkte

Derzeit buhlt eine Reihe kommerzieller Nachrüstlösungen mit unterschiedlichen Schwerpunkten zu verschiedensten Preisen um Kundschaft. Die Integrierbarkeit in eigene Automatisierungskonzepte schwankt dabei stark: Einige Hersteller verkaufen zwar an sich geschlossene Systeme, öffnen aber immerhin Programmierschnittstellen zu Webdiensten, andere dokumentieren die verwendeten Protokolle zwar nicht, versuchen aber auch nicht, »Reverse Engineering« zu unterbinden. Ein wichtiger Aspekt ist auch der Produktlebenszyklus: Verliert ein Hersteller mangels Wirtschaftlichkeit das Interesse an einer Lösung, die einen kleinen Steuerungsserver im Heimnetz voraussetzt, und stellt die Aktualisierung seiner Smartphone-Apps ein, stehen Sie möglicherweise bei der nächsten Smartphone-Anschaffung im Regen, wenn keine kompatible App mehr verfügbar ist.

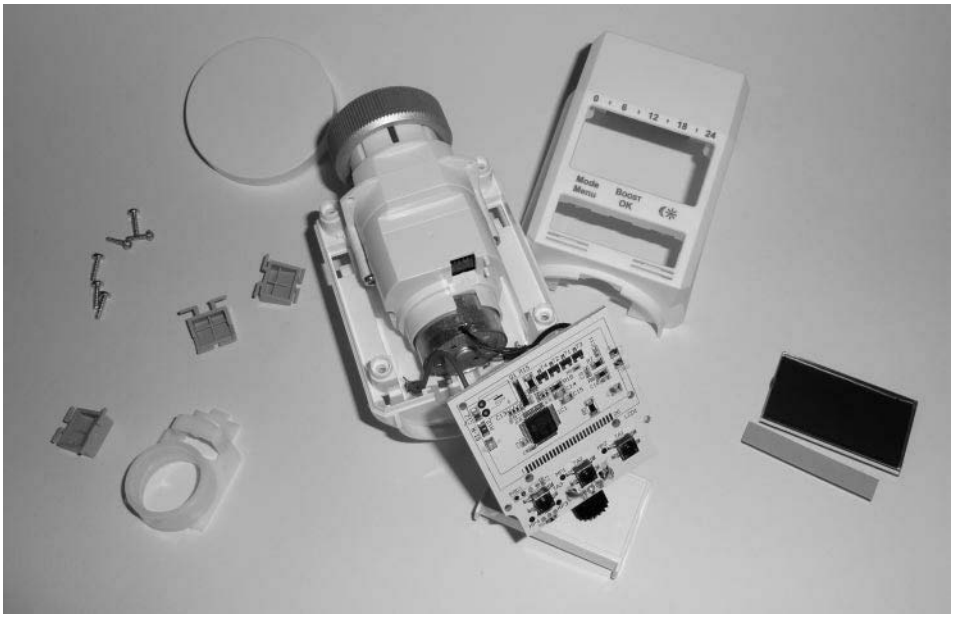


Bild 1.1: Ein einziger Mikrocontroller genügt: Programmierbare Thermostate sind seit über zehn Jahren billig auf dem Markt zu bekommen. Die einfache Hardware erlaubt Bastelei ohne Reue.

FS20

Eines der beliebtesten und am längsten am Markt erhältlichen Systeme ist das vom Versender ELV (und in geringerem Volumen von Conrad) vertriebene System FS20. Es beinhaltet Komponenten zur Steuerung von Steckdosen, Heizungsthermostaten und Wasserventilen sowie generische Aktoren. Auf Sensorseite stehen verschiedene Temperaturfühler, Wetterstationen, Lecksensoren, Bewegungsmelder und Türklingeln bereit. Das von FS20 verwendete Funkprotokoll auf 868 MHz ist zwar proprietär, die Schaltbefehle sind jedoch größtenteils vom Hersteller ELV selbst dokumentiert. FS20 integriert wegen der fehlenden Verschlüsselung nur sicherheitsunkritische Sensoren und Aktoren. Die Preise für FS20-Komponenten bewegen sich zwischen billiger Chinaware und aufwendigen IP-Steckdosen: 25 bis 30 Euro sind pro IP-Steckdose zu rechnen, dafür ist diese gut verarbeitet und bringt einen Schalter zum manuellen Aus- und Einschalten mit. Weit wichtiger für Bastler ist die gute Integrierbarkeit in selbst geschaffene Umgebungen: Neben USB-Sendern für den PC (die einen seriellen Port emulieren) existieren generische Sender und Empfänger, die über eine serielle Schnittstelle angesprochen werden können und sich damit für den Einsatz mit Arduino eignen. Allerdings hat FS20 mittlerweile seinen Zenit erreicht: Seit einer Weile werden keine neuen Sensoren und Aktoren mehr hinzugefügt, und seltener nachgefragte Komponenten wurden aus dem Programm genommen. Aufgrund der moderaten Preise und der guten Dokumentation kann der günstige Einkauf von FS20-Komponenten viel zur eigenen Automatisierungslösung beitragen – das ist sicherlich ein

Grund dafür, dass FS20 beim freien Home-Automatisierungsserver FHEM so beliebt ist. Beim Gebrauchtkauf von FS20-Komponenten sollten Sie darauf achten, dass diese nicht von LTE gestört werden. Der stark im Kommen begriffene Mobilfunkstandard verwendet Frequenzen bis 862 MHz, also knapp unter den 868 MHz, die FS20 nutzt. Ältere, sehr breitbandige Empfänger waren darauf nicht vorbereitet und schalten in der Umgebung von LTE-Mobiltelefonen und -Breitbandroutern nur sehr unzuverlässig.

Bild 1.2: Solide verarbeitet, leicht anzusteuern und relativ moderat im Preis: Funksteckdose aus dem FS20-System.

Homematic/eQ-3

Designierter Nachfolger des FS20-Systems ist Homematic. Auch Homematic funkt auf 434 MHz und benutzt ähnlich aussehende Sensoren und Aktoren. Softwareseitig kann Homematic als weitgehende Neuentwicklung verstanden werden. Als Steuerungszentrale für Homematic dienen nicht mehr simple, mikrocontrollerbasierte Fernsteuerungen, sondern ein »Cube« genannter kleiner Linux-Rechner, der Befehle per Webinterface und REST entgegennimmt und diese über ein proprietäres (teilweise verschlüsseltes Protokoll) an die Aktoren weitergibt.

Homematic/eQ-3 nutzt zwar manche Konzepte von FS20 – so auch den CuL genannten USB-Stick, der eine serielle Schnittstelle über Funk bereitstellt –, ist aber nicht protokollkompatibel. Das Homematic-Protokoll ist mittlerweile zum Teil nachvollzogen, wo dies nicht der Fall ist, ist es immerhin möglich, über den HTTP-Umweg Aktorzugriff auf den Cube zu erlangen. Dieser Weg ist teilweise in die freie Lösung OpenHAB integriert. Bei Redaktionsschluss wurden Homematic-basierte Lösungen stark von Internetprovidern wie Telekom und Mobilcom oder Energieversorgern wie RWE angeboten, sodass wir es auf jeden Fall für angebracht halten, die Integration solcher Systeme im Buch anzusprechen.

Belkin WEMO

Der Hersteller von (Heim-)Netzwerkequipment und PC-Zubehör Belkin bietet seit 2013 unter dem Label »WEMO« eine Hausautomatisierungsserie an, die bei Redaktionsschluss lediglich aus drei Funksteckdosen bestand: einer Funksteckdose, die ausschließlich als Aktor dient, einer weiteren, die zudem den Energieverbrauch angeschlossener Geräte messen und dokumentieren kann, und einer dritten, die zusätzlich mit Infrarotsensor ausgestattet ist. Eine Besonderheit bei WEMO ist die Integration in den Webdienst IFTTT (*If This Then That*), bei dem bausteinartig bestimmte Ereignisse bestimmte Aktionen auslösen können – ein Klassiker ist die E-Mail, die daran erinnert, dass die Waschmaschine gerade fertig geworden ist.

Bild 1.3: Hip: Belkins WEMO-Familie versucht, durch Cloud-Integration zu punkten, enthält aber bislang nur wenige Aktoren. (Pressefoto: Belkin)

AVM Fritz

AVM ist vor allem für seine DSL-Router mit und ohne integrierte Telefonanlage bekannt. Allerdings produziert AVM auch periphere Komponenten wie Powerline-Adapter oder DECT-Telefone. Relativ neu im Sortiment sind zwei schaltbare Steckdosen, die auch den Stromverbrauch messen sowie protokollieren und sich mit Webdiensten wie Googles Kalender kombinieren lassen. Das einfachere Modell wird per DECT gesteuert (und setzt daher eine FRITZ!Box mit Telefonanlagenfunktion voraus), das teurere Modell integriert einen Powerline-Adapter und einen WLAN-Repeater, wird über TCP/IP angesteuert und kann auch ohne FRITZ!Box arbeiten. Sein Einsatz ist vor allem dort sinnvoll, wo schaltbare Steckdose und Netzwerk zusammen benötigt werden – beispielsweise an einem PC-Arbeitsplatz, der nach Updates automatisch heruntergefahren und schließlich mitsamt Peripherie (Monitor, USB-Festplatten, Drucker) stromlos geschaltet wird. Morgens wird der Strom dann automatisch angeschaltet und der PC einige Minuten vor Arbeitsbeginn per Wake On LAN wieder aufgeweckt.

Pearl CasaControl

Das im badischen Buggingen ansässige Versandhaus Pearl sieht sich selbst als Schnittstelle zwischen Massenfertigern weltweit und Einzelabnehmern im deutschsprachigen Raum. Der Großteil der von Pearl angebotenen Produkte ist unverändert aus Fernost bezogenes Elektronikzubehör. Allerdings erlaubt es Pearls Marktmacht als Einkäufer auch, größere Lose mit eigenen Änderungen zu beziehen. So fasst Pearl in der Produktfamilie »CasaControl« verschiedene, eher generische Komponenten wie Funksteckdosen, per Funk geschaltete Lampenfassungen, Rollladen- und Heizkörpersteuerungen zusammen und bietet zentrale Steuerungskomponenten an. Interessant dabei ist, dass verschiedene am Massenmarkt etablierte und günstig umgesetzte Funkprotokolle zum Einsatz kommen (434 MHz Amplitudenmodulation für Steckdosen, 2,4 GHz für steuerbare LED-Lampen etc.). Pearl verspricht die Verwendung von Standardkomponenten, um bei Bedarf schnell den Zulieferer wechseln zu können. Für Anwender und Bastler dürfte sich CasaControl zu einer relativ günstigen, leicht in eigene Lösungen integrierbare Produktfamilie mausern. CasaControl-Testgeräte erreichten uns kurz vor Redaktionsschluss. Über unsere Erfahrungen berichten wir im Blog *www.arduino-hausautomation.de*.

China 434

Die am häufigsten anzutreffende Familie von Hausautomatisierungslösungen hat keinen Namen – sie verwendet zur Übertragung meist 434 MHz (Achtung, es sind auch 868-MHz-Geräte erhältlich) und wird unter Fantasie- oder Eigenmarken sehr billig verkauft: drei Steckdosen mit Fernbedienung in Baumärkten oder Discounteraktionen oft um 15 Euro, Dimmer einzeln unter 10 Euro. Letztlich kann damit alles angesteuert werden, was nicht mehr als acht Zustände benötigt und keine Rückkopplung erfordert. Das simple Protokoll und die günstige Verfügbarkeit von Bausteinen machen »China 434« für viele Einsatzzwecke zur ersten Wahl für Bastler.

Bild 1.4: Zweimal chinesische Billigware: links von Conrad im Dreierpack vertriebene Funksteckdosen, rechts eine spritzwassergeschützte Version vom Discounter.

1.3 Freie Projekte

Angesichts problematischer Interoperabilität kommerzieller Lösungen haben sich in den letzten Jahren einige freie Projekte zusammengefunden, die entweder verschiedene kommerzielle Produktfamilien auch nach deren Abkündigung unter einer gemeinsamen Nutzerschnittstelle zusammenfassen wollen oder gleich offene Hardware für die Hausautomation nutzen.

FHEM

Ein lange etabliertes System ist der in Perl geschriebene Hausautomatisierungsserver FHEM, der auf jedem Gerät läuft, das einen Interpreter für die Programmiersprache Perl mitbringt: FRITZ!Boxen mit modifizierter Firmware, PCs, DSL-Router, die auf OpenWRT basieren. Aufgrund der Geschichte des Projekts ist ein Schwerpunkt der Einsatz in Haushalten, die bereits über einige FS20- oder Max-Komponenten verfügen. Da auf dem Arduino Yún Perl nachinstalliert werden kann, ist der Einsatz von FHEM auf dem Yún denkbar und dann sinnvoll, wenn Sie schon über FS20-Komponenten verfügen und weniger basteln wollen.

OpenHAB

Ähnlich wie FHEM versucht OpenHAB, hersteller- und protokollagnostisch zu sein und damit verschiedenste Lösungen Dritter zu integrieren. OpenHAB beeindruckt durch ein aufwendiges, einfach zu bedienendes Webinterface, eine klare Nutzerführung und flexible Möglichkeiten, eigene Schaltregeln zu definieren. Kehrseite der Medaille ist, dass OpenHAB Java voraussetzt, also mindestens einen gut ausgestatteten NAS als Server benötigt – vereinzelt wurde schon der erfolgreiche Betrieb auf einem Raspberry Pi gemeldet. Sollten Sie irgendwann zum Schluss kommen, dass OpenHAB die für Ihr Netz geeignetste Serversoftware ist, verwenden Sie den Arduino Yún zum Steuern billiger Funksteckdosen und selbst gelöteter Komponenten, indem Sie ihn als Slave des OpenHAB-Servers per HTTP steuern.

Smarthomatic

Smarthomatic ist ein Projekt von Uwe Freese, der offene, günstig nachzubauende Hardware auf Basis von ATmega328-Mikrocontrollern etablieren möchte. Die Kommunikation findet dabei über RFM12B-Funkmodule statt, eine Besonderheit ist jedoch, dass AES-256-Verschlüsselung zum Einsatz kommt. Ein 24-Bit-Paketzähler sorgt in Verbindung mit dem Rückkanal des verwendeten Funkmoduls dafür, dass jede Nachricht nur einmal gültig ist, dabei reizt Smarthomatic die Grenzen des ATmega328 sehr geschickt aus. Bislang realisierte Hardwarekomponenten sind Schalter, Dimmer und Erdfeuchtesensoren (bei Letzteren erschließt sich mir der Sinn einer verschlüsselten Datenübertragung nicht unbedingt). Die von Freese entwickelte Basis wird per USB-nach-TTL an jedes USB-fähige Gerät angeschlossen. Damit eignet sich Smarthomatic

zumindest für sicherheitskritische Anwendungen auch für Komponenten in dem hier im Buch beschriebenen Beispielnetz.

1.4 Der Maker-Arbeitsplatz

Der ideale Arbeitsplatz für Elektronikbasteleien ist ein trockener, staubfreier Hobbykeller, in dem zur Arbeit an Projekten benötigte Gerätschaften auch einmal über Nacht liegen bleiben können und vielleicht sogar genug Platz vorhanden ist, Testläufe von Platinen tagelang durchzuführen, ohne dass diese bei weiteren Arbeiten stören. So etwas funktioniert auf kleinem Raum – in diesem Zusammenhang erinnere ich mich an meine Ausflüge als Kind in Opas Bastelkeller. Auf weniger als zwölf Quadratmetern brachte der gelernte Uhrmacher alles unter, was er für feinmechanische und elektronische Basteleien, die Entwicklung von Elektronik und die Fertigung von Kleinserien benötigte. Dabei entstanden nach seiner Verrentung im Unruhestand noch einige Patente in diesem Bastelkeller. Möglich war das durch eine sehr disziplinierte Ordnung in Batterien von Kleinteileregalen, in denen Widerstände, Kondensatoren sowie Transistoren aller Formen, Größen und Nennwerte so sortiert waren, dass nicht einmal eine Beschriftung der Schubladen nötig war.

Arbeitsplatz

Nun werden Sie nicht unbedingt ein ganzes Zimmer für Elektronikbasteleien zur Verfügung haben, dennoch ist ein separater Arbeitsplatz sinnvoll, um Auf- und Abbauaufwand zu sparen. Basis hierfür kann ein einfacher IKEA-Schreibtisch sein – 50 × 70 cm² in Kiefer ohne Schublade ab 15 Euro – besser sind 60 cm Tiefe und ein paar Schubladen. Eine Antistatikmatte als Arbeitsunterlage schadet nicht, ist aber nicht zwingend erforderlich, wenn Sie sich beim Platznehmen immer an einem geerdeten Gegenstand entladen. Ist kein eigener kleiner Elektroniktisch möglich, lassen Sie sich eine Buche-Stäbchenplatte oder Multiplexplatte in der Tiefe Ihres Schlafzimmerschranks und in 60 bis 70 cm Breite zuschneiden. Bringen Sie rechts, links und hinten eine quadratische oder halbrunde Leiste an, um das Herunterfallen von Bauteilen zu verhindern, und nähen Sie aus Baumwoll- oder Leinenstoff (wegen der geringeren statischen Aufladung) eine Abdeckung, die Sie rundherum mit einem Klettverschluss anbringen können. Mit solch einer Konstruktion können halb fertige Komponenten auch mal zwei oder drei Wochen auf dem Schlafzimmerschrank überdauern, ohne einzustauben oder verloren zu gehen.

Aufbewahrung

Für die Aufbewahrung von Bauteilen sind Kleinteileboxen oder Sortimentskästen im Kofferformat meist ganz praktisch. Für sehr kleine Bauteile sind Fachtiefen von rund 2½ cm gut geeignet, weil dies das Herausfischen erleichtert. Zweiseitige Kleinteilekoffer bringen mit diesem Aufbau 20 verschiedene Fächer mit – die sich durch

Kombination nicht verwechselbarer Bauteile (LEDs in Rot teilen sich ein Fach mit 2,2-kΩ-Widerständen und grüne mit 1,8 kΩ) effizienter nutzen lassen. Zwei solche Koffer sollten ausreichen, den Kleinteilefundus vieler Bastler abzudecken. Schubladenbatterien sind meiner Ansicht nach eher schlecht geeignet, da Spalten bleiben, die Bauteile verschlucken. Bei einem permanenten Elektronikarbeitsplatz können Sie zudem daran denken, für häufig benötigtes Schüttgut Kaviar-(Ersatz-), Leberwurst- oder Marmeladengläser mit dem Deckel von unten an ein an der Wand befestigtes Brett anzuschrauben. Babybreigläser sind mit ihren acht oder zwölf Gewindegängen ungeeignet.

Bild 1.5: In Kleinteilekoffern bringen Sie eine Menge elektronischer Komponenten unter – Fächer für Mikrocontroller sollten Sie mit antistatischem Schaumstoff auspolstern.

Löten und messen

Viele Arduino-Projekte gelingen ohne oder mit geringem Lötaufwand, denn kaum etwas hält länger als ein Provisorium. Wenn es die Platzverhältnisse zulassen, eine permanente Stromversorgung per Netzteil möglich ist und keine Vibrationen auftreten, spricht nichts dagegen, einen Arduino Uno nebst Breadboard im Objekt der Automatisierung unterzubringen. Bei Hausautomatisierungslösungen ist das oft nicht möglich, weil keine Kabel erwünscht sind, eine Schaltung klein sein muss oder aus Kostengründen kein kompletter Uno zum Einsatz kommen kann, sondern eine Miniplatine die Basis bildet. Das bedeutet: Es muss gelötet werden. Zum Löten empfehlen wir eine günstige, regelbare Lötstation (ab ca. 25 Euro), unregelbare Lötkolben aus

dem Baumarkt sind oft zu heiß für empfindliche Halbleiter. Lötzinn mit Flussmittelkern sollte in 0,5 mm und in 1 mm Durchmesser bereitstehen. Als Privatperson können Sie für den »Eigenbedarf« bleihaltiges Lötzinn erwerben, dieses lässt sich leichter verarbeiten als das gesündere, ROHS-konforme bleifreie. Aufpassen müssen Sie aber beim Anlöten oder Nachlöten bleifreier Verbindungen: Geschieht dies mit bleihaltigem Lot, steigt die Empfindlichkeit gegen Kalt-warm-Zyklen und die Gefahr der Korrosion merklich – sicher kein Problem bei Bauteilen, die in der Wohnung betrieben werden, wohl aber bei Sensoren, die im Außenbereich mit Temperaturbereichen zwischen –20 °C und +40 °C sowie mit Luftfeuchtigkeit konfrontiert sind. Achten Sie bei der Arbeit mit bleihaltigem Lot darauf, dass am Arbeitsplatz weder Nahrungsmittel noch Getränke stehen, und waschen Sie sich vor dem Verzehr des »Pausenbrots« gründlich die Hände. Am Lötplatz sollte eine separate kleine Dose Flussmittelpaste bereitstehen. Um Zinnbrücken zu entfernen, ist Entlötlitze von Vorteil. Für die Arbeit auf Lochrasterplatinen benötigen Sie zudem verzinnten Kupferdraht. Praktisch ist eine »dritte Hand«, die Bauteile und Platinen für die Verlötung in Position zueinander hält. Ob Sie die bei vielen dritten Händen mitgelieferten Lupen einsetzen, bleibt Ihnen überlassen. Mitunter steckt bei der Arbeit an Arduino & Co. der Teufel im Detail. Insbesondere wenn Sie planen, per Funk meldende autarke Sensoren zu verwenden, muss der Energieverbrauch für den Akku- oder Solarbetrieb massiv gedrosselt werden. Es spielt dann eine große Rolle, ob Ihr fertiger Schaltungsaufbau in Kombination mit dem geplanten Sketch 10 mA, 1 mA oder weniger als 1 µA Strom aufnimmt – alles sind typische Leistungsaufnahmen im Leerlauf. Ein Multimeter, dessen Einstellung bis auf 20 µA herunterreicht, sollte daher vorhanden sein.

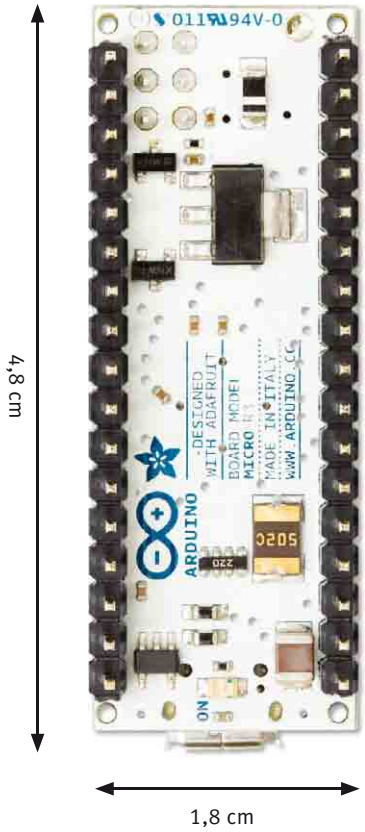
Bild 1.6: Erbstück: Weller-Lötstationen können auf eBay schon mal dreistellige Preise bringen. Regelbare chinesische Billigware ist für weniger als die Hälfte zu haben.

Dr. Günter Spanner

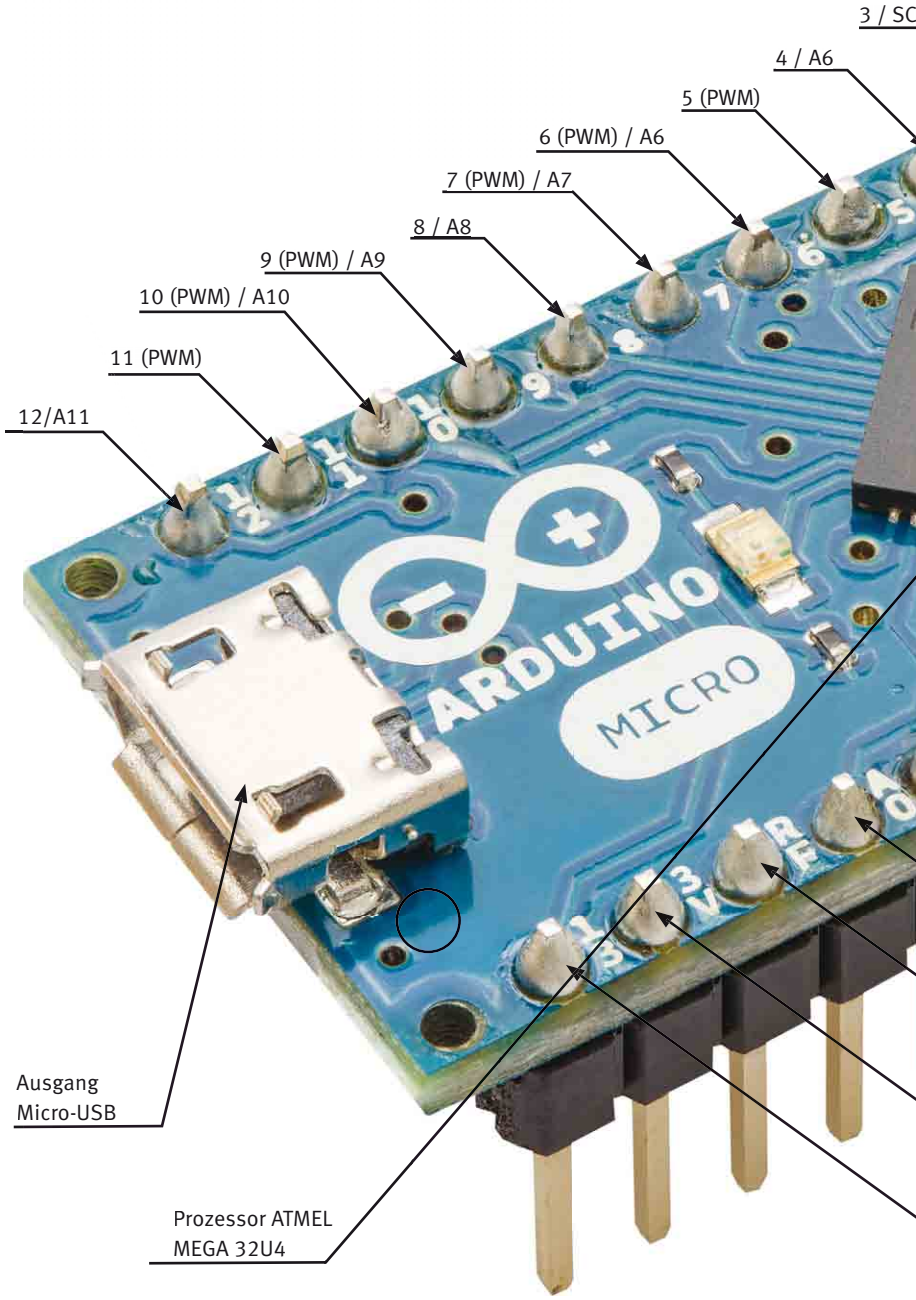
Coole Projekte mit dem Arduino™ Micro

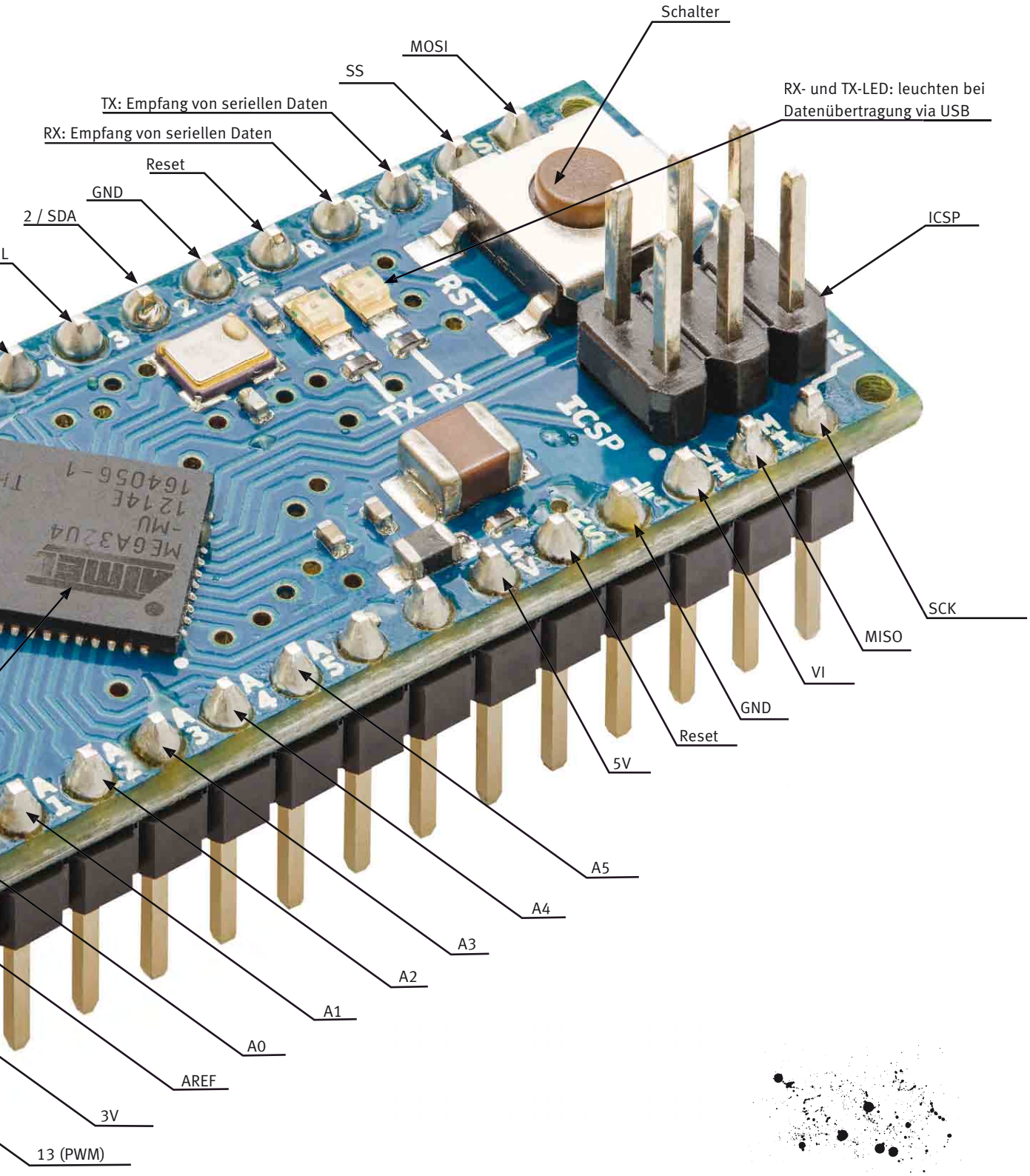
Physical Computing im Projekteinsatz
Alles für den Einstieg: Platine, ATmega32U4, USB-Zugriff und Programmierung
Projekte für die Praxis: Lichteffekte, Designeruhr, Motorensteuerung, Messen, Steuern und Regeln

Experten erfassen den Aufbau des Arduino Micro schnell. In seinem Aufbau unterscheidet er sich aber doch sehr von den anderen Arduino-Modellen. Deshalb lohnt sich der Blick auf die Details.



Gewicht:
6,6 Gramm




Schalter
MOSI
SS
TX: Empfang von seriellen Daten
RX: Empfang von seriellen Daten
Reset
GND
2 / SDA
RX- und TX-LED: leuchten bei Datenübertragung via USB
ICSP
SCK
MISO
VI
GND
Reset
5V
A5
A4
A3
A2
A1
A0
AREF
3V
13 (PWM)

2 Die Technik des ATmega32U4

Bereits in den frühen 70er-Jahren begannen die ersten Firmen mit der Entwicklung der Mikrocontrollertechnik. Sie vollzogen damit den Übergang von der Technik der verdrahteten Analog- und Logikschaltungen hin zur Welt der hochintegrierten Elektronikkomponenten.

Die ersten Controller wurden für ein spezielles Anwendungsgebiet entworfen und konnten zunächst ausschließlich in den für sie vorgesehenen Bereichen eingesetzt werden. Waren neue Aufgaben zu lösen, wurde die Entwicklung eines vollständig neuen Bausteins erforderlich oder man musste bereits vorhandene Spezialcontroller mit entsprechend hohem Aufwand erweitern oder anpassen.

2.1 Rückblick auf die Mikrocontrollerentwicklung

Erst mit der Entwicklung programmierbarer Logikeinheiten, den sogenannten *Field Programmable Gate Arrays* (FPGAs) konnte man flexibel auf sich ändernde Anforderungen reagieren. FPGAs bestehen im Prinzip aus konfigurierbaren Logikblöcken mit einem zugeordneten Befehlsspeicher. Als sich elektronische Komponenten in vielen Bereichen der Wissenschaft und Technik durchsetzten, waren diese ersten Vorläufer der Mikrocontrollertechnik bestens geeignet, um schnell und präzise auf Veränderungen in der Betriebsumgebung zu reagieren.

Im Rahmen dieser Entwicklung wurde nun auch erstmals zwischen zwei wesentlichen Entwicklungsbestandteilen unterschieden. Einerseits war die Hardware ein fest vorgegebener, nachträglich nicht mehr veränderbarer Teil einer Schaltung, andererseits konnte über die Soft- oder auch Firmware, also über die in den Befehlsspeichern enthaltenen Programmanweisungen, bereits fertiggestellte Hardware an jeweils neue Einsatzgebiete angepasst werden.

Miniaturisierung und Zusammenfassung der nun auf sogenannte »Chips« integrierten Mikroschaltungen führten zusammen mit immer weiteren Funktionseinheiten schließlich zur Entwicklung der ersten Mikrocontroller (µC). So bestanden die ersten Controller nur aus Prozessorkernen mit integrierten Speichereinheiten. Als Speicher waren ursprünglich nur lesbare und wiederbeschreibbare Speicher (ROM und RAM) verfügbar. Die wiederbeschreibbaren Speicher hatten zudem den Nachteil, dass sie ihren Inhalt nach dem Abschalten der Versorgungsspannung verloren. Erst viel später wurden mit dem Aufkommen der Flashspeicher wiederbeschreibbare, nicht flüchtige Speichereinheiten verfügbar.

Nachdem die ersten zuverlässigen Controllereinheiten in großen Stückzahlen hergestellt werden konnten, trennte sich die Entwicklung in die Richtungen Mikrocontroller und Mikroprozessoren. Prozessoren wurden für die klassischen Anwendungen wie Großrechenanlagen und später auch für den sogenannten Mikrocomputerbereich benötigt.

Controller kamen immer häufiger in sogenannten *Embedded Systems*, d. h. vollständig integrierten Systemeinheiten zum Einsatz. Hierzu zählen Anwendungen im Automobilbereich genauso wie in der Produktionstechnologie. Aber auch die modernen Anwendungen wie Smartphones oder intelligente Haushaltsgeräte sind nichts anderes als Embedded Systems, die erst durch die aktuelle Controllertechnologie möglich wurden.

Allgemein kann man eine integrierte 4-Bit-Einheit der Firma Texas Instruments als den ersten Vorläufer der modernen Controllertechnik betrachten. Die Anwendungsgebiete dieser ersten 4-Bit-Controller waren zunächst einfache Rechenmaschinen und später die ersten, noch wenig leistungsfähigen Taschenrechner.

Abb. 2.1: Früher Taschenrechner mit Mikrocontrollertechnologie

Auch Intel stellte schon bald parallel zu den bekannten Prozessoren auch Controller her. Hierzu zählten die Typen 8048 und 8051. Der seit den 80er-Jahren verfügbare 8051-Controller wurde später zum Goldstandard der Controllertechnik und schließlich sogar zu einem der am weitesten verbreiteten elektronischen Bauelemente überhaupt.

Durch die weite Verbreitung wurde der 8051 später auch von einer Vielzahl anderer Hersteller in Lizenz nachgebaut. Auch Atmel gehört zu diesen Lizenzherstellern. So sind die AT89-Typen von Atmel prinzipiell nichts anderes als 8051-Typen mit Flashspeicher, und letztendlich basieren auch die modernen Varianten wie die ATmega-, ATtiny- und ATX-Serien immer noch auf den Grundlagen des Urvaters 8051.

2.2 Funktionseinheiten des ATmega32U4

Aktuelle Mikrocontroller wie den ATmega32U4 kann man als *Ein-Chip-Computer* bezeichnen. Mit ihnen lassen sich komplexe Aufgaben lösen, für die sonst ein aufwendiger Schaltungsaufbau notwendig wäre. Die Controller der ATmega-Serie sind sogar programmierbar, wenn sie bereits fest in eine Schaltung eingebaut sind. Man spricht hier von ISP(In System Programming)-fähigen Controllern.

Mikrocontroller enthalten immer die folgenden Hauptbestandteile:

- CPU
- SRAM
- Flashspeicher für Programmcode

Dazu kommt bei allen modernen Controllersystemen eine umfangreiche Peripherie. Bei den ATmega-Controllern sind standardmäßig die folgenden Einheiten integriert:

- analoge und digitale Ports
- mehrere AD/DA-Wandler
- Timer und Counter
- Schnittstellen zur Kommunikation mit der Außenwelt

Mikrocontroller werden in der Regel mit Taktfrequenzen von 1 MHz bis ca. 20 MHz betrieben, entsprechend hochpreisige 32-Bit-Modelle sogar mit weit über 100 MHz. Auf dem Arduino Micro liegt die Taktfrequenz mit 16 MHz im Mittelfeld.

Die Bitzahl eines µC bezeichnet die Breite des internen Datenbusses. Typische Werte sind 8 Bit, 16 Bit und 32 Bit. Hier liegt der ATmega32U4 am unteren Ende der Skala. 8-Bit-Controller sind aber weitverbreitet und reichen für Hobby- aber auch viele professionelle Anwendungen vollkommen aus. Erst bei hochkomplexen und rechenintensiven Aufgaben werden 8-Bit-Systeme immer häufiger von den 16- und 32-Bit-Versionen abgelöst.

In der nachfolgenden Abbildung ist der allgemeine schematische Aufbau eines Mikrocontrollers dargestellt.

Häufig werden Mikrocontroller in Harvard-Architektur realisiert. Dabei sind Daten- und Programmspeicher physikalisch voneinander getrennt. Der Vorteil gegenüber der alternativen Von-Neumann-Architektur, bei der Daten und Programmcode im selben Speicherbereich liegen, ist die Möglichkeit, mit einem einzigen Taktzyklus einen Befehl und die dazugehörigen Daten in die CPU zu holen. Daher wird diese Architektur bevorzugt bei RISC-Kernen verwendet, so auch beim ATmega32U4. Er verfügt über eine Advanced-RISC-Architektur mit 135 leistungsstarken Befehlen. Die meisten dieser Anweisungen können mit einem einzelnen Taktzyklus ausgeführt werden. Das bedeutet, dass der Controller bei einer Taktfrequenz von 16 MHz bis zu 16 MIPS (Mega Instructions per Second) abarbeiten kann.

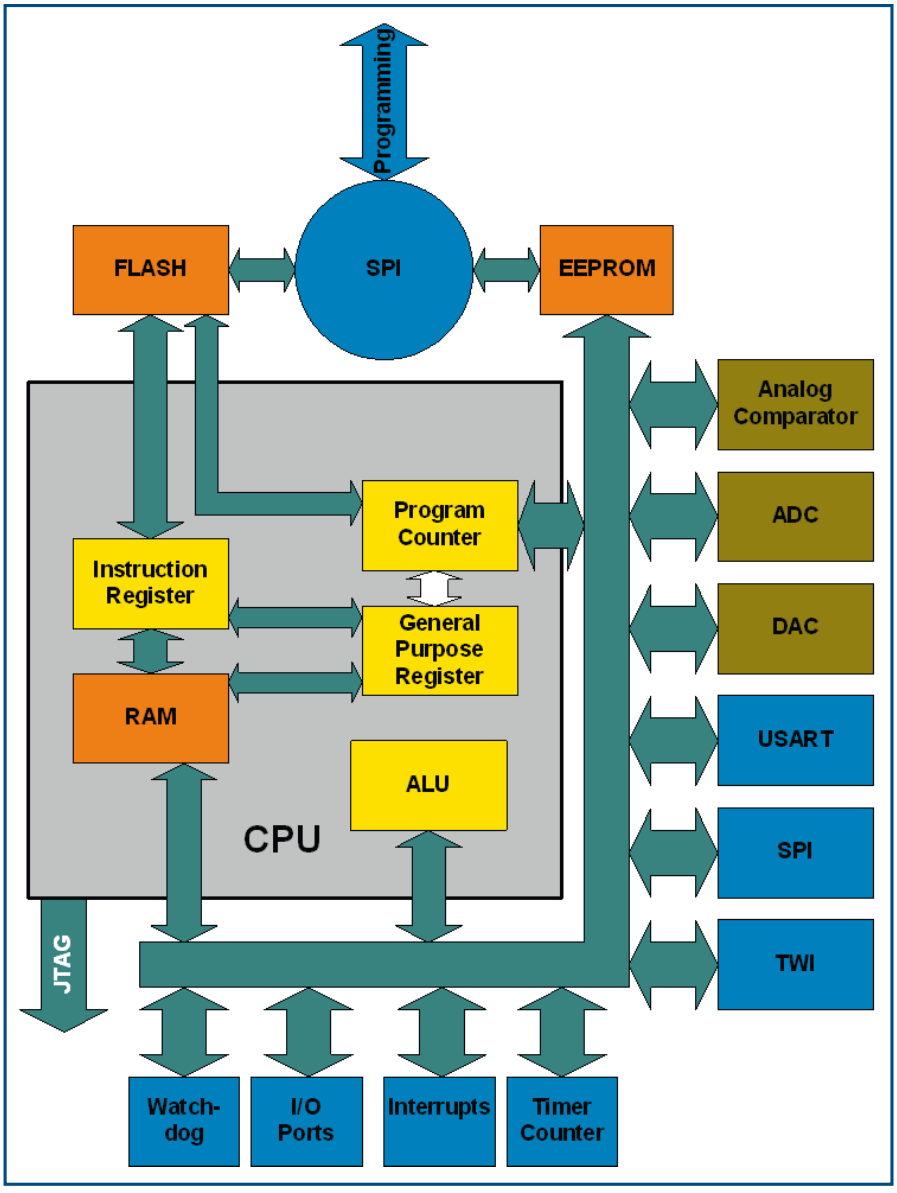


Abb. 2.2: Aufbau eines AVR-Mikrocontrollers

Im Folgenden werden die einzelnen Komponenten eines Controllers und die ihnen zugeordneten Aufgaben näher erläutert.

Die CPU (Central Processing Unit) ist die zentrale Verarbeitungseinheit eines Prozessors oder Controllers. Gelegentlich wird sie auch einfach als *Prozessor* bezeichnet. In der CPU wird ein Programmbefehl nach dem anderen abgearbeitet.

Für die sequenzielle Abarbeitung von Anweisungen enthält die CPU ein Steuerwerk, das jeden einzelnen Befehl in zeitlich aufeinanderfolgende Bearbeitungsschritte umsetzt. Bei Operationen mit arithmetischen oder logischen Verknüpfungen wird zusätzlich das Rechenwerk miteinbezogen.

Die für die Befehlsbearbeitung notwendigen Operanden werden im Allgemeinen den CPU-internen Speicherplätzen, den sogenannten Registern, entnommen. Die Register wiederum werden aus den externen Speicherplätzen, insbesondere dem RAM, versorgt.

Externe Speicherplätze sind über das Bussystem mit der CPU verknüpft. Dieses Bussystem wird auch für die Kommunikation der CPU mit anderen Elementen des Controllers, also etwa den Ports oder der Peripherie, genutzt.

Ein spezieller Fall ist die Option, dass periphere Komponenten, also z. B. ein ADC, ohne Teilnahme der CPU-Daten über einen Bus beispielsweise in das RAM senden können. Dieses Verfahren wird auch als DMA (Direct Memory Access) bezeichnet. DMA gestattet es also z. B. analoge Messwerte schnell zu erfassen und im Speicher abzulegen.

Die Koordination aller Bussignale übernimmt ein eigener sogenannter Steuerbus. Hier wird u. a. auch die Übertragungsrichtung der Bussignale überwacht.

In modernen Controllern gibt es drei Arten von Speichern:

Typ	Eigenschaft	Anwendungen	Verfügbare Speichergröße im ATmega32U4
Flash	Permanentspeicher, nichtflüchtig	Hier werden Programmanweisungen und spezielle Daten abgelegt	32 kB
EEPROM	Nicht flüchtiger Speicher für wenig veränderliche Daten	Speicherung von Kalibrationskonstanten für A/D-Wandler oder Sensoren etc.	1 kB
RAM	Flüchtiger Arbeitsspeicher	Speichern von Variablen und Werten, die im Programmverlauf häufig verändert werden	2,5 kB, internes SRAM

Tab. 2.1: Speichertypen und deren Größe im ATmega32U4

Im Speicher des Controllers werden sowohl die Programme als auch die Daten, also Variablen und Konstanten, für die CPU bereit gehalten. Moderne Controller enthalten mehrere Kilobytes Flashspeicher. Diese Speicherart zeichnet sich dadurch aus, dass ihr Inhalt auch nach dem Abschalten der Betriebsspannung erhalten bleibt. Ein einmal in den Flashspeicher geschriebenes Programm steht also so lange für den Controller zur Verfügung, bis es durch ein neues ersetzt wird. Die Größe des Flashspeichers im ATmega32U4 wird von Atmel durch die »32« in der Typenbezeichnung des Controllers angegeben.

Im RAM(Random Access Memory)-Speicher werden während der Ablaufzeit des Programms alle Variablen und alle sich häufig verändernden Daten abgelegt, da ein RAM-Speicher praktisch beliebig oft beschrieben und gelesen werden kann. Die im RAM abgelegten Daten gehen allerdings nach dem Abschalten der Versorgungsspannung verloren.

Schließlich enthält praktisch jeder Controller noch ein sogenanntes EEPROM (Electrically Erasable Programmable Read Only Memory). Dieser Speichertyp ist nicht flüchtig, d. h., die darin enthaltenen Daten bleiben auch ohne angelegte Betriebsspannung erhalten. Hier werden beispielsweise wichtige Kalibrationsdaten für externe Sensoren etc. abgelegt.

Für die Permanentspeicher werden von Atmel 10.000 Schreib-Lese-Zyklen für den Flashspeicher und 100.000 Lesezyklen für das EEPROM garantiert. Die Daten bleiben bei normaler Raumtemperatur (25 °C) mindestens 100 Jahre lang erhalten. Bei erhöhten Temperaturen von beispielsweise 85 °C, wie sie z. B. im Automotive-Sektor häufig auftreten, verringert sich dieser Zeitraum auf 85 Jahre.

Darüber hinaus verfügt der ATmega32U4 über einen optionalen Boot-Code-Sektor, der über spezielle Lockbits gesichert werden kann.

Jeder Controller benötigt einen sogenannten Takt. Dieser besteht aus einem rechteckförmigen Signal, das den zeitlichen Ablauf in einem Controller steuert. Bei jedem Pegelwechsel des Taktsignals geht der Mikrocontroller von einem elektrisch stabilen Zustand in den nächsten über.

Im Idealfall wird bei jedem Takt ein Controllerbefehl ausgeführt. Bei verschiedenen Controllertypen werden aber auch mehrere Taktzyklen für die Abarbeitung eines einzelnen Befehls benötigt.

Im Arduino Micro wird der ATmega32U4 mit seiner maximalen Taktrate von 16 MHz betrieben. Deshalb muss der Chip auch mit 5 V versorgt werden. Bei geringeren Versorgungsspannungen sinkt die maximal zulässige Taktrate auf 8 MHz ab. Um eine hohe Frequenzstabilität zu gewährleisten, wird der Takt über einen externen 16-MHz-Quarz erzeugt. Auf diese Weise wird auch ein sicherer Betrieb der USB-Schnittstelle gewährleistet.

Die Eingabe/Ausgabe(I/O)-Ports (I/O = Input/Output) stellen die Schnittstelle des Controllers zur Außenwelt dar. Der ATmega32U4 verfügt über 26 programmierbare I/O-Pins. Sie können jeweils entweder als Ein- oder als Ausgänge konfiguriert werden.

Im Unterschied zu einem Mikroprozessor verfügt ein Controller über integrierte Peripherie-Einheiten. Die wichtigsten peripheren Systeme wie Timer, serielle Schnittstellen oder Pseudoanalogausgänge (PWM) sind in jedem

modernen Controller vorhanden. Die folgende Tabelle zeigt eine Übersicht über die wichtigsten allgemeinen Peripherieeinheiten des ATmega32U4.

Um schnell auf unvorhergesehene Ereignisse reagieren zu können, enthalten Mikrocontroller auch sogenannte Interrupteinheiten. Damit kann erreicht werden, dass der Controller nahezu verzögerungsfrei auf bestimmte Signale reagiert. Das ist beispielsweise bei Alarmsignalen von großer Bedeutung. Der ATmega32U4 besitzt keine speziellen Interrupt-Pins. Vielmehr kann jeder I/O-Pin so konfiguriert werden, dass er bei einem Pegelwechsel einen Interrupt auslöst.

Zur Standardperipherie kommen häufig weitere wichtige Komponenten wie Analog-digital-Wandler (ADCs) oder Digital-analog-Umsetzer (DACs) hinzu. Tab. 2.2 gibt einen Überblick über die Spezialperipherie-Einheiten im ATmega32U4.

Während bei früheren Controllerfamilien nur eine RS-232-kompatible Schnittstelle zum Standard gehörte, finden sich in modernen Controllern immer häufiger auch integrierte USB-Einheiten. Damit wird der Tatsache Rechnung getragen, dass sowohl bei den PCs als auch im Bereich professioneller Steuer- und Regelungssysteme die RS-232-Schnittstelle ausstirbt und durch USB-Systeme ersetzt wird. So verfügt auch der ATmega32U4 über eine integrierte USB-Einheit. Das wird durch das »U« in der Typenbezeichnung angedeutet.

Einheit	Anwendung	Verfügbare Einheiten im ATmega32U4
Timer	Messung von Zeitintervallen, Frequenzen oder Periodendauern	1x 8-bit Timer/Counter mit separatem Prescaler und Compare Mode, 2x 16-bit Timer/Counter mit separatem Prescaler, Compare Mode und Capture Mode, 1x 10-bit High-Speed Timer/Counter mit 64 MHz-PLL und Compare Mode
I/O-Ports	Ein- und Ausgabe von digitalen Signalen	26 Programmierbare I/O-Pins Alle I/Os sind als CMOS-Ausgänge und LVTTL-Eingänge konfigurierbar
Interrupts	Schnelle Reaktion auf unvorhersehbare Ereignisse	Interrupt-Auslösung durch Pegeländerung an einem beliebigen I/O-Port

Tab. 2.2: Allgemeine Peripherie-Einheiten des ATmega32U4

Neben der Standardperipherie finden sich beim ATmega32U4 noch weitere nützliche Systemeinheiten. Damit wird der Anwendungsbereich des Controllers deutlich erweitert und die Anzahl eventuell notwendiger externer Komponenten stark reduziert.

Einheit	Anwendung	Verfügbare Einheiten im ATmega32U4
USART und serielle Schnittstellen	Übertragung von Daten vom Controller zu einem Rechner	Programmierbare serielle USART mit Hardware Flow-Control, Master/Slave SPI Serial Interface, Byte-orientiertes 2-wire serielles Interface
USB-Schnittstelle	Datenübertragung über den Universal Serial Bus	Integriertes USB-2.0-full-speed- und low-speed-Device-Modul, vollständig kompatibel mit Universal-Serial-Bus-Spezifikation Rev. 2.0, Unterstützung von Datentransferraten von bis zu 12 Mbit/s
PWM-Kanäle	Erzeugung von Quasianalog-signalen	4x 8-bit PWM-Kanäle,mit programmierbarer Auflösung von 2 bis 16 Bits, 6 PWM-Kanäle für High Speed Operationen mit einer programmierbaren Auflösung von 2 bis 11 Bits
Analog-digital-Converter	Erfassung von analogen Spannungspegeln	12-Kanal-ADC mit 10-bit-Auflösung mit differenziellen Eingängen und programmierbaren Verstärkungsfaktoren

Tab. 2.3: Spezialperipherie im ATmega32U4

Manuel Di Cerbo / Andreas Rudolf

Android mit Arduino™ Due

Kommunikation zwischen Adroid-Geräten und Arduino™ Due
Steuern Sie Ihren Arduino™ Due mit einem Android-Gerät
Praxisbeispiele zeigen, wie es geht

Inhaltsverzeichnis

1 Einleitung

1.1 Android

Android ist ein Betriebssystem, das besonders im Smartphone- und Tablet-Bereich eingesetzt wird. Entwickelt wird es hauptsächlich von der sogenannten *Open Handset Alliance*, die von Google angeführt wird.

Im Kern von Android befindet sich der Linux-Kernel, der gänzlich unter der Open-Source-Lizenz *General Public License* (GPL) entwickelt wird. Der Linux-Kernel ist ein Stück Software, um unter anderem Hardware anzusteuern und Speicher zu verwalten. Außerdem werden Prozesse und Rechte zur Verfügung gestellt. Generell kann der Linux-Kernel als »Fundament« des Betriebssystems aufgefasst werden. Bis vor Kurzem wurde Linux vor allem in Server- und Embedded-Systemen verwendet. Dies änderte sich aber schlagartig in den letzten Jahren mit der rasanten Verbreitung von Android.

Linux
Linux ist ein Betriebssystemkern, der vom Hacker und Bastler Linus Torvalds in seiner Studienzeit entwickelt wurde. Mit der Lizenzierung unter GPL ermöglichte er die Verbreitung seines Systems ohne Restriktionen, weil er Entwicklern am System möglichst viel Freiraum geben wollte. Heute verbessern tausende Entwickler den Linux-Source-Code. Der »Vanilla-Kernel«, ein Referenz-Kernel von Linux, wird in regelmäßigen Abständen von Linus Torvalds veröffentlicht. Auch führende Technologiekonzerne wie Intel, ARM, Microsoft und IBM arbeiten am Linux-Kernel und tragen mit »Patches« zu dessen Weiterentwicklung bei.

Wie der Kernel ist der Android-Source-Code auch als »Open Source« lizensiert. Allerdings wird für die Bestandteile von Android, die außerhalb des Kernels liegen, eine weniger restriktive Lizenz (zumeist Apache oder LGPL) verwendet. Dies erlaubt es den Hardware-Vertreibern, Modifikationen vorzunehmen, ohne dazu verpflichtet zu sein, den dazugehörigen Source-Code zu offenbaren. Primär hat dies zum Ziel, den Wettbewerb zwischen den einzelnen Vertreibern anzukurbeln und die Plattform wirtschaftlich für Hardware-Hersteller (meist auch Mitglieder der *Open Handset Alliance*) interessant zu machen.

GPL
GPL ist eine »Copyleft«-Lizenz. Bei der Erweiterung von GPL-lizenziertem Source-Code muss dieser mit der gleichen GPL-Lizenz weitergegeben werden. Dies führt dazu, dass alle Änderungen, die am Source-Code vorgenommen werden, dem originalen Verfasser mitgeteilt werden müssen. Bei einer Apache-Lizenz ist das nicht nötig, allerdings muss bei Weiterentwicklungen eine Notiz vorhanden sein, dass der Source-Code auf einem Apache-lizenzierten Ausgangsprojekt beruht.

Das Hauptinteresse von Google ist es, die Android-Plattform zu verbreiten, sodass Endbenutzer in Zukunft möglichst viele Dienstleistungen über den Suchmaschinengiganten in Anspruch nehmen. Zusätzlich bemüht sich Google, ein »Entwickler-Ökosystem« zu pflegen, das als zentralen Bestandteil den *Google Play Store* hat. Entwickler sollen in der Lage sein, gewinnbringend Applikationen – die bekannten Android-Apps – zu vermarkten, und Google ist dabei am Erfolg beteiligt (30 Prozent des Preises einer App gehen beim Kauf an Google).

Die »Google Play«-App selbst ist – wie auch die Apps »GMail« und »Google Maps« – nicht Open Source und wird dem Kunden nur zur Verfügung gestellt, wenn die Hardware und die Android-Implementierung den Standards der *Open Handset Alliance* entsprechen. Dies erfolgt aus Gründen der Qualitätssicherung.

Android ist für Entwickler interessant, da es statt klassischer Softwareentwicklung eine nahezu geniale Einbettung von Applikationen in das Betriebssystem erlaubt. Mit akribisch genauer Dokumentation und guten Tutorials für das *Android Software Development Kit* (Android SDK) motiviert Google Entwickler, das Betriebssystem als Basis für ihre nächste große App zu verwenden.

Android SDK
Das SDK ist zentraler Bestandteil der Android-Philosophie. Auch mitgelieferte Apps wie »Telefon«, »E-Mail« und »Browser« basieren auf dem SDK. Regelmäßig veröffentlicht Google neue Versionen des SDKs, die es den Entwicklern erlauben, neue Features in ihre Apps einzubauen.

Als Programmiersprache für Android-Apps wird überwiegend *Java* verwendet. Mit der Möglichkeit, nativen Code in *C/C++* zu schreiben und diesen via *Java Native Interface* (JNI) einzubinden, existiert zusätzlich die Option, zeitkritische Komponenten einer App auszulagern. In diesem Buch werden wir uns allerdings auf die klassische Android-Entwicklung mit Java konzentrieren.

Der Open-Source-Charakter des Betriebssystems zieht sehr viele Entwickler an. Allein schon die Möglichkeit, in den Source-Code des Betriebssystems Einblick zu nehmen, um Software-Patterns zu lernen und Bestandteile eines modernen Systems zu erkunden,

begeistert viele Software-Entwickler. Auch der Bestandteil Linux ist ein Faktor, warum Bastler und Hobbyisten gerne mit Android arbeiten. Denn für Linux gibt es Hardware-Treiber wie Sand am Meer – und alle sind frei verfügbar.

1.2 Arduino

Die Marke *Arduino* hat vieles gemeinsam mit Android. Arduino ist eine Entwicklerplattform für Hardware. Neben den Hardware-Boards, die mit leistungsfähigen Mikrocontrollern ausgestattet sind, ist auch die Softwareplattform *Arduino IDE* (IDE = Integrated Development Environment) als Entwicklungsumgebung ein wichtiger Teil des Projekts. Beide Bestandteile, Hardware und Software, sind als Open Source verfügbar. Der Name *Arduino* ist allerdings als Marke eingetragen und geschützt.

Arduino-Boards
Originale Arduino-Boards werden in Italien hergestellt und weltweit vertrieben. Die Erfinder von Arduino arbeiten konstant an Verbesserungen bestehender Boards und an neuen Shields, die es ermöglichen, die Arduinos mit Peripherie zu erweitern.

Während der letzten zwei Jahre erlebte die Hardware-Branche mit Arduino etwas Unvorstellbares. Open-Source-Hardware, die jeder kopieren und vertreiben kann, bricht als Geschäftskonzept alle Regeln der Betriebswirtschaft. Mit einem Mix aus Markenstärke, durchdachtem Vertrieb und Weiterentwicklung der Arduino-Plattform schaffte es das Produkt in praktisch jedes Zimmer und jede Garage der passionierten Hobbytüftler. Wenn man heute einen Arduino kauft, dann kauft man Qualität sowie Kompatibilität, und dafür steht die Marke *Arduino*.

Für dieses Buch werden wir uns mit dem *Arduino Due* befassen. Er ist ein Entwicklerboard, das mit einem *Cortex-M3* als Mikrocontroller bestückt ist und Aus- und Eingänge für Peripherie besitzt, z. B. serielle Schnittstelle, SPI, D/A-Wandler, A/D-Wandler und CAN.

Die Kommunikation mit Android ist ohne Zukauf von Modulen über USB möglich. Dies kann beim Arduino Due auf zwei Arten geschehen:

1. Arduino als USB-Device
   Der Arduino wird über den USB-Seriell-Wandler (USB-Serial-Konverter) von Android angesprochen, dazu übernimmt Android die Rolle des USB-Hosts; je nach Gerät erfolgt dies mit einem USB-OTG-Konverter (USB On-The-Go). Gewisse Tablets verfügen über einen herkömmlichen USB-A-Port, wie er vom Desktop-PC bekannt ist. Der Arduino wird dabei vom Android-Gerät mit Strom versorgt und muss nicht extern gespeist werden.

2. Arduino als USB-Host
   Der Arduino übernimmt die Rolle des USB-Hosts und versorgt das Android-Gerät mit Energie. Hierzu wird auf der Seite von Android die *Android Accessory API* verwendet und der Arduino als sogenanntes »Accessory« betrieben.

Auf beide Varianten wird in den kommenden Teilen des Buchs näher eingegangen.

Der Arduino kann so die Schnittstelle zur »realen Welt« werden und externe Hardware ansteuern. Vom Toaster-Regulator bis zur Rasenmäher-Steuerung sollte so alles möglich sein.

Tipp
Auf der Webseite von Arduino werden regelmäßig die neuesten Erfindungen präsentiert: *http://arduino.cc/en/Tutorial/HomePage*. Suchen Sie hier z. B. unter dem Punkt *Examples*.

In Verbindung mit einem Arduino kann ein Android-Tablet oder ein Smartphone also in eine »Controller-Steuerung« verwandelt werden. Im Speziellen können dazu die Vorteile des Android-Betriebssystems ausgenutzt werden, z. B. Wi-Fi und Bluetooth oder Beschleunigungssensoren und Touchscreen.

Bluetooth und Wi-Fi
Natürlich könnte man auch den Arduino mit einen Bluetooth- oder Wi-Fi-Modul ausstatten. Allerdings ist es wesentlich einfacher, diese Bestandteile dem schon konfigurierten Smartphone zu übergeben und sich nicht um das Debuggen dieser Kommunikationsprotokolle auf niedriger Ebene kümmern zu müssen. Gerade wenn es darum geht, einen Netzwerk-Stack zu implementieren, greift man gerne auf Embedded-Systeme zurück, welche bereits Kernel-Treiber zur Verfügung haben.

Der *Universal Serial Bus* (*USB*) ist zentraler Bestandteil der Kommunikation zwischen Android und Arduino. In diesem Buch werden die zwei verschiedenen Varianten *Accessory API* und *USB Serial* genauer unter die Lupe genommen. Die Variante *USB Serial* ermöglicht übrigens auch die Kommunikation mit anderen Boards, die eine serielle Schnittstelle haben, wie z. B. dem *Arduino Uno*.

Seit Sommer 2011 und Android 3.1 (*Honeycomb*) ist es möglich, über das Android SDK via USB-Host mit Peripherie zu kommunizieren. Dies ermöglicht den Anschluss von allerlei Hardware an Android-Geräte. Dazu gehören z. B. Maus oder Tastatur, das kann aber auch bis hin zu USB-Spielzeug wie Mini-Raketenwerfern gehen. Von den Autoren wird beispielsweise das Open-Source-USB-Oszilloskop »OsciPrime« entwickelt, welches via USB-High-Speed Gebrauch von der neuen *USB-Host-API* macht. Damit wird das Android-Gerät in ein Oszilloskop inklusive Multitouch-Funktionalität verwandelt.

Viele Arduino-Boards besitzen einen USB-Serial-Konverter. Das ist ein Mikrocontroller, der die Aufgabe hat, USB in »klassische« serielle UART-Kommunikation zu verwandeln. Diese serielle Schnittstelle wird dazu verwendet, den Haupt-Mikrocontroller auf dem Arduino-Board zu programmieren, oder während der Laufzeit mit dem Haupt-Mikrocontroller zu kommunizieren. Die serielle Schnittstelle (RS 232) stellt eines der meistverwendeten Protokolle in der Welt der Mikrocontroller dar. Ihre Einfachheit und Universalität sind Kernfaktoren für die heutige Verbreitung. Da USB wesentlich komplexer ist und spezielle Hardware bei Mikrocontrollern voraussetzt, ist bei vielen Controllern auch heute noch die klassische serielle Schnittstelle anzutreffen.

USB hat einen *Host*, der Ursprung aller Kommunikation ist, und bis zu 127 *Devices*. Dabei diktiert der Host, welches Device gerade etwas senden darf. USB erlaubt den Bezug von maximal 500 mA und läuft bei einer Spannung von 5 V. Maximal werden bei USB-High-Speed Datenraten von 480 Mbit/s erreicht. Dies sind allerdings nur theoretische Werte – in der Praxis ist, je nach Host-Controller, nur etwa die Hälfte erreichbar.

USB-Devices haben eine hierarchische Struktur. So besitzt ein USB-Device verschiedene »Configurations«, welche wiederum verschiedene »Interfaces« zur Verfügung stellen können. Jedes Interface enthält eine gewisse Anzahl »Endpoints«, die jeweils die Richtung »in« (vom Device zum Host) oder »out« (vom Host zum Device) vorschreiben. Die Richtungen *in* und *out* sind also mit Bezug auf den USB-Host definiert. Ausnahme dabei ist der Endpoint »0«, welcher bidirektional betrieben werden kann. Den Endpoints ist jeweils eine Adresse zugeordnet, welche vom Host zum Absenden oder Empfangen von Daten verwendet wird. So kann der Host einen Endpoint eines Device adressieren, welcher dem Interface der aktuellen Konfiguration zugewiesen ist.

Der USB-Standard definiert Geräteklassen. Darunter fallen z. B. Audio, Drucker oder Human Interface Device (HID; wie Maus und Tastatur). Eine Geräteklasse ist auch die »serielle« CDC-Klasse, welche eine virtuelle serielle Schnittstelle emuliert. Das USB-Device kommuniziert jeweils die Geräteklasse, und der USB-Host lädt daraufhin den entsprechenden Treiber. So wird zum Beispiel beim Anschließen einer Maus der passende Maustreiber geladen. Das USB-Device kommuniziert daher auch, um welchen »Typen« es sich handelt, mittels der Parameter »Vendor ID« (VID) und »Product ID« (PID). Für unsere Versuche filtern wir die VID/PID vom USB-Serial-Konverter des Arduinos, um von Android her kommunizieren zu können.

Tipp
»USB in a Nutshell« (*http://www.beyondlogic.org/usbnutshell/usb1.shtml*) bietet eine detaillierte Übersicht des USB-Standards. Hier werden u. a. alle vorhandenen »Geräte-Profile« besprochen.

Ein weiteres wichtiges Feature von USB ist *USB On-The-Go* (OTG). Falls vom Betriebssystem unterstützt, ermöglicht OTG es, ein USB-Device in einen USB-Host

umzuwandeln. So ist es auch möglich, bei gewissen Android-Geräten eine Maus oder Tastatur anzuschließen.

Eine Standard-Typ-A-USB-Buchse wie bei einem Computer gibt es bei Android-Geräten äußerst selten. Deswegen muss für das Anschließen einer Maus oder ähnlicher Geräte in den meisten Fällen ein sogenannter USB-OTG-Adapter verwendet werden. Mit USB OTG ist es auch möglich, externe USB-Devices mit Energie zu versorgen. USB OTG wird allerdings nicht von allen Geräten automatisch unterstützt.

Zwar ist USB OTG hardwaremäßig bei fast allen verbauten Mikrocontrollern für Smartphones und Tablets vorhanden, aber es hängt allein vom Hardware-Vertreiber ab, ob die Software USB OTG unterstützt. Besitzt man ein Android-Gerät, das USB OTG nicht unterstützt, kann man dem immer noch mit einem »Custom ROM« entgegenwirken. Speziell das *CyanogenMod*-Projekt (*http://www.cyanogenmod.org/*) ist bestrebt, dem Benutzer eine Alternative zur vorinstallierten »Interpretation« von Android zu geben.

USB OTG
Galaxy Nexus, Motorola Xoom, Nexus 7 und Galaxy S3 unterstützen USB OTG und auch die Android-USB-Host-API. Zum jetzigen Zeitpunkt sieht es so aus, als würde das Nexus 4 USB OTG nicht unterstützen. Dies kann sich jedoch mit einem Softwareupdate noch ändern – allerdings scheinen weder der Hersteller noch Google daran interessiert zu sein, da kein Statement zur Abwesenheit dieses Features vorhanden ist. Das Projekt *http://usbhost.chainfire.eu/* versucht mit »user-reports«, USB-Host-Support experimentell ausfindig zu machen. Auf der Webseite kann man also nachschauen, ob ein Gerät USB OTG bzw. USB-Host unterstützt.

Dank des Android-ADK (*Accessory Development Kit*) ist es allerdings möglich, mit Android über USB zu kommunizieren, ohne einen USB-Host-Mode vorauszusetzen. Alle Android-Geräte, die einen »Play Store« haben, sind dazu verpflichtet, einen USB-Device-Port zu besitzen. Darüber findet nicht nur Debugging von Apps und Datenaustausch zwischen PC und Android-Gerät statt, sondern auch Kommunikation zu Android-»Accessories«.

Der *Arduino Due* kann dank modernem Mikrocontroller einerseits die Rolle des USB-Hosts übernehmen (mit einem USB-OTG-Adapter) und andererseits auch für ein Android-Gerät mit USB-Host-Support als »serielles« Device verwendet werden. Wird der Arduino an einen USB-Host angeschlossen, so wird er mit Strom versorgt. Unser Android-Tablet oder Android-Smartphone kann also auch als portable Batterie für den Arduino betrachtet werden. Da mit dem *Arduino Due* beide Varianten (USB-Host und USB-Device) möglich sind, werden wir uns in den Programmierbeispielen jeweils mit zwei Lösungsansätzen beschäftigen.

4 Android und USB

Kernstück für die Experimente mit Android und Arduino ist die USB-Schnittstelle zwischen Android-Device und Arduino Due. Android verfügt über zwei Application Programming Interfaces (APIs), die es erlauben, über USB mit Peripherie zu kommunizieren.

Das Android-Device übernimmt dabei die Rolle des USB-Hosts und versorgt die Peripherie mit Strom. So ist es z. B. möglich, eine USB-Maus an ein Android-Gerät anzuschließen. Je nach Gerät ist hierzu allerdings ein USB-OTG-Konverter nötig. Android liefert dem Entwickler mit der USB-Host-API eine Möglichkeit, selber einen USB-Treiber für die angeschlossene Peripherie zu schreiben. Der Arduino verfügt über eine USB-Serial-Schnittstelle und kann daher via Android-USB-Host-API mit Android kommunizieren.

Die Android-Accessory-API bezweckt genau das Gegenteil. Die »Peripherie« agiert als USB-Host und versorgt das Android-Device mit Strom. Dabei wird der »native« USB-Port des Arduino Due mit einem USB-OTG-Konverter ausgestattet und das Android-Gerät angeschlossen.

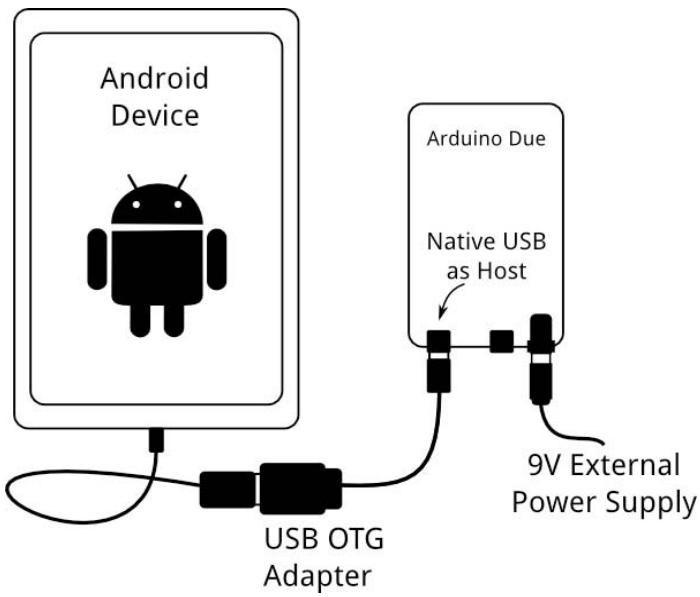


Bild 4.1a: Die USB-Kommunikation mit dem Arduino als Host.

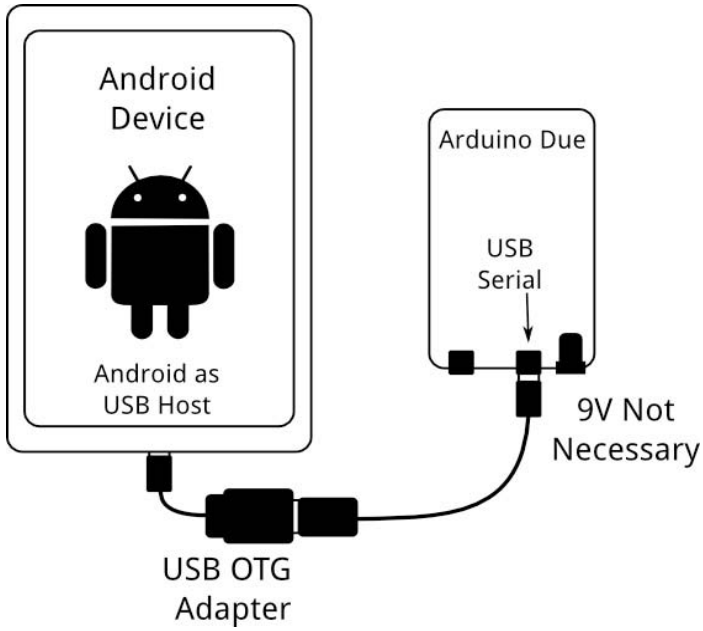


Bild 4.2b: Die Variante mit dem Android-Gerät als Host.

Lesezeichen
Auf der Entwicklerseite von Android findet man Dokumentation zu beiden APIs.
Android ADK:
http://developer.android.com/guide/topics/connectivity/usb/accessory.html
Android USB Host API:
http://developer.android.com/guide/topics/connectivity/usb/host.html

In den folgenden Abschnitten wird die Kommunikation zwischen Android und Arduino Due behandelt. Dazu wird das »Hello World«-Beispiel als Basis für beide Kommunikationsvarianten (Android-USB-Host-API und Android-Accessory-API) verwendet.

4.1 Kommunikation: Arduino als Accessory

Der Arduino soll als Android-»Accessory« betrieben werden. Dabei verwenden wir die Android-Accessory-API. Dazu teilen wir unserem Android-Manifest mit, dass wir eine Verbindung mit unserem Arduino erwarten. Sobald der Arduino verbunden ist, findet das Android-Betriebssystem Applikationen, welche auf das Anschließen eines Accessorys warten. Stimmen Modellname, Hersteller und Version des Accessorys mit dem im

Manifest erstellten »Filter« überein, kann so direkt eine Android-App gestartet werden. Zusätzlich wird der Activity vom System eine Schnittstelle zur Hardware via Argument übergeben. So kann die Kommunikation beginnen.

Die folgende Vorgehensweise kann auch in der API-Dokumentation auf *http://developer.android.com* gefunden werden.

Kommunikation über die Accessory-API
Es gibt zwei verschiedene Möglichkeiten, die Kommunikation via Accessory-API aufzunehmen. Unterschiedlich ist, wie die »Berechtigung« zur Kommunikation mit dem Accessory gehandhabt wird.

- Via Manifest-Eintrag:
Sobald der Arduino angeschlossen wird, erscheint eine Meldung vom System, die der Benutzer verwenden kann, um die Activity zu starten oder den Vorgang abzubrechen. Wird die Activity gestartet, so ist die Berechtigung bereits erteilt worden.

- Via Enumeration:
Während der Laufzeit einer App kann der Entwickler die vorhandenen ADK-Geräte vom System auflisten lassen. Bevor allerdings eine Kommunikation beginnen kann, muss die App den Benutzer nach der Berechtigung fragen.

Im Folgenden werden wir uns inhaltlich einfachheitshalber auf die erste Variante via Manifest-Eintrag beschränken.

Um einen Filter für unser Accessory zu erstellen, benötigen wir eine neue Datei *res/xml/accessory_filter.xml*. Der Ordner *xml* unter dem *res*-Verzeichnis ist standardmäßig nicht vorhanden und muss erst erstellt werden.

Den Inhalt des XML-Files schreiben wir folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<resource>
    <usb-accessory model="HelloWorldModel"
                   manufacturer="Hello Inc" />
</resource>
```

Diese Werte müssen mit den Definitionen im Arduino-Sketch übereinstimmen, damit das System erfolgreich die zugehörige Activity finden kann.

Als Nächstes referenzieren wir diese Meta-Daten in der Manifest-Datei von Android (*AndroidManifest.xml*). Hierzu wird ein neues *IntentFilter* für unsere HelloWorld-Activity erstellt.

```
<intent-filter>
    <action android:name=
     "android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
</intent-filter>
```

Das obige *IntentFilter* wird der Activity hinzugefügt.

Direkt anschließend referenzieren wir die Meta-Informationen:

```
<meta-data android:name=
"android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
android:resource="@xml/accessory_filter"/>
```

Was wir noch zusätzlich ändern, ist die mindestens benötigte SDK-Version (`minSdkVersion`). Diese setzen wir auf 12 (Android 3.1). Weiter teilen wir dem System mit, dass wir das »accessory hardware feature« verwenden möchten. Dies stellt sicher, dass die App nur auf Geräten installiert werden kann, welche dieses Feature auch unterstützen.

Die komplette *AndroidManifest.xml* sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.helloworld"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="12"
    android:targetSdkVersion="17" />

  <uses-feature
  android:name="android.hardware.usb.accessory"/>

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.helloworld.MainActivity"
```

```
      android:label="@string/app_name" >
      <intent-filter>
        <action
        android:name="android.intent.action.MAIN" />
        <category
        android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

      <intent-filter >
        <action
        android:name="android.hardware.usb.action.
          USB_ACCESSORY_ATTACHED" />
      </intent-filter>
      <meta-data
      android:name="android.hardware.usb.action.
        USB_ACCESSORY_ATTACHED"
      android:resource="@xml/accessory_filter"/>
    </activity>
  </application>
</manifest>
```

Wird nun ein Arduino angeschlossen, auf dem ein Android-Accessory-Sketch läuft, so erscheint ein Dialog, der den User darauf aufmerksam macht, dass eine App mit dem angeschlossenen Accessory kommunizieren möchte.

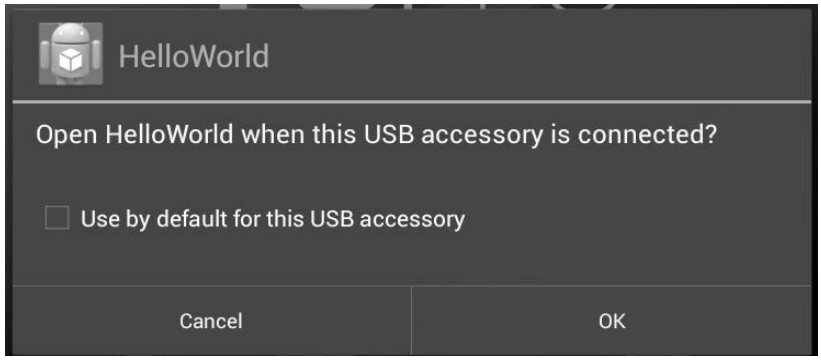


Bild 4.3: Die Dialogmeldung bei erfolgreich angeschlossenem Accessory.

Wird keine entsprechende Activity zum Accessory gefunden, so erscheint eine Dialogbox bei Anschließen des Arduino-Boards, die auf die in dem Arduino hinterlegte URL verweist.

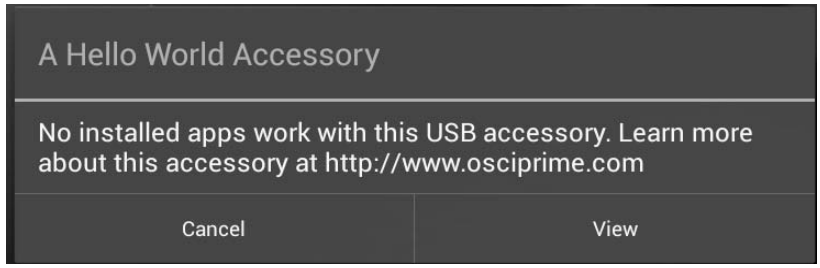


Bild 4.4: Diese Dialogbox erscheint, wenn keine Activity zum Accessory passt.

Beim Anschließen des Arduino Due wird ein »Extra« im Intent mitgeliefert, nämlich eine Referenz auf das Accessory. Mit dem Android-*UsbManager* erhalten wir so Zugriff auf das Accessory, um mit der Kommunikation beginnen zu können.

Die Android-Accessory-API nimmt uns sehr viel Arbeit ab. Tatsächlich ist das Accessory-USB-Protokoll eine komplexe Schnittstelle, welche Timing und Know-How von der Hardware benötigt. Die API nimmt uns die Initialisierung und die detailierte Kommunikation ab und ermöglicht das »Sprechen« mit dem Accessory über zwei simple Objekte. Dazu erhalten wir einen »FileDescriptor», in dem wir lesen und schreiben können.

```
FileDescriptor fd =
  mAccessoryFileDescriptor.getFileDescriptor();
FileInputStream fis = new FileInputStream(fd);
FileOutputStream fos = new FileOutputStream(fd);

...

fis.read(...); //receive something from the accessory
fos.write(...); //send something to the accessory
```

So erleichtert sich die Kommunikation mit dem Arduino erheblich und es wird ein rasches Implementieren der gewünschten Funktionalität ermöglicht.

Für unser nachfolgendes erstes Experiment »Hello World mit Android Accessory« senden wir dem Accessory ein »Byte« und regeln damit die Helligkeit der On-board-LED.

Beim Starten der Activity finden wir zuerst heraus, ob die Activity über den Launcher oder über das Anschließen des Accessorys gestartet wurde. Im letzteren Fall wird der Activity über den Intent, welcher die App startet, das Accessory als Argument mitgeliefert. In `onCreate` kann so direkt eine Referenz auf das Accessory-Objekt geholt werden.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
```

```
  setContentView(R.layout.activity_main);

  //see if we are started by ADK
  if(getIntent().hasExtra(UsbManager.EXTRA_ACCESSORY)){
    UsbAccessory accessory = (UsbAccessory)getIntent()
    .getParcelableExtra(
      UsbManager.EXTRA_ACCESSORY
    );
  }
  ...
}
```

Wird die Activity via Accessory-System-Dialog gestartet, so ist die Berechtigung auf das Device automatisch gewährt.

In der App wird nun eine Referenz auf das Accessory gespeichert, sodass beim Klick auf den Button eine Nachricht (Byte) gesendet wird. In unserem »HelloWorld« existiert bereits ein zweiter Thread, den wir etwas abändern.

Dem Klassenrumpf fügen wir eine Referenz auf das Accessory hinzu (`mAccessory`).

```
private UsbAccessory mAccessory = null;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Log.d(TAG, "onCreate in HelloWorld!");
  if(getIntent().hasExtra(UsbManager.EXTRA_ACCESSORY)){
    mAccessory = (UsbAccessory)getIntent()
      .getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
  }
  Button btStart = (Button) findViewById(R.id.btStart);
  btStart.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
      Log.d(TAG, "click!");
      if(sTaskThread == null){
        sTaskThread = new Thread(mTaskRunnable);
        sTaskThread.start();
      }
    }
  });
}
```

Friedrich und Andreas Plötzeneder

Powerprojekte mit Arduino™ und C

Schrittmotor
Distanzmessung mit Ultraschall
Transistorkennlinie
EKG und schwebende Kugel

Inhaltsverzeichnis

6 Programmierung der seriellen Schnittstelle des AVR

In diesem Kapitel werden vier Methoden gezeigt, nach denen man die serielle Schnittstelle programmiert. Zuerst wird die Programmierung mit der Entwicklungsumgebung von CodeVisionAVR gezeigt. Der Vorteil besteht darin, dass man sich weder bei der Initialisierung noch bei der Ein- und Ausgabe um Register kümmern muss. Man kann sogar die serielle Schnittstelle mit *printf()* und *scanf()* ansprechen, ohne das Datenblatt zu lesen.

Bei der nächsten Methode wird die serielle Schnittstelle im Prozessor auf einfache Weise angesprochen. Die notwendigen Programmteile sind in der Datei *serial1.h* gespeichert. Mithilfe von *#include* kann das Programm *serial1.h* leicht in ein eigenes Programm eingebaut werden. Eine weitere Methode, die im Wesentlichen in der Datei *serial2.h* programmiert ist, erlaubt es, im AVR Studio die Befehle *printf()* und *scanf()* zu verwenden. Die Standardeingabe wird damit auf die serielle Schnittstelle umgelenkt, wodurch eine formatierte Ein- und Ausgabe möglich ist.

Bei der vierten Methode geht es um die verbreitete Bibliothek von Peter Fleury. Mit dieser Bibliothek kann man auch im Interrupt, also im Hintergrund, kommunizieren. Der Link ist angegeben.

6.1 Programmierung mit CodeVisionAVR

Mit dem CodeVisionAVR Compiler ist die Programmierung der seriellen Schnittstelle besonders einfach. Der besondere Vorteil kommt vom Wizzard, der die Konfiguration erleichtert, und der Bibliothek zum Lesen und Schreiben auf die RS-232-Schnittstelle. Mit der Bibliothek stehen Funktionen wie *printf()*, *scanf()*, *getchar()* … zur Verfügung, die elegant auf die serielle Schnittstelle zugreifen.

Das folgende Programm wurde für Arduino Uno mit ATmega8 erstellt. Es wurde mit ISP in den Flash des Prozessors übertragen und erfolgreich getestet. Sollten Sie mit einem Arduino Uno arbeiten, ist nur im Wizzard der Prozessor ATmega328P anzugeben. Die Hex-Datei kann dann in den Prozessor geladen werden.

Abb. 6.1: Neues Projekt anlegen

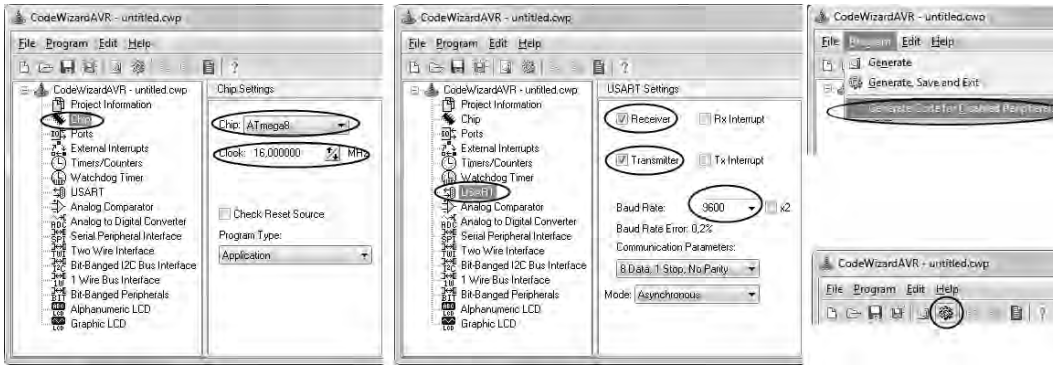


Abb. 6.2: Auswahl des Prozessors und der Clock-Frequenz; Konfiguration der seriellen Schnittstelle; danach erfolgen die Konfiguration des Wizzards und das Erstellen der C-Datei (rechtes Bild, Symbolleiste, markierter Stern).

Durch die Konfiguration des Wizzards mit *Generate Code for Disabled Peripherals* wird vom Wizzard ein kürzerer Code erzeugt.

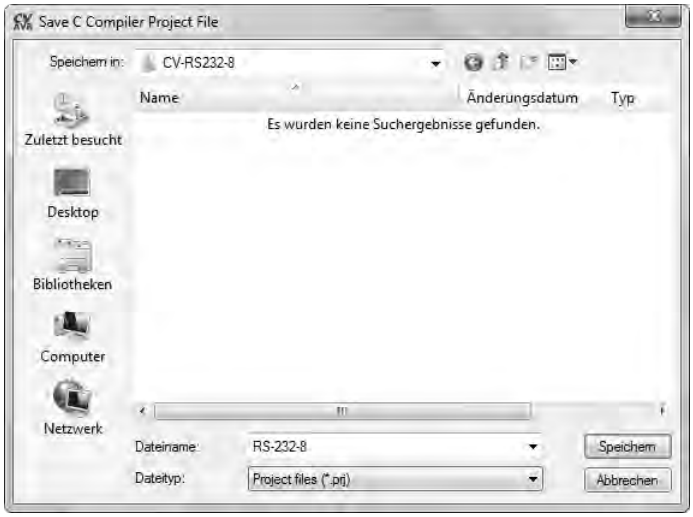


Abb. 6.3: Speichern der Projektdateien in einen Ordner; es sind drei Dateinamen zu vergeben.

Danach erscheint ein C-Programm, das alle Initialisierungen für die serielle Schnittstelle enthält. In dieses Programm sind folgende sechs Zeilen einzufügen:

```
char z; //einfuegen

z = getchar();          //einfuegen
if(z == 'a')            //einfuegen
  printf(" Anton\r\n"); //einfuegen
else if(z == 'b')       //einfuegen
  printf(" Berta\r\n"); //einfuegen
```

Dabei wird mit *char z;* ein Speicherplatz für ein Zeichen angelegt. Mit *z = getchar();* wird ein Zeichen von der seriellen Schnittstelle eingelesen und *printf(" Anton\r\n");* schreibt auf die serielle Schnittstelle.

```
/*****************************************************
This program was created by the
CodeWizardAVR V2.60 Evaluation
Automatic Program Generator
© Copyright 1998-2012 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 28.08.2012
Author  :
Company :
Comments:

Chip type               : ATmega8
Program type            : Application
AVR Core Clock frequency: 16,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

#include <mega8.h>

// Declare your global variables here

// Standard Input/Output functions
#include <stdio.h>

void main(void)
{
// Declare your local variables here
char z;                                          //einfuegen
// Input/Output Ports initialization
// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) |
(0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
```

```
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) |
(0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) |
(0<<DDC0);
// State: Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) |
(0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) |
(0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) |
(0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) | (0<<UPE) |
(0<<U2X) | (0<<MPCM);
UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) | (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2)
| (0<<RXB8) | (0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) | (0<<USBS) | (1<<UCSZ1)
| (1<<UCSZ0) | (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x67;

while (1)
      {    z = getchar();      //einfuegen
      if(z == 'a')             //einfuegen
         printf(" Anton\r\n"); //einfuegen
      else if(z == 'b')        //einfuegen
         printf(" Berta\r\n"); //einfuegen
      }
}
```

Das Programm wurde mit Arduino-Hardware und ATmega8/ISP funktionell geprüft. Bei Verwendung des ATmega328P ist dieser lediglich im Wizzard auszuwählen.

Mit diesem Programm kann man über ein Terminal per Tastendruck den Buchstaben a oder b senden, um danach Anton oder Berta zu empfangen, das vom Arduino abgegeben wird. Welches Terminal-Programm man verwendet, ist bei dieser Anwendung unbedeutend. Im Beispiel wird mit dem Terminal-Programm der IDE von CodeVisionAVR gearbeitet. Dafür muss der COM-Port für das Programmiergerät und die virtuelle serielle Schnittstelle gewählt werden. (Falls nicht bekannt ist, an welchem

Port das Programmiergerät und der Arduino angeschlossen sind, ist das in *Systemsteuerung/Gerätemanager/Anschlüsse* zu sehen.)

Beim Senden eines Zeichens vom Terminal ist es nicht erforderlich, ein zusätzliches LF an das zu sendende Zeichen zu hängen (Settings/Terminal; nicht *Append LF on Transmission* setzen). Das ist für die einwandfreie Funktion von *getchar()* notwendig.

Abb. 6.4: Auswahl der COM-Schnittstellen; bei *Terminal* ist die Baudrate einzustellen. Die Einstellung Settings • Programmer ist nur nötig, wenn man den ATmega8 verwendet und ihn mit einem ISP von CodeVison programmieren will.

Abb. 6.5: Nach Start des Terminals in der Symbolleiste und Verbinden (Connect) kann man mit dem Mikroprozessor kommunizieren.

Der CodeWizzard hat bei der Initialisierung der seriellen Schnittstelle fünf Register beschrieben. Das sind:

```
UCSRA=(0<<RXC) | (0<<TXC) | (0<<UDRE) | (0<<FE) | (0<<DOR) |
      (0<<UPE) | (0<<U2X) | (0<<MPCM);
UCSRB=(0<<RXCIE) | (0<<TXCIE) | (0<<UDRIE) |
      (1<<RXEN) | (1<<TXEN) | (0<<UCSZ2) | (0<<RXB8) | (0<<TXB8);
UCSRC=(1<<URSEL) | (0<<UMSEL) | (0<<UPM1) | (0<<UPM0) |
      (0<<USBS) | (1<<UCSZ1) | (1<<UCSZ0) | (0<<UCPOL);
UBRRH=0x00;
UBRRL=0x67;
```

Falls Sie in Zukunft nur noch den CodeVisionAVR Compiler verwenden, können Sie Initialisierungswerte, die vom Wizzard kommen, als gegeben hinnehmen. Sollten Sie aber mit dem Atmel Studio arbeiten, ist es notwendig, die Initialisierungswerte explizit anzugeben. Daher werden die Initialisierungen in den Registern UCSRA, UCSRB, UCSRC, UBRRH, UBRRL besprochen.

Zuerst wird für die Baudrate das Teilungsverhältnis berechnet. Gegeben: Oszillatorfrequenz = 16 MHz, Baudrate = 9.600 Bit/sec. Das Teilungsverhältnis, mit dem die Register UBRRH und UBRRL zu laden sind, berechnet sich mit der in Abb. 6.6 gezeigten Formel.

$$\text{Teilungsverhältnis} = \frac{\text{Oszillatorfrequenz}}{\text{Baudrate} \cdot 16} - 1 = \frac{16000000}{9600 \cdot 16} - 1$$

Abb. 6.6: Berechnung des Teilungsverhältnisses für die serielle Schnittstelle

Als Zahlenwert für das Teilungsverhältnis erhält man 103,166 oder ganzzahlig und in hexadezimal 67_H.

Der ganzzahlige Wert weicht nur 0,16 % vom berechneten Wert ab. Damit ist die tatsächliche Baudrate 9.615 Baud, statt der gewünschten 9.600 Baud. Es fragt sich, ob die Abweichung der Baudrate von 0,16 % akzeptabel ist. Dazu stellen wir eine einfache Überlegung an: Ein Byte, das auf der Schnittstelle übertragen wird, hat zusätzlich ein Start-Bit und ein Stopp-Bit und damit 10 Bits. Synchronisiert wird auf die steigende Flanke des Start-Bits und danach wird, zeitlich betrachtet, in der Mitte eines Bits abgetastet. Eine zeitliche Abweichung von einem halben Bit in der Abtastung führt zu einem Fehler. Ein halbes Bit Verschiebung bei 10 Bits bedeutet 5 % Fehler. Billigt man nun Sender und Empfänger den gleichen Fehler zu und rechnet 1 % Sicherheit dazu, kann man einen Baudratenfehler von 2 % tolerieren.

$$\frac{16000000}{9600 \cdot 16} - 1 = 67_{\mathrm{H}}$$

Abb. 6.7: Teilungsverhältnis bei 16 MHz Quarz und 9.600 Baud

Der Faktor 16 im Nenner der Formel ist durch den Aufbau des UART begründet. Der Wert –1 wird anhand eines Beispiels erläutert. Falls das Teilungsverhältnis auf 2 gestellt wird, zählt der Teiler 0, 1, 2, 0, 1, 2, 0 …, also mit einer Periodizität von 3. Daher ist als Teilungsverhältnis um eins weniger einzustellen, als das gewünschte Frequenzteilungsverhältnis ist.

Daraus folgt, dass bei der Initialisierung der seriellen Schnittstelle UBRRH=0x00 und UBRRL=0x67 zu setzen ist.

$$\mathrm{Baudrate} = \frac{\mathrm{Quarzfrequenz}}{(\mathrm{Teilungsverhältnis} + 1) \cdot 16}$$

Abb. 6.8: Berechnung der Baudrate aus Teilungsverhältnis und Quarzfrequenz

Tabelle 6.1: UCSRA im Programm oben mit 0x0 initialisiert; dieses Byte wird bei der Initialisierung auf 0 gesetzt und muss nicht verändert werden.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

RXC (Receive Complete) und TXC (Transmit Complete)

RCX zeigt an, dass ein Byte empfangen worden ist und von UDR ausgelesen werden kann. TCX gibt an, dass ein Zeichen vollständig übertragen wurde.

Der Status der Ausgabe an der RS-232-Schnittstelle kann an zwei Bits erkannt werden. Die entsprechenden Bits sind im *UCSRA Data Register Empty* (UDRE) und *Transmit Complete* (TXC). Das UDRE zeigt an, dass der Sendepuffer neue Daten übernehmen kann. Ist das UDRE Bit 1, bedeutet das, dass der Sendepuffer leer ist. Mit einer Zuweisung *UDR = (char)data;* kann man in diesem Fall ein Zeichen an die RS-232 ausgeben.

Bis das Zeichen vollständig ausgegeben ist, wird das UDRE auf 0 gesetzt. Nachdem das Zeichen vollständig ausgegeben ist, setzt der UART im Controller das UDRE auf 1.

Tabelle 6.2: UCSRB im Programm oben mit 0x18 initialisiert

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

RXEN und TXEN stehen für Receive Enable und Transmit enable. Diese Bits sind auf 1 zu setzen, um die serielle Schnittstelle zu aktivieren.

Tabelle 6.3: UCSRC im Programm oben mit 0x86 initialisiert

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

Das Bit URSEL (USART Register Select) muss beim Beschreiben des Registers URSRC immer gesetzt werden. Der Grund dafür ist, dass das UCSRC und das UBRRH dieselbe Adresse haben. Das oberste Bit entscheidet, ob in das URSRC oder UBRRH geschrieben wird.

Die Bits UCSZ0 und UCSZ1 muss man im Zusammenhang mit dem UCSZ2 im UCSRB sehen. Die drei Bits sind auf 011_B gesetzt und ergeben die Zahl 3_D. Um die Anzahl der Datenbits bei der Übertragung der seriellen Schnittstelle zu konfigurieren, erfolgt folgende Berechnung: UCSZ2, UCSZ1, und UCSZ0 + 5_D = 8_D. Es wird dadurch mit acht Daten-Bits bei der Übertragung gearbeitet.

6.2 Programmierung im Atmel Studio

In diesem Beispiel wird mit dem ATmega8 oder dem ATmega328P (der Prozessor des Arduino UNO) über die serielle Schnittstelle kommuniziert. Bei den Prozessoren ATmega8 und ATmega328P sind die Register, die für die serielle Schnittstelle relevant sind, auf unterschiedlichen Adressen. Damit ist auch erklärt, warum ein Programm, das für den ATmega8 geschrieben worden ist, nicht auf den ATmega328P läuft. Die Register, die für die serielle Schnittstelle relevant sind, haben nicht nur verschiedene Registeradressen, sondern werden auch unterschiedlich bezeichnet.

Tabelle 6.4: Register zur Initialisierung der seriellen Schnittstelle beim ATmega8 und ATmega328P

Register ATmega8	*Register ATmega328P*
UCSRA	UCSR0A
UCSRB	UCSR0B
UCSRC	UCSR0C
UBRRH	UBRR0H
UBRRL	UBRR0L

Zusätzlich sind auch einzelne Bits unterschiedlich bezeichnet (siehe *iom386p.h, iom8.h*).

Tabelle 6.5: Bezeichnung der Bits zur Konfiguration der seriellen Schnittstelle beim ATmega8 und ATmega328P

Bits ATmega8	*Bits ATmega328P*
TXEN	TXEN0
RCEN	RXEN0
UCSZ0	UCSZ00
UCSZ1	UCSZ01
UMSEL1	UMSEL01
RXC	RXC0

Der nächste Schritt ist, ein Programm zu schreiben, das für verschiedene Prozessoren die richtigen Register verwendet. Das erfolgt mit einer bedingten Übersetzung, die der Präprozessor durchführt. Der Präprozessor ist das erste Programm, das die Übersetzung des Quellcodes durchführt. Eine Aufgabe des Präprozessors ist die bedingte Übersetzung, die an einem Beispiel erläutert wird. Im Programm unten wird abgefragt, ob UCSR0A schon definiert ist. Falls das der Fall ist, wird Code1 eingesetzt, andernfalls Code2. Das darf nicht mit einem *if-else* in einem Programm verwechselt werden. Bei einem *if* im Programm führt der Prozessor (AVR) die Entscheidung aus. Ein *#if* hingegen ist eine Information an den Compiler, der damit gesteuert wird, nur einen bestimmten Code zu berücksichtigen.

```
#ifdef UCSR0A
// Code1
#else
//Code2
#endif
```

Dafür werden die notwendigen Funktionen in die Datei *serial1.h* geschrieben und in das Hauptprogramm eingefügt. Das Einfügen ins Hauptprogramm erfolgt in drei Schritten:

1. Das Programm *serial1.h* ist in den Projektordner zu kopieren, in dem das Hauptprogramm ist (*serial1-test.c*).
2. Im Hauptprogramm ist vor *main()* mit der Zeile *#include "serial1.h"* die Datei *serial1.h* einzufügen.
3. Im Projektordner ist *serial1.h* aufzunehmen (siehe Abb.).

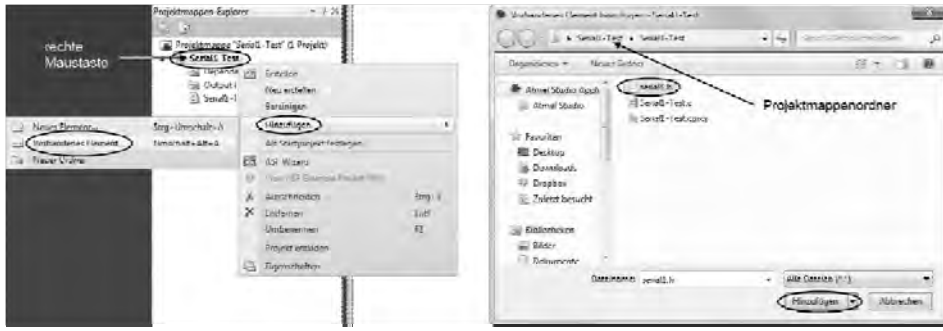


Abb. 6.9: Include-Datei *serial1.h* in das Projekt aufnehmen

Alternativ könnte an dieser Stelle auch der Inhalt von *serial1.h* in Textform ins Hauptprogramm eingefügt werden.

Das folgende Programm *serial2.h* besteht aus zwei Teilen. Die erste Hälfte des Programms ist beim Einsatz des ATmega328P relevant, die zweite Hälfte kommt beim Einsatz des ATmega8 zum Zug. Die Entscheidung könnte man von der gewählten CPU ableiten. Im Programm unten wird aber die Entscheidung vom Register UCSR0A abgeleitet, wodurch auch Prozessoren mit gleichen Registeradressen mit diesem Programm arbeiten können.

```
/*
 * serial1.h  fuer Atmel Studio
 *
 * Created: 01.08.2012 12:00:00
 *  Author: Friedrich Ploetzeneder
 */

#ifdef UCSR0A

//Initialisiert UART gegebener Oszillatorfrequenz und Baudrate
void uart_init(long Oszi, long Baud)
{
  UBRR0L = (unsigned char)(Oszi / 16L / Baud -1);
  UBRR0H = (unsigned char)((Oszi / 16L / Baud -1) >> 8);
  UCSR0A = 0x00;
  UCSR0B = UCSR0B | (1 << TXEN0) | (1 << RXEN0);
  UCSR0C = UCSR0C | (1 << UMSEL01) | (1 << UCSZ01) |
           (1 << UCSZ00);
}

//Empfaengt 1 Byte vom UART
unsigned char getchar2(void)
{
  while (!(UCSR0A & (1 << RXC0)));
  return UDR0;
}

//Sendet 1 Byte an UART
void putchar2(unsigned char data)
```

```
{
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = (char)data;
}

//Sendet String an UART
void puts2(char *str)
{
  unsigned char c;
  while ((c = *str++) != '\0')
    putchar2(c);
}

#else

//Initialisiert UART gegebener Oszillatorfrequenz und Baudrate
void uart_init(long Oszi, long Baud)
{
  UBRRL = (unsigned char)(Oszi / 16L / Baud -1);
  UBRRH = (unsigned char)((Oszi / 16L / Baud -1) >> 8);
  UCSRA = 0x00;
  UCSRB = UCSRB | (1 << TXEN) | (1 << RXEN); //RX TX Enable
  //Bit Mode
  UCSRC = UCSRC | (1 << URSEL) | (1 << UCSZ1) | (1 << UCSZ0);
}

//Empfaengt 8 Daten Bits vom UART
unsigned char getchar2(void)
{
  while (!(UCSRA & (1 < <RXC)));
  return UDR;
}

//Sendet 8 Daten Bits an UART
void putchar2(unsigned char data)
{
  while (!(UCSRA & (1 << UDRE)));
  UDR = (char)data;
}

//Sendet String an UART
void puts2(char *str)
{
  unsigned char c;
  while ((c = *str++) != '\0')
    putchar2(c);
}

#endif
```

Die Include-Datei *serial1.h.* enthält elementare Kommunikationsroutinen für die serielle Schnittstelle.

Im Hauptprogramm unten steht nur das Wesentliche zur Ein- und Ausgabe. Mit *#include "serial1.h"* wird der Inhalt der Datei *serial1.h* in das Hauptprogramm *serial1-test.c* eingesetzt und *serial1.h* in das Projekt aufgenommen. Danach erfolgt die Übersetzung des C-Programms mit dem Compiler. Jedenfalls wird durch die Verwendung von *#include* das Hauptprogramm übersichtlicher.

(Auch wenn es üblich ist, Zusatzpakete in Form von c- und h-Dateien aufzuspalten, ist die Lösung, dass nur eine Datei zusätzlich verwendet wird, einfacher und wird daher angewandt.) Eine Besonderheit ist, dass der Dateiname in Anführungszeichen geschrieben wird. Das bewirkt, dass die Include-Datei (*serial1.h*) im selben Ordner gesucht wird, in dem das Hautprogramm *serial1-test.c* steht.

```
/*
 * serial1-test.c

 *  fuer Atmel Studio

 * Created: 01.08.2012 12:00:00
 *  Author: Friedrich Ploetzeneder
 */

#define F_CPU 16000000UL
#define BAUD  9600UL

#include <avr/io.h>
#include "serial1.h"

int main(void)
{
  char z;

  //UART mit Clock-Frequenz und Baudrate initialisieren
  uart_init(F_CPU, BAUD);

  while (1)
  {
    z = getchar2();       //ein Zeichen einlesen
    if (z == 'a')
      puts2(" Anton\r\n"); //Zeichenkette ausgeben
    else if (z == 'b')
      puts2(" Berta\r\n");
  }
}
```

Hauptprogramm zur Ausgabe von Zeichen an die serielle Schnittstelle

Das Programm wurde mit Arduino-Hardware und ATmega8 (ISP) als auch mit ATmega328P (Bootloader) überprüft.

Auswahl der Prozessoren ATmega328P oder ATmega8 im Atmel Studio:

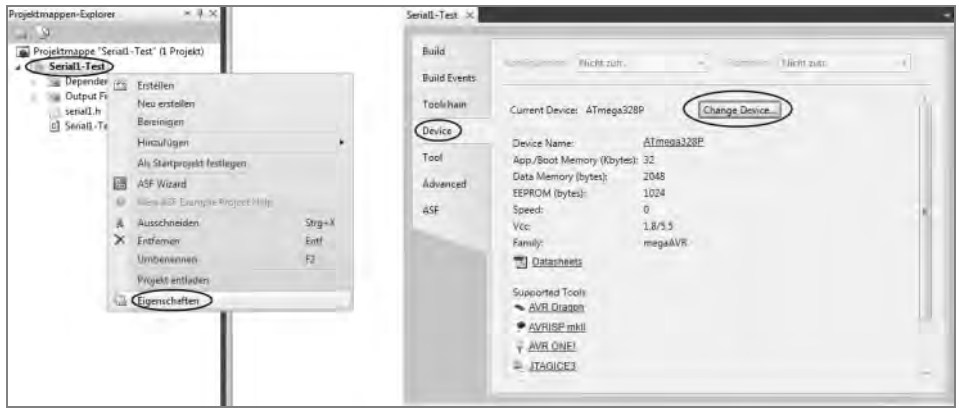


Abb. 6.10: Auswahl des Prozessors mit Projektmenü/Properties, Auswahl des Devices/Change Device

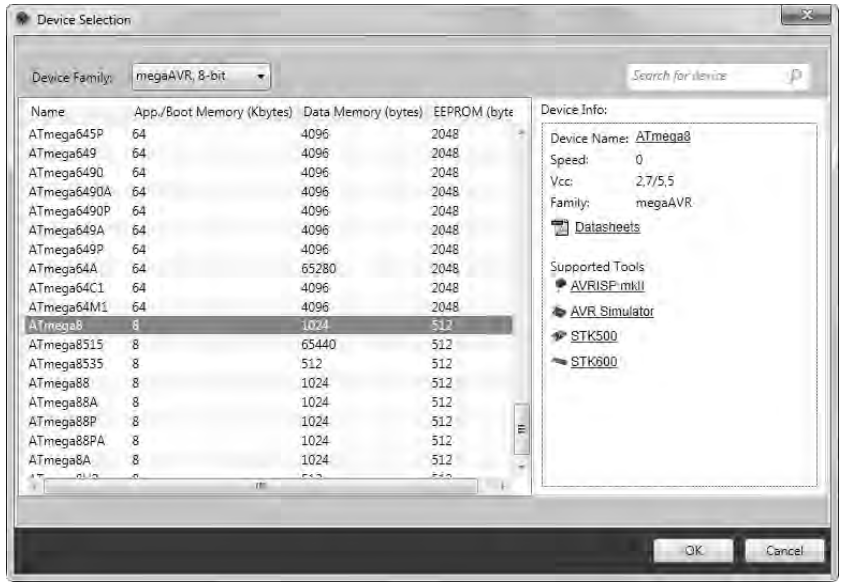


Abb. 6.11: Angabe des neuen Prozessors

Die Prozessorauswahl bewirkt, dass die richtigen Adressen für die Funktionsregister verwendet werden. Daher können bei einem Wechsel des Prozessors unterschiedliche HEX-Dateien erzeugt werden. Bei einfachen Aufgaben kann mit den Funktionen in *seriel1.h* gearbeitet werden. Manchmal wünscht man sich aber einen eleganteren Zugriff auf die serielle Schnittstelle, und das wird in der Folge vorgestellt.

Ulli Sommer

Arduino™

Mikrocontroller-Programmierung mit Arduino™/Freeduino

Über 70 Quellcodes zu den Experimenten
Open-Source-VB.NET-Programme zum Messen und Steuern

Inhaltsverzeichnis

1 Mikrocontroller-Grundlagen

Mikrocontroller werden vor allem im Bereich der Automatisierungs-, Mess-, Steuer- und Regeltechnik eingesetzt. Der Vorteil eines Mikrocontroller-Systems ist, dass es auf kleinstem Raum energie- und kosteneffizient physikalische Größen misst und interpretiert, darauf aufbauend Entscheidungen trifft und Aktionen durchführt. Im Grunde ist jede Aufgabe, die man mit Arduino lösen möchte, eine MSR(Messen, Steuern, Regeln)-Aufgabe.

1.1 Messen

Unter Messen versteht man im Allgemeinen, physikalische Eingangswerte von z. B. Schaltern, Lichtsensoren, Drucksensoren, Bewegungsmeldern, Lichtschranken, Spannungsteilern (Potenziometern) u. v. m. zu erfassen. Diese werden dem Controller über die digitalen oder analogen Eingänge zugeführt. Arduino versteht Signale mit einem Spannungspegel von 0 Volt (V) oder 5 V bei den digitalen Eingängen. Die analogen Eingänge hingegen können 0 V bis 5 V mit einer Auflösung von 10 Bit auswerten. Dabei entsprechen 0 V 0 und 5 V 1.023. Diese Werte können in der Software entsprechend interpretiert werden, um z. B. eine Mignonzelle auf ihre Spannung zu überprüfen. Für größere Spannungen an den Eingängen, egal ob digital oder analog, benötigen wir einen Spannungsteiler, der die Eingangsspannung auf die maximale Eingangsspannung der Arduino-Ports herunterteilt. Dazu später mehr.

1.2 Steuern

Unter Steuern versteht man, auf einen Eingangswert zu reagieren. Ein einfaches Beispiel ist ein Bügeleisen. Es steuert die Temperatur über einen Bimetallschalter. Ist die Bügelfläche kälter als am Bügeleisen eingestellt, wird so lange geheizt, bis die gewünschte Temperatur erreicht ist. Fällt die Temperatur unter einen bestimmten Wert, wird nachgeheizt, bis die Temperatur wieder stimmt. Zwischen den beiden Punkten »Heizung ein« und »Heizung aus« befindet sich ein kleines Fenster, eine sogenannte *Hysterese.* Sie sorgt dafür, dass nicht zu oft ein- bzw. ausgeschaltet wird, dadurch die mechanischen Kontakte nicht zu schnell verschleißen und die Steuerung nicht zu »nervös« reagiert. Bei einem Mikrocontroller könnte man das Bimetall durch einen Temperatursensor ersetzen. Der Mikrocontroller würde den Wert analog oder digital über einen passenden Temperatursensor erfassen, das Programm würde die Schwellenwerte für »ein« und »aus« vergleichen und einen digitalen Ausgang schalten, der wiederum ein Relais oder einen Transistor ansteuert, um die Heizung zu schalten.

1.3 Regeln

Der Unterschied zwischen Steuern und Regeln besteht darin, dass eine Steuerung nur bestimmte Ein/Aus-Zustände kennt. Eine Regelung hingegen ist stufenlos. Ein Tempomat im Auto ist z. B. eine Regelung, die versucht, immer die gespeicherte Geschwindigkeit zu halten. Wäre hier nur eine Steuerung verbaut, wäre die Fahrt mit Tempomat sehr unangenehm, da dieser nur Gasgeben, Nichtstun und Bremsen kennen würde.

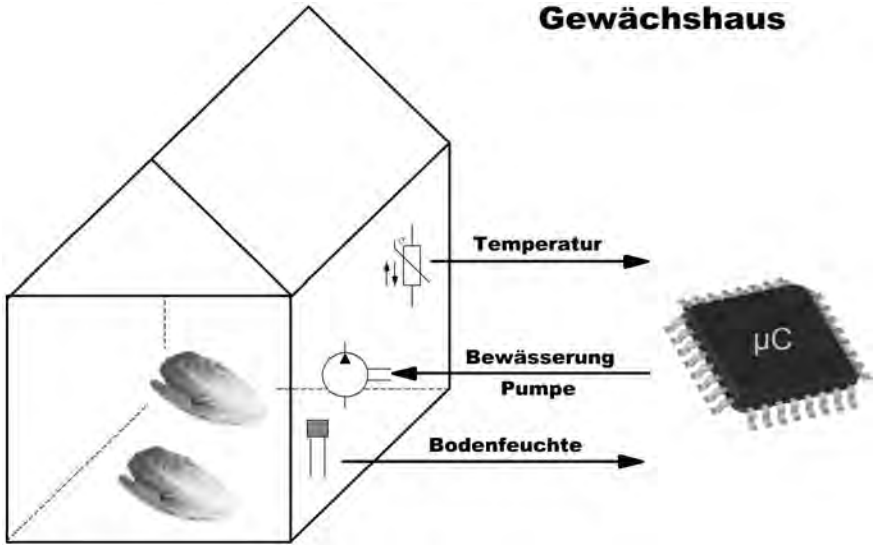


Bild 1.1: Beispiel einer Ein- und Ausgabeverarbeitung anhand eines Gewächshauses

Das Spektrum möglicher Anwendungen von Mikrocontrollern reicht vom privaten (z. B. der Steuerung eines Gewächshauses, des Aquariums oder der Hausbeleuchtung) bis zum industriellen Bereich, in dem komplette Produktionsanlagen mit Mikrocontroller-Systemen gesteuert, gewartet und betrieben werden können. Die Abbildung zeigt eine typische Datenverarbeitung zur Steuerung der Bewässerungsanlage eines Gewächshauses. Der Mikrocontroller nimmt dabei über Sensoren die Messwerte der Umgebungstemperatur und der Bodenfeuchte auf. Die Messwerte werden durch eine digitale Logik in Form eines Programms im Mikrocontroller (kurz: *µC* oder *MC*) interpretiert und es wird bestimmt, ob die Pumpe nun gießen soll oder nicht.

1.4 Aufbau und Funktionsweise

Als vollwertiger Computer im Kleinstformat weist jeder Mikrocontroller – ähnlich einem PC – grundlegende elektronische Bausteine auf. Grundbausteine eines jeden Mikrocontrollers sind die CPU, der Arbeitsspeicher (RAM) sowie der Programmspeicher (Flash) und die Peripherie.

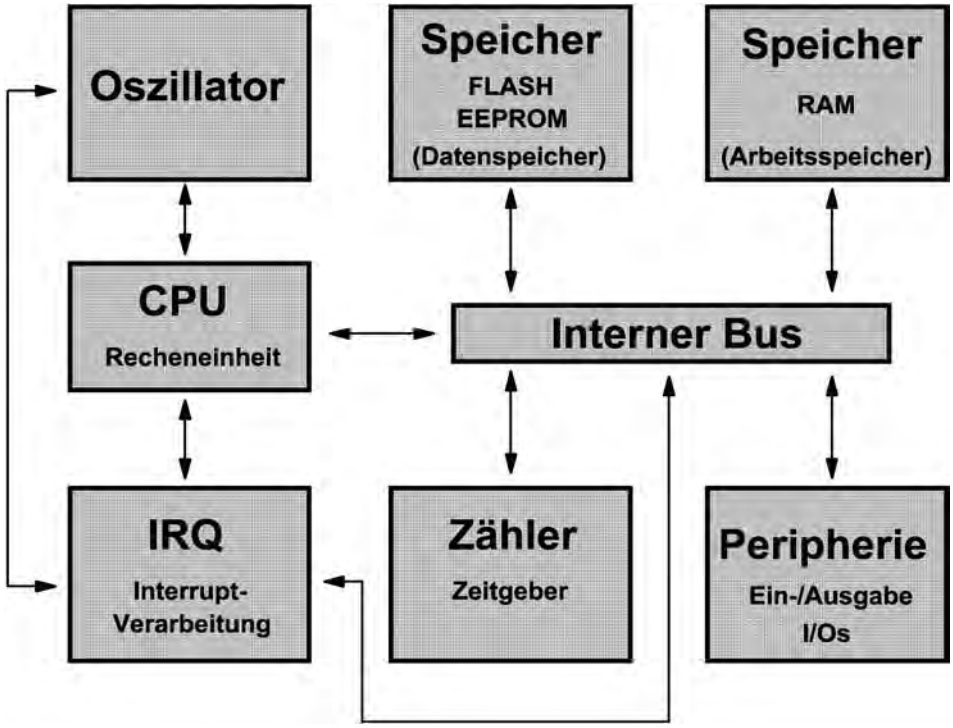


Bild 1.2: Prinzipieller Aufbau eines Mikrocontrollers

1.4.1 CPU

Die wichtigste Funktionseinheit ist die zentrale Recheneinheit, die CPU (engl.: Central Processing Unit). Sie kann als das »Gehirn« des Mikrocontrollers verstanden werden. Dort werden die Befehle und arithmetische Operationen abgearbeitet.

1.4.2 Arbeits- und Programmspeicher

Arbeits- und Programmspeicher sind logisch getrennt. Das Benutzerprogramm wird dabei in einem nichtflüchtigen Flash-Speicher abgelegt, dem Programmspeicher. Je nach Controllersystem kann man auf (implementierten) Programmspeicher von mehreren Kilo- bis Megabyte zurückgreifen. Bei einigen Systemen ist es darüber hinaus möglich, den Programmspeicher durch externe Flash-Komponenten aufzustocken. Der Arbeitsspeicher (RAM) dient zur temporären Ablage von Rechen-, Mess- und Steuergrößen. Ziel ist, möglichst schnell auf eine begrenzte Anzahl von Daten zugreifen zu können. Der RAM(Random Access Memory)-Speicher ist in der Regel deutlich kleiner als der Flash-Speicher, dafür aber um ein Vielfaches schneller. Die Werte des RAM werden zur Laufzeit erzeugt und sind, anders als beim Flash-Speicher, flüchtig. D. h., dass nach einem Neustart des Controllers im RAM keine Werte gespeichert sind.

1.4.3 Peripherie

Als *Peripherie* bezeichnet man jene Komponenten eines Mikrocontrollers, die nicht durch CPU und Speicherbausteine abgedeckt werden, insbesondere Komponenten, die eine Schnittstelle zur Außenwelt darstellen. Digitale Ein- und Ausgänge (kurz: I/O für Input/Output) werden z. B. zu den Peripheriebausteinen gezählt. Die meisten Mikrocontrollerboards, auch unser Arduino, bieten eine Vielzahl von digitalen und analogen Ein- und Ausgängen mit verschiedenen Funktionen.

1.5 Programmierung der Mikrocontroller

Ein Programm ist die Beschreibung eines Informationsverarbeitungsprozesses. Im Lauf eines solchen Prozesses wird aus einer Menge von variablen oder konstanten Eingangswerten eine Menge von Ausgangswerten berechnet. Die Ausgangswerte sind entweder selbst Ziel der Informationsgewinnung oder dienen mittelbar zur Reaktion auf die Eingangswerte. Neben den eigentlichen Berechnungen kann ein Programm Anweisungen zum Zugriff auf die Hardware des Computers oder zur Steuerung des Programmflusses enthalten. Ein Programm besteht aus mehreren Zeilen sogenannten Quelltextes. Dabei enthält jede Zeile eine oder mehrere Rechen- oder Steueranweisungen. Neben diesen Anweisungen selbst bestimmt ihre Reihenfolge wesentlich die eingangs beschriebene Informationsverarbeitung. Die Ausführung der den Anweisungen entsprechenden Operationen durch den Steuercomputer erfolgt sequenziell, also der Reihe nach. Eine Folge von Programmanweisungen mit einem bestimmten Ziel nennt man auch *Algorithmus.*

1.5.1 Arduino versteht C

C oder auch ANSI-C ist eine einfach zu erlernende Programmiersprache. C ist eine imperative Programmiersprache, die der Informatiker Dennis Ritchie in den frühen 70er-Jahren an den Bell Laboratories für das Betriebssystem Unix entwickelt hat. Seitdem ist sie weltweit stark verbreitet. Die Anwendungsbereiche von C sind sehr verschieden. Es wird z. B. zur System- und Anwendungsprogrammierung eingesetzt. Die grundlegenden Programme aller Unix-Systeme und die Systemkerne vieler Betriebssysteme sind in C programmiert. Zahlreiche Sprachen wie C++, Objective-C, C#, Java, PHP oder Perl orientieren sich an der Syntax und anderen Eigenschaften von C. Es ist also mehr als lohnend, sich mit dieser Programmiersprache zu beschäftigen, da man später auch leicht auf andere Mikrocontrollersysteme umsteigen kann. Für fast alle Mikrocontroller existiert ein freier C-Compiler, den die Hersteller zum Download anbieten. Das C von Arduino ist jedoch um einiges einfacher gehalten als die professionellen C-Compiler und nimmt dem Anwender dadurch sehr viel Arbeit ab. Vor allem um die komplizierten Hardware-Routinen muss man sich bei Arduino nicht kümmern, da sie bereits als feste Befehle in der Entwicklungsumgebung integriert sind. Zudem gibt es mittlerweile für fast jede Hardware eine Arduino-Bibliothek, die einfach eingebunden wird – und schon kann man mit der neuen Hardware, z. B. einem digitalen Drucksensor, kommunizieren.

3 Arduino-Shields

Es gibt eine Menge verschiedener Erweiterungs-Boards, die mit den Arduino-Boards verwendet werden können. Wenn man sich im Internet umsieht, findet man fast monatlich neue Boards und nützliche Erweiterungen. Die Erweiterungs-Boards werden in der »Arduino-Gemeinde« *Shields* genannt und besitzen alle den gleichen Formfaktor. Das hat den Vorteil, dass man sie einfach auf die Arduino-Boards aufstecken kann. Ausgenommen sind die kleinen Units und das LilyPad.

3.1 Arduino ProtoShield

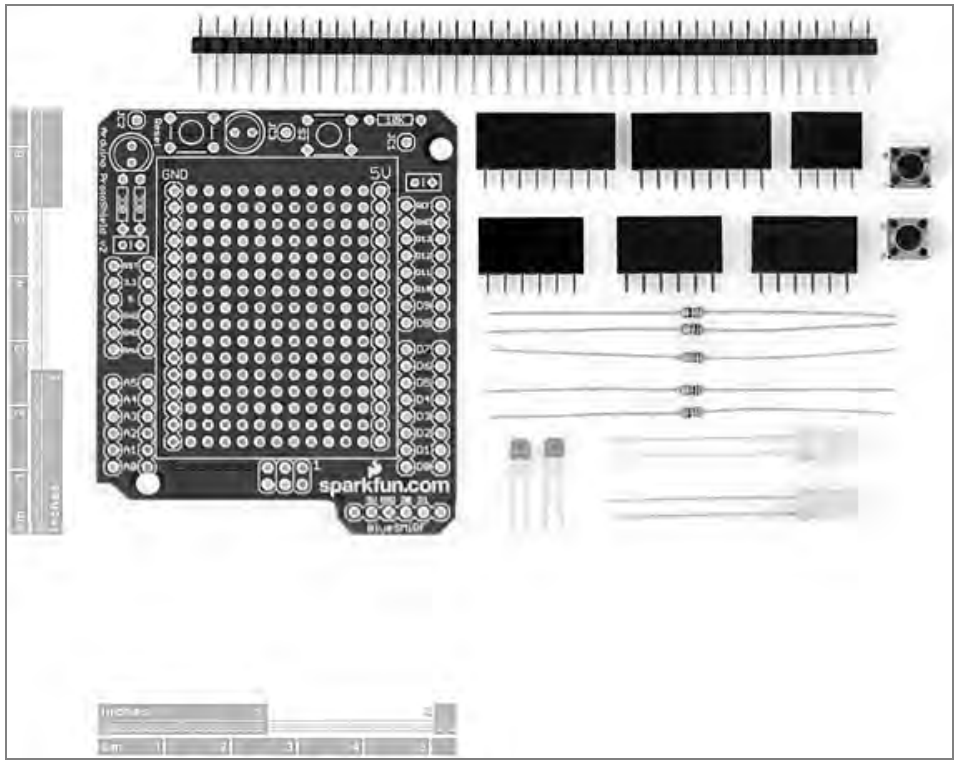


Bild 3.1: Arduino-ProtoShield-Kit (Quelle: SparkFun)

Für eigene Basteleien ohne Lötarbeit bietet sich das ProtoShield an. Es ermöglicht Experimente auf einem kleinen Steckbrett.

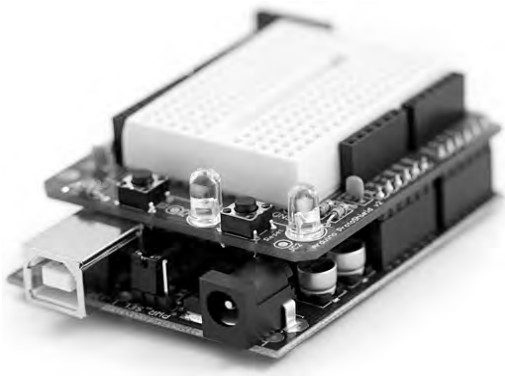


Bild 3.2: Arduino UNO mit ProtoShield (Quelle: SparkFun)

3.2 Ardumoto

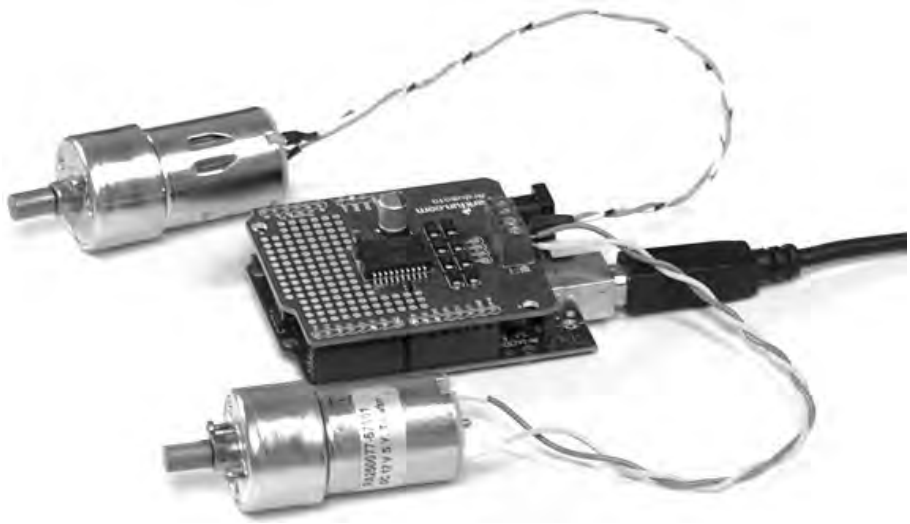


Bild 3.3: Ardumoto – Motor Driver Shield (Quelle: SparkFun)

Das Motor Driver Shield Ardumoto ist ideal, um kleine Motoren anzusteuern. Die Anschlussdrähte der Motoren werden einfach an die Schraubklemmen des Motor-Shields angeschlossen, und ein kleines Programm lässt die Motoren mit gewünschter Geschwindigkeit und Richtung drehen. Die technischen Daten entsprechen den verwendeten Motortreiber-ICs L298. Das Datenblatt finden Sie auf der mitgelieferten CD. Ein vergleichbares Shield ist das Arduino Motor-Shield

3.3 TellyMate

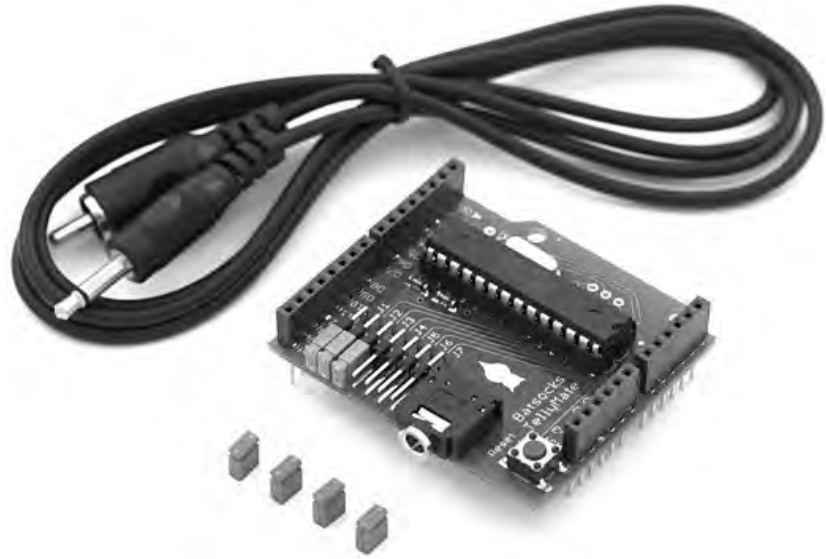


Bild 3.4: TellyMate (Quelle: SparkFun)

Das TellyMate ist wohl das genialste Shield, das es für Arduino gibt. Seine Einsatzmöglichkeiten sind fast unbegrenzt. Es ermöglicht, Daten oder auch einfach nur Texte oder Grafiken auf dem TV-Bildschirm darzustellen. Es macht den Fernseher zum Arduino-Display. Der Arduino-Mikrocontroller verwendet zur Kommunikation mit TellyMate die serielle Schnittstelle.

Features:

- Arduino-TV-Ausgabe
- PAL- oder NTSC-Composite-Video
- Arbeitet mit Serial.println () usw.
- 38 x 25 Zeichen
- Darstellung der Zeichen schwarz-weiß
- Einfache Grafiken
- Einfache Programmierung

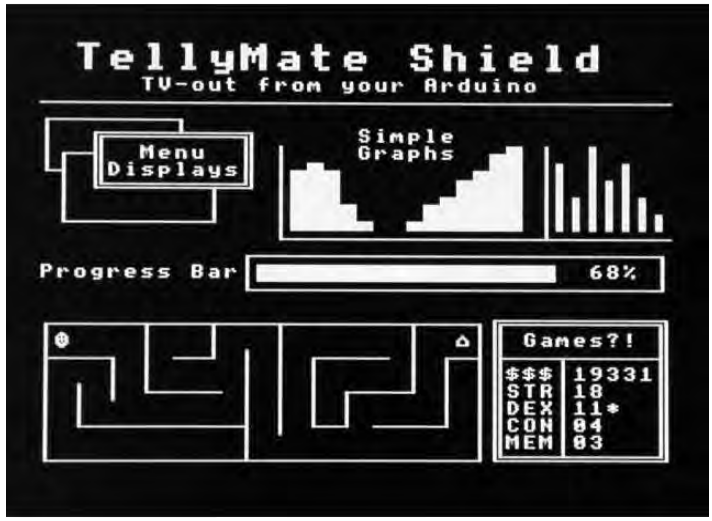


Bild 3.5: TellyMate in Aktion (Quelle: SparkFun)

3.4 XBee-Funkmodule

Bild 3.6: XBee ZNet 2.5 OEM-Modul (Quelle: SparkFun)

Wer drahtlos Daten übermitteln möchte, sollte sich die XBee-Module zulegen. Sie stellen eine drahtlose serielle (UART-)Verbindung her. Man könnte damit zwei Arduino-Boards oder einen PC und ein Arduino-Board über eine Funkstrecke kommunizieren lassen.

Technische Daten

- Betriebsspannung: 2,8 V bis 3,4 V
- Frequenz: ZigBee Standard, 2,4-GHz-ISM-Band

- Sendeleistung: 0 dbm (1 mW)
- Empfindlichkeit: -92 dbm
- Reichweite: 30 m in Gebäuden / 100 m draußen (abhängig von Umgebungsbedingungen)
- Stromaufnahme TX: 45 mA, RX: 50 mA, Stand-by: 10 µA
- Datenrate (über Funk): 250.000 bps
- Datenrate (Interface): 1.200–115.200 bps
- Serielle Schnittstelle: 0 V/3,3 V, zum Anschluss an den PC ist ein 3,3-V-Pegelwandler (MAX3232) zwingend erforderlich
- Standard: kompatibel zu ZigBee/802.15.4
- Topologien: Point-to-Point, Point-to-Multipoint
- Abmessungen: 24,38 x 27,61 x 4mm, 2-mm-Raster

3.5 Ethernet Shield

Bild 3.7: Ethernet Shield (Quelle: Solarbotics)

Das Arduino Ethernet Shield ermöglicht, ein Arduino-Board mit dem Netzwerk/Internet zu verbinden. Es basiert auf dem Wiznet W5100 Ethernet-Chip. Der Wiznet W5100 bietet ein Netzwerk(IP)-Stack, der TCP und UDP unterstützt. Er ermöglicht zudem bis zu vier gleichzeitige Socket-Verbindungen. Arduino bietet eine umfangreiche Bibliothek und verschiedene Beispielprogramme an, um den Einstieg in die Netzwerkwelt zu erleichtern.

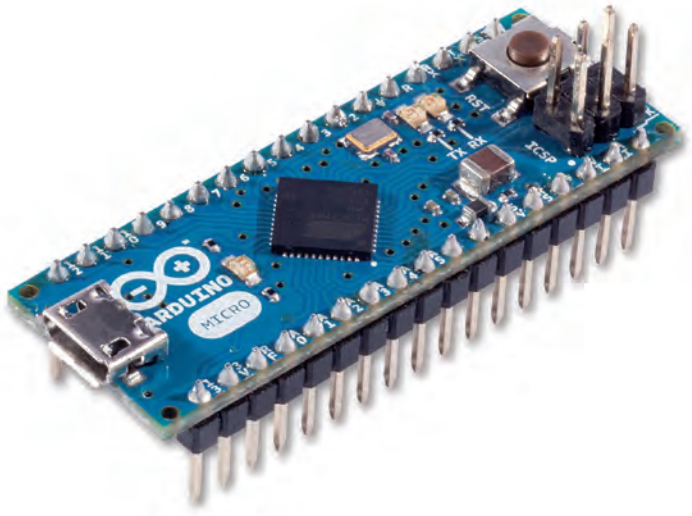
Tipp: Eine Übersicht über alle verfügbaren Arduino-Boards gibt es unter *http://arduino.cc/en/Main/Hardware*.

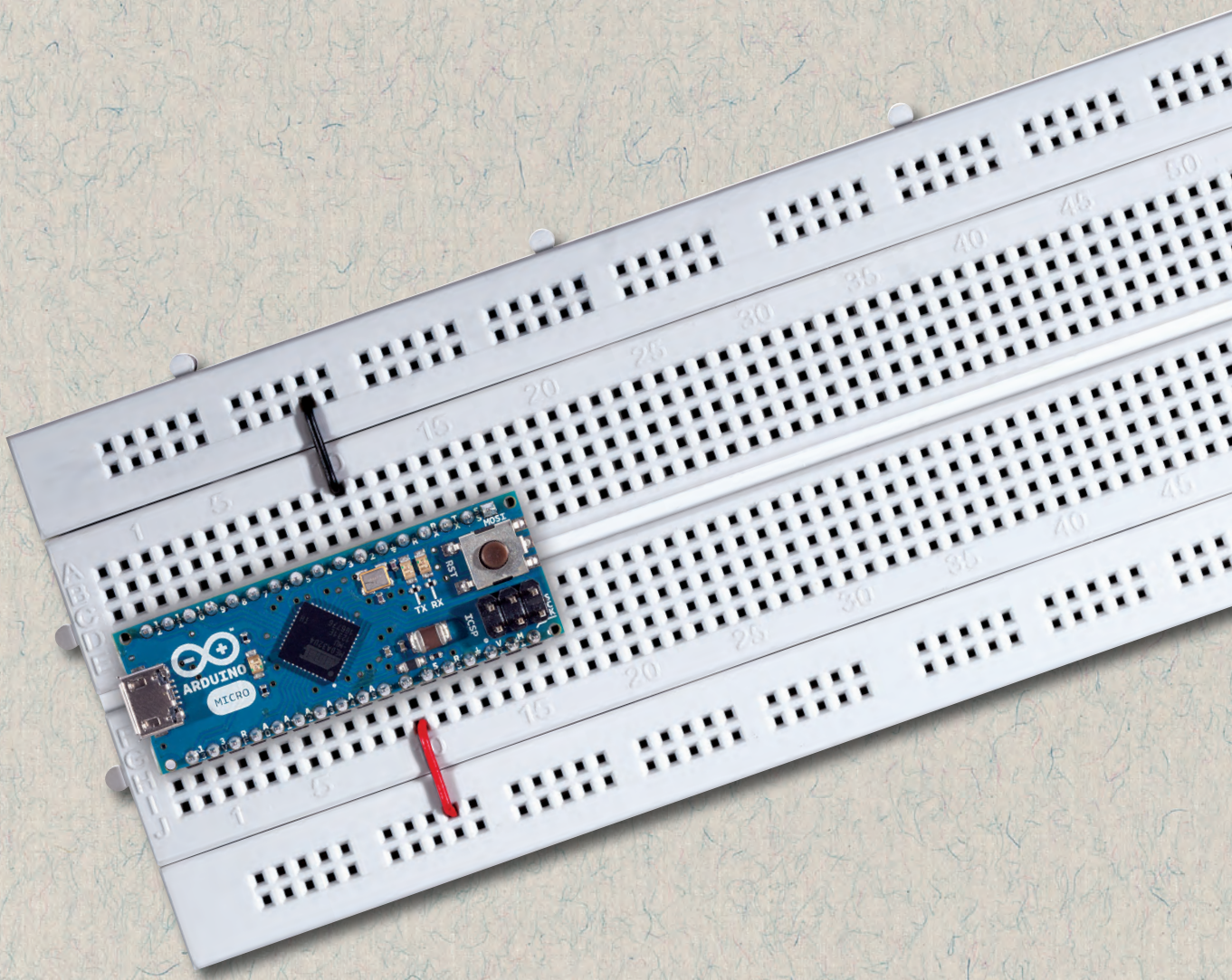
Fabian Kainka

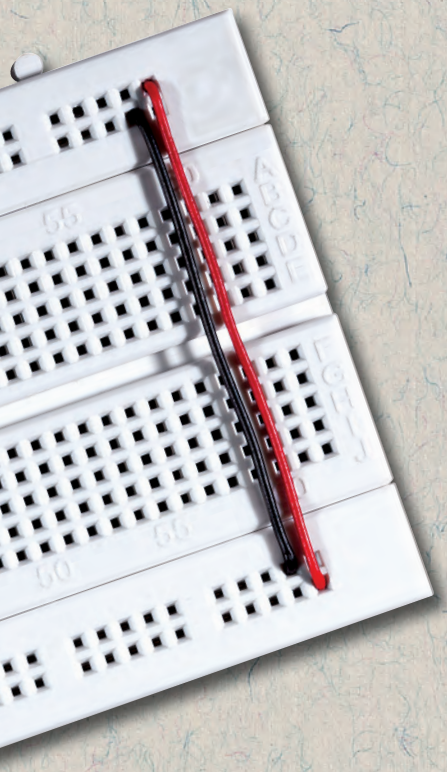
Das Franzis Starterpaket ARDUINO™ MICRO

Das Handbuch

Inhalt




ARDUINO
MICRO
TX RX
RST
ICSP
MOSI
A B C D E F G H I J
1 5 10 15 20 25 30 35 40 45 50



2 HINTERGRUND-WISSEN

Einleitung
Die Hardware
Die Elemente des Boards
Die Software

Einleitung

Arduino ist ein Thema, an dem man in der Mikrocontroller-Welt nicht mehr vorbeikommt. Geschaffen wurde die Arduino-Plattform ursprünglich vor allem für Künstler, Designer und Bastler. Die Mischung aus leicht verständlicher Software und unkomplizierter Hardware macht den Einstieg besonders einfach.

Was steckt hinter dem Namen „Arduino"?

Die Platine wurde nach der Lieblingsbar der Entwickler benannt. In Ivrea, dem italienischen Firmensitz, hatte bis ins Jahr 1005 der italienische König Arduino geherrscht. Nach Re Arduino wurde wiederum die Bar benannt.

Was zeichnet Arduino gegenüber anderen Entwicklungsplattformen aus?
Die Software ist nicht nur einfach, sie ist auch auf gängigen Betriebssystemen nutzbar (Windows, OSX, Linux). Das komplexere AVR-C wurde vereinfacht. Ergebnis war das übersichtlichere Arduino-C. Dennoch kann ein C-Code direkt in ein Arduino-Programm eingefügt werden, sodass Programmierer auf keine Funktion verzichten müssen. Zudem lassen sich auch C-Libraries einbinden. Diese werden in der Arduino-Community offen verbreitet und machen komplizierte Prozeduren für jedermann zugänglich.

Alles in allem bietet Arduino die ideale Plattform, um Ihrer Kreativität freien Lauf zu lassen.

Dieses modulare Konzept wurde auch in der Hardware konsequent weitergeführt. Nicht nur, dass Programmer und Controller auf eine Platine integriert wurden, das Board lässt sich zusätzlich mit Shields um elementare Funktionen erweitern. Egal, ob man sein System vom Internet aus zugänglich machen oder eine eigene Funkkommunikation zwischen verschiedenen Controllern aufbauen möchte, ein vorgefertigtes Shield steht schon bereit und erspart die Planung einer Hardwareerweiterung. Und natürlich ist es trotzdem möglich, schnell mit einem Steckboard ein paar Versuche zusammenzustecken oder sogar sein eigenes Shield zu bauen. Ein weiterer Pluspunkt: Die Soft- und Hardware arbeitet auf Open-Source-Basis. Das bedeutet: Jeder kann die vorliegenden Beispiele, Programme und Tools frei nutzen. Und jeder darf seine eigene Library oder Hardware entwickeln und dennoch den Arduino-Bootloader und die Software verwenden.

Die Hardware

Arduino Micro ist eine Platine mit vielen Bauteilen und integrierten Schaltungen. Aber was steckt eigentlich alles auf so einer Platine? Was kann Ihr Mikrocontroller, und welche Funktion haben die LEDs? Diese und andere Fragen wollen wir jetzt klären.

Was unterscheidet den Micro von den bisherigen Arduino-Boards?

Vor Ihnen liegt vermutlich gerade der Arduino Micro. Dieser Arduino fällt vor allem durch seine Größe auf. Er wurde so gebaut, dass er perfekt auf ein Steckboard passt. Dadurch ist es leicht, ihn mit seinen eigenen Schaltungen zu erweitern.
In den Funktionen ist der Arduino Micro dem Arduino Leonardo ähnlich. Wie im größeren Verwandten kommt auch hier ein ATMega32U4 zum Einsatz. Der Mikrocontroller mit eingebauter USB-Funktion erlaubt es, den Controller direkt zu programmieren, ohne den Umweg über einen anderen Controller zu nehmen. Nur dadurch ist es überhaupt möglich, die Hardware auf so kleinem Raum unterzubringen. Der eingebaute USB-Controller hat aber noch einen weiteren Vorteil. Er ermöglicht es, den Controller als USB-Gerät, z. B. als Tastatur, am PC anzumelden. Dadurch ist der Arduino Micro mit ähnlich vielen Funktionen wie der Leonardo ausgestattet, besitzt aber ein viel handlicheres Format.

Quickcheck Arduino Micro - Die Unterschiede

1. Der Arduino Micro basiert auf dem ATmega32U4.
2. Der Arduino Micro ist die kleinere Variante des Arduino Leonardo.
3. Die integrierte USB-Kommunikation ermöglicht dem Micro, von einem verbundenen Computer als Maus und Tastatur erkannt zu werden.

Auf dem nachfolgenden Bild finden Sie die wichtigsten Hardwareelemente, die später noch detaillierter dargestellt werden.

Spannungsregler
Spannungsregler dienen der Stabilisierung elektrischer Spannungen.

LEDs
Insgesamt befinden sich vier LEDs auf dem Arduino-Board

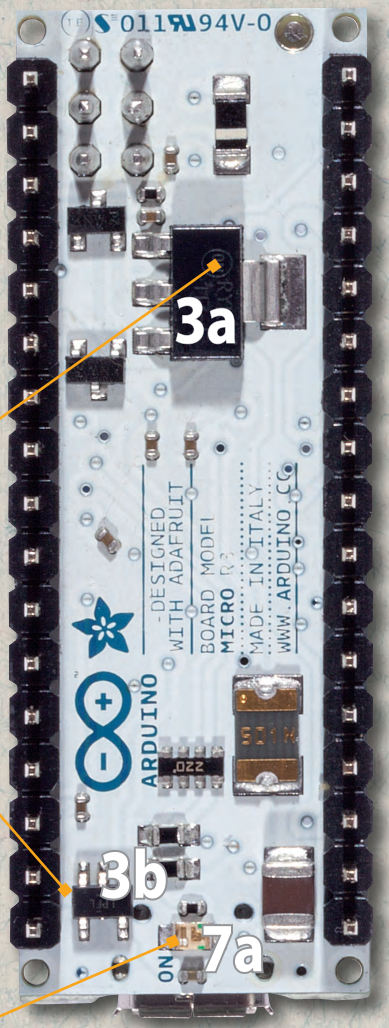
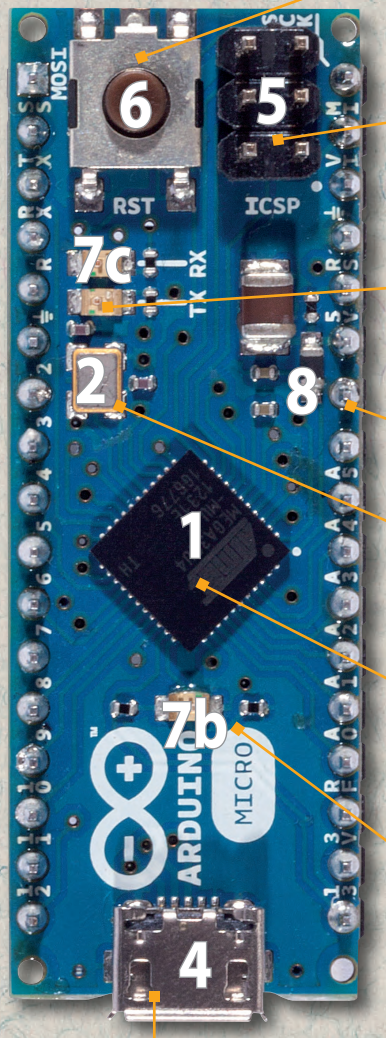


Abb. 5: Der Arduino Micro mit seinen Bauteilen



Der Reset-Button
Er setzt den Controller zurück.

ICSP-Anschluss
Mit diesem Anschluss und einem externen Programmiergerät ist es möglich, den Arduino zu programmieren.

LEDs
Insgesamt befinden sich vier LEDs auf dem Arduino-Board

Power-Pins
Bei Power-Pins handelt es sich um Anschlüsse, an denen Sie Spannungen abgreifen oder zuführen können.

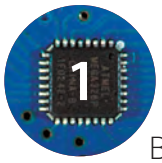
Quarz
Der Quarz ist der Taktgeber des Controllers.

Mikrocontroller
Er ist Herz und Hirn des Boards: Es handelt sich hier um einen ATmega32U4.

LEDs
Insgesamt befinden sich vier LEDs auf dem Arduino-Board

Der Micro-USB-Anschluss
Er versorgt den Arduino mit Programmen und liefert Strom für Board und Peripherie.

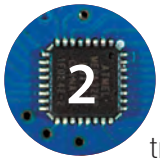
Die Elemente des Boards



Der Mikrocontroller

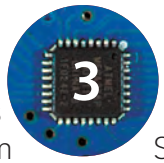
Der Controller stellt Herz und Hirn des Boards dar. Hier wird das Programm hineingeladen und ausgeführt. Es handelt sich in diesem Fall um einen ATmega32U4. Die Eckdaten des Controllers:

- 32 KB Flash-Speicher
- Insgesamt 44 Pins (davon 20 Input-/Output-Pins)
- 16 MHz maximale Clock-Frequenz
- CPU: 8-Bit-AVR
- Integrierter USB-Controller
- Insgesamt 13 externe Interrupt-Pins, 7 PWM-Kanäle und 12 analoge Eingänge (ADCs)
- Zugelassene Arbeitsspannung: 2,7 – 5,5 V.



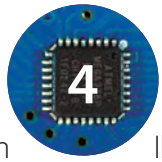
Der Quarz

Der Quarz ist der Taktgeber des Controllers. Es handelt sich hier um einen 16-MHz-Quarz. Der ATmega32U4 wird also mit maximaler Geschwindigkeit betrieben.



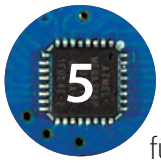
Die Spannungsregler

Auf dem Bild kann man gut die beiden Spannungsregler erkennen. Der größere von beiden sorgt für eine konstante 5-V-Spannung und der kleinere für eine konstante 3,3-V-Spannung.



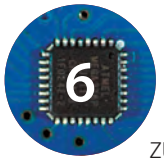
Der USB-Anschluss

Dieser Anschluss dient nicht nur dazu, Ihren Arduino mit den neuesten Programmen zu versorgen, sondern er liefert praktischerweise noch 5 V und bis zu 500 mA für das Board und die angeschlossene Peripherie. Es ist also kein Problem, auch ein paar LEDs anzuschließen, selbst wenn keine externe Stromversorgung zusätzlichen Strom liefert.



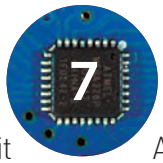
Der ICSP-Anschluss

ICSP (oft auch ISP) ist die Abkürzung für „In-circuit serial programming“ . Mit diesem Anschluss und einem externen Programmiergerät (wie z. B. dem STK500) ist es möglich, den Arduino direkt zu programmieren oder den internen Bootloader zu überschreiben. Dies sollten Sie allerdings nur dann tun, wenn Sie erfahren im Umgang mit Mikrocontrollern sind. Sie können diese Schnittstelle aber auch benutzen, um mit einem anderen Controller über das SPI-Protokoll zu kommunizieren.



Der Reset-Button

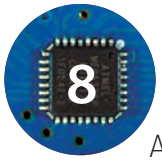
Der Reset-Button setzt den Controller zurück. Eingespielte Programme bleiben erhalten, werden aber an den Anfang zurückgesetzt. Dies ist nützlich, wenn Sie ein laufendes Programm unterbrechen wollen oder aber durch einen Programmierfehler keine Verbindung mehr zum Controller herstellen können. Mehr Informationen zu derartigen Themen finden Sie im Anhang.



Die LEDs

Es befinden sich vier LEDs auf Ihrem Arduino-Board:

a) On: Diese LED leuchtet, sobald das Board mit Strom versorgt wird.
b) L: Die LED L ist direkt mit dem Pin13 des Boards verbunden. Dahinter steckt der Gedanke, dass Sie eine LED direkt ansteuern und testen können, ohne externe Hardware anschließen zu müssen.
c) TX und RX: Anhand dieser beiden LEDs kann man die serielle Kommunikation des Boards beobachten. Wenn das Board programmiert wird, blinkt die LED RX, die das Ankommen von Daten visualisiert. Die TX-LED zeigt dagegen an, wenn Daten vom Controller gesendet werden. Wenn Sie später Peripheriegeräte an die RX- und TX-Pins anschließen, können Sie anhand dieser LEDs überprüfen, ob überhaupt etwas gesendet oder empfangen wird.



Die Pins

Die Pin-Kennzeichnungen sind beim Arduino Micro nicht so eindeutig wie bei größeren Arduinos. Jedoch kann man anhand der verwendeten Symbole und Abkürzungen immer erkennen, um welchen Pin es sich handelt.

Die Power-Pins

Bei diesen Pins handelt es sich nicht um I/O-Pins, sondern um Anschlüsse, an denen Sie Spannungen abgreifen oder zuführen können.

a) ⏚: Masse-Pins (GND). Es gibt insgesamt drei. Benötigt werden sie vor allem, um beispielsweise eine LED anzuschließen oder einen Eingang auf LOW (Masse-Niveau) zu schalten.

b) 5 V: Der 5-V-Ausgang liefert eine konstante Spannung von 5 V für externe Hardware. Die Belastbarkeit hängt von der gewählten Stromversorgung ab, entweder über USB oder über ein externes Netzteil. Bis zu 100 mA können gefahrlos entnommen werden.

c) 3,3 V: Der 3,3-V-Ausgang liefert eine konstante Spannung von 3,3 V von einem eigenen Spannungsregler. Diese Spannung wird von manchen Shields als Arbeitsspannung benötigt. Sie können aber auch eigene Hardware darüber versorgen. Die maximale Belastbarkeit dieses Pins beträgt 50 mA.

d) VI: VI steht für V_{in}. Der Pin ermöglicht es, eine externe Batterie mit 9 bis 12 V als Spannungsversorgung anzuschließen. Der negative Pol der Batterie wird dabei einfach an einen beliebigen GND-Pin angeschlossen.

Die optionalen Zusatz-Pins

Bei diesen Pins handelt es sich um speziell ausgeführte Pins, die vor allem für Shields von Bedeutung sind.

a) RF: An dem RF-Pin liegt die Referenzspannung (Vergleichsspannung) für analoge Messungen an. Der Pin wird wichtig, wenn Sie Spannungen mit dem ADC messen. Beim Micro liegen hier im Normalfall 5 V an.

b) RS/R: An diesen Stellen wird der Reset-Pin des Controllers für Shields oder andere Anwendungen zugänglich gemacht. Um einen Reset auszuführen, müssen Sie diesen Pin mit GND verbinden.

c) MOSI/SCK/MI/SS: Diese drei Pins können mit SPI-Library angesprochen werden und somit eine SPI-Verbindung mit einem anderen Controller aufbauen. Diese Pins lassen sich auch über den ICSP-Anschluss erreichen. Der SS-Pin (Slave-Select-Pin) ist gleichzeitig mit der RX-LED verbunden.

Die 20 I/O-Pins

Allgemein können Sie jeden dieser Anschlüsse als Eingang für Schalter oder als Ausgang etwa für LEDs nutzen. Der maximale Strom pro Pin beträgt 40 mA. Das reicht, um eine LED mit passendem Vorwiderstand hell leuchten zu lassen. Einige dieser I/O-Pins erfüllen noch Sonderfunktionen. Hier eine kurze Zusammenfassung:

a) PWM-Pins (3, 5, 6, 9, 10, 11, 13): Bei diesen Pins handelt es sich um Ausgänge, die sich via PWM (Pulsweitenmodulation) ansteuern lassen, um z. B. die Helligkeit einer LED zu steuern. Beim Arduino Micro können acht Pins mit diesem Verfahren angesteuert werden. Genaueres finden Sie in Kapitel 3 im Abschnitt „PWM".

b) Analoge Eingänge (A0-A5 & Pins 4, 6, 8, 9, 10, 12): Gerade bei Sensoren, z. B. einem Umgebungslichtsensor, ist es wichtig, eine Spannung messen zu können. Dafür gibt es am Micro insgesamt 12 analoge Eingänge. Diese befinden sich gut gekennzeichnet an den Pins A0 bis A5 und an den digitalen Pins 4, 6, 8, 9, 10 und 12.

c) RX und TX (Pin 0 & 1): Mithilfe der Anschlüsse RX und TX können Sie Daten über die integrierte serielle Schnittstelle des Controllers empfangen (RX) und senden (TX). Dadurch können Sie unabhängig vom USB-Controller eine serielle Verbindung zu Ihrem PC oder anderen Controllern herstellen. Gleichzeitig sind die beiden Pins aber auch die digitalen I/O-Pins 0 und 1.

d) SDA&SCL: Mit diesen Pins lassen sich über den I^2C-Bus andere Controller, Shields oder Module ansteuern. Beim Arduino Micro sind die beiden Anschlüsse an den I/O-Pins 2 (SDA) und 3 (SCL) zu finden.

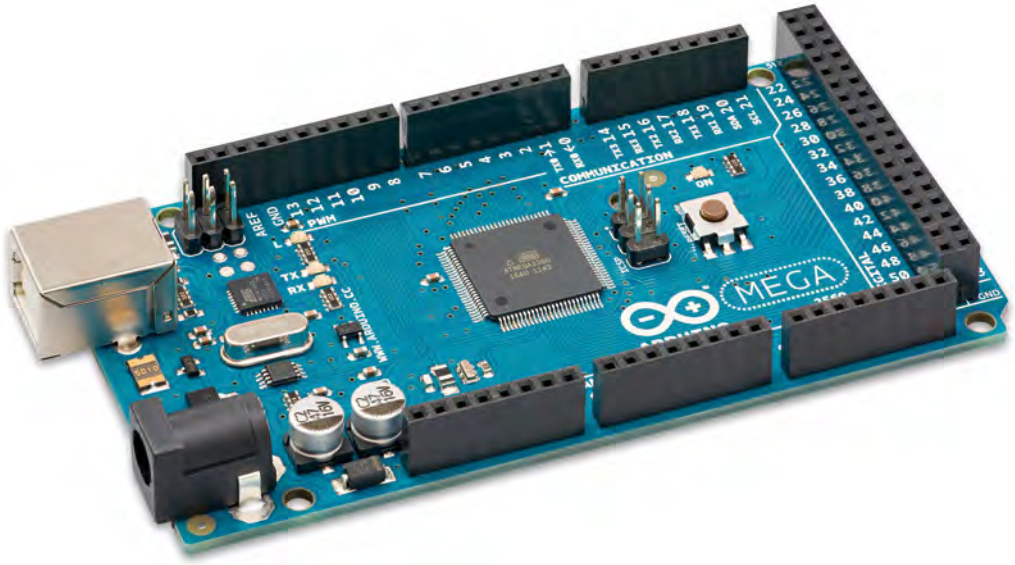
e) Externe Interrupts (Pin 0 – 3): Wird ein Interrupt ausgelöst, wird das laufende Programm unterbrochen und ein Unterprogramm ausgeführt. Dadurch kann ein Programm flexibel auf ein externes Signal reagieren. Eine solche Signalquelle können Sie an die Pins 0 bis 3 anschließen.

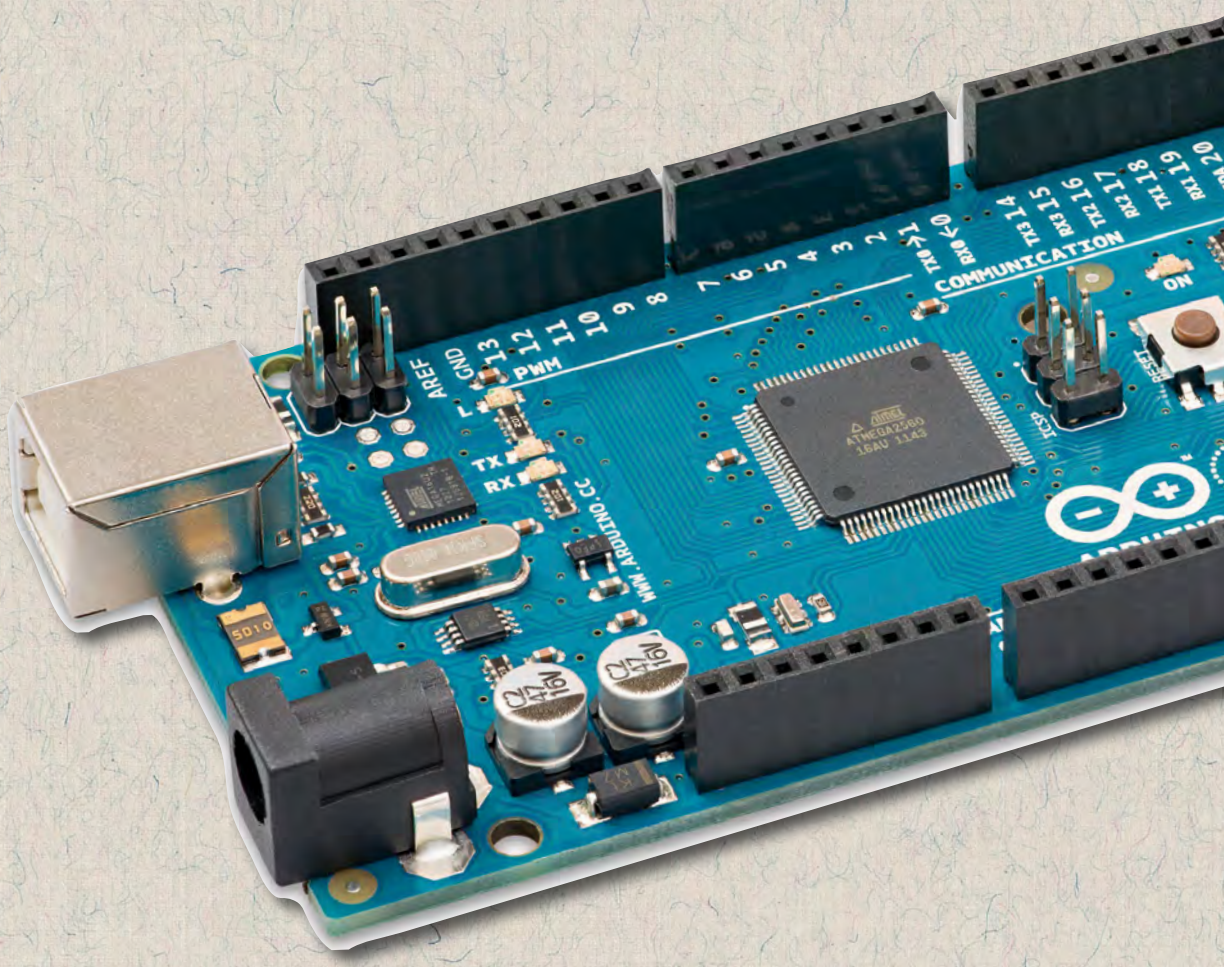
Fabian Kainka

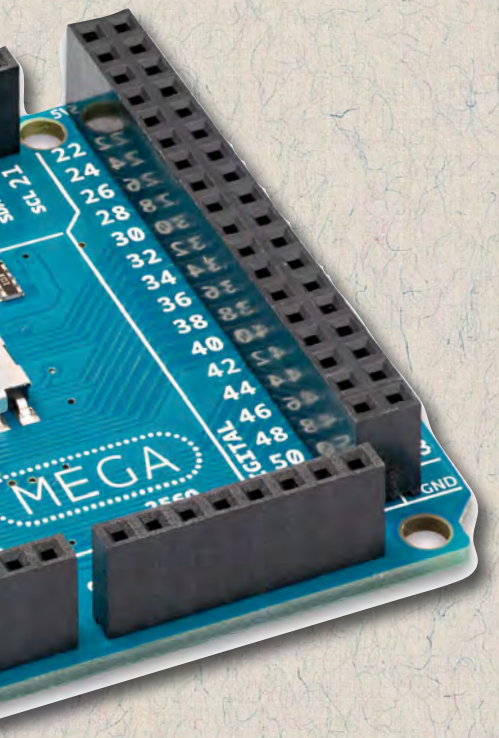
Das Franzis Starterpaket ARDUINO™ MEGA 2560

Das Handbuch

Inhalt




AREF
GND
13
12
PWM
11
10
9
8
7
6
5
4
3
2
TX→1
RX←0
COMMUNICATION
TX3 14
RX3 15
TX2 16
RX2 17
TX1 18
RX1 19
RX0 20
ON
RESET
ICSP
TX
RX
SDIO
ATMEGA2560
16AU 1143
WWW.ARDUINO.CC



SCHNELLSTART

Starthilfe
Drei Beispiele

Starthilfe

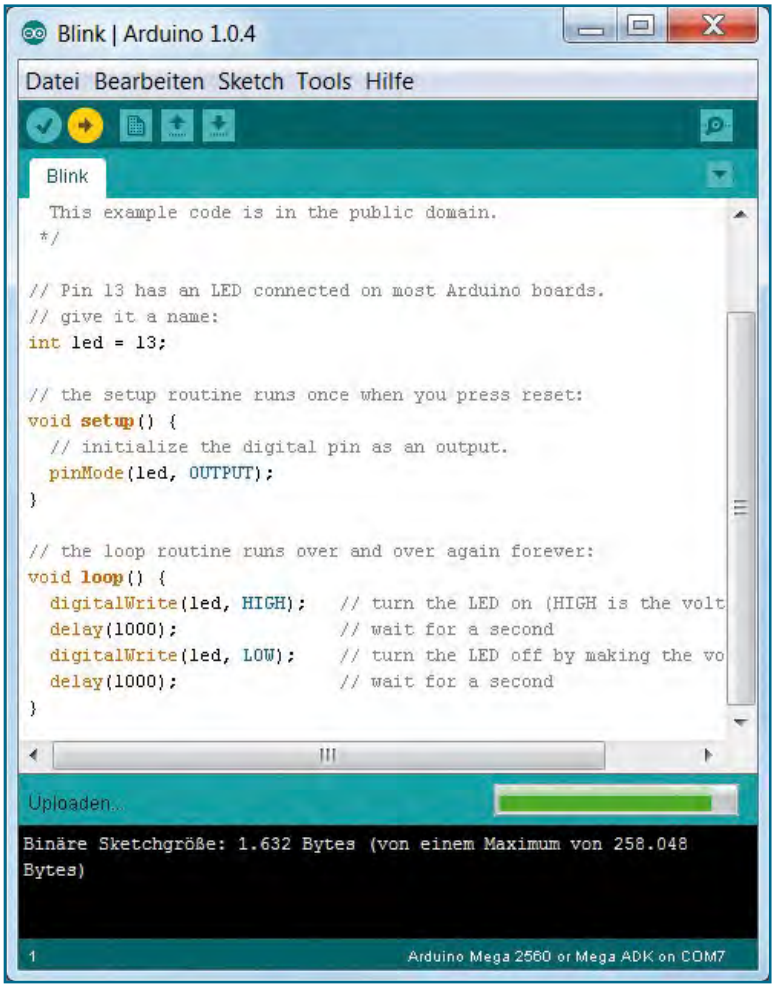


Abb. 1: Die Programmierumgebung während des Uploads

Möchten Sie jetzt gleich loslegen und ein Programm auf Ihrem Arduino ausprobieren? Wenn ja, finden Sie an dieser Stelle zunächst einmal eine Schnellstartanleitung und ein paar Beispiele, bevor Sie sich den Details zuwenden.

Sollten Sie beim Schnellstart auf Probleme stoßen, die sich daraus ergeben, dass Sie noch nicht so fit in der Materie sind, finden Sie später noch eine ausführlichere Anleitung für die Installation. Aber mit etwas Vorerfahrung reichen die folgenden Schritte für den ersten Erfolg.

So gehen Sie vor

1. Aktuelle Software laden und installieren. Sie finden sie unter http://www.arduino.cc/en/Main/software
2. Controller anschließen.
3. Treiber (nur unter Windows) installieren (Treiber finden Sie im Zip-Ordner).
4. Arduino-Programmierumgebung starten.
5. Unter *Tools > Board* die verwendete Hardware (Arduino 2560 orMega ADK) auswählen.
6. Unter *Tools > Serieller Port* den entsprechenden COM-Port wählen.
7. Unter *Datei > Beispiele > 01.Basics > Blink* das Beispiel laden.
8. Den Upload-Button klicken.
9. Auf die Meldung „Upload abgeschlossen“ warten.
10. Die LED „L“ auf dem Board blinkt – fertig!

Herzlichen Glückwunsch! Auf Ihrem Arduino läuft nun ein funktionsfähiges Programm.

Drei Beispiele

Für die folgenden drei Beispiele benötigen Sie keine zusätzliche Hardware, nur ein kleines Stück Verbindungsdraht. Anhand der Beispiele lässt sich ein erster Eindruck von den Möglichkeiten des Arduino-Controllers gewinnen. Die Beispiele sind in die Arduino-Software integriert und werden nur minimal variiert. Dadurch können Ihnen kaum Fehler unterlaufen, und Sie sammeln erste Erfahrungen mit Ihrem Arduino-Controller.

Tipp: Falls doch etwas schiefgeht, finden Sie Hilfe im Anhang.

Button

Dieses Beispiel finden Sie unter:
Datei > Beispiele > 02.Digital > DigitalInputPullup
Hier wird demonstriert, wie Sie mit einem Schalter eine LED schalten, ohne externe Pullup-Widerstände zu benötigen (die Erklärungen hierzu später):

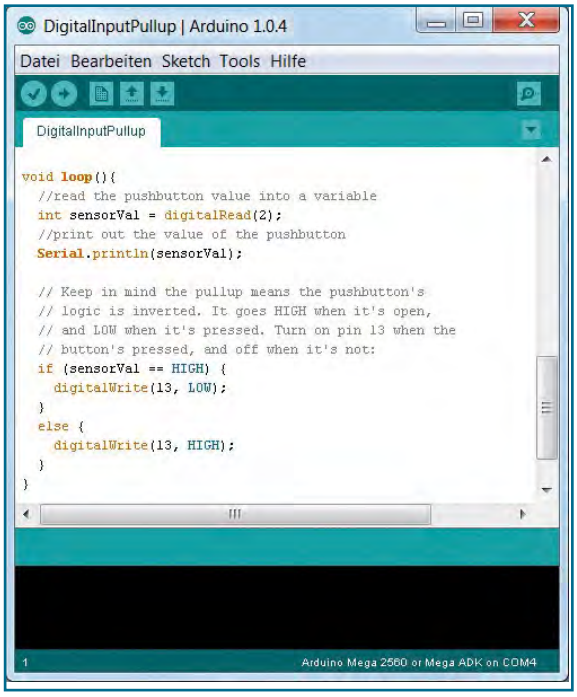


Abb. 2: Die Software mit Serial Monitor Button oben rechts

Nach einem erfolgreichen Upload stellen Sie mit einer Drahtbrücke eine Verbindung vom digital Pin2 (nicht A2) zu einem GND-Pin her. Sie sehen, dass die LED leuchtet. Wenn Sie die Verbindung trennen, geht die LED wieder aus. Zusätzlich sendet das Programm den aktuellen Button-Status an den Serial Monitor (auch Terminal genannt), den Sie mit dem Symbol oben rechts starten können. Immer wenn Sie die Drahtverbindung herstellen, wird eine 0 gesendet, ansonsten eine 1.

Spannung messen

Das folgende Beispiel finden Sie unter:
Datei > Beispiele > 01.Basics > ReadAnalogVoltage
In diesem Sketch wird eine analoge Spannung gemessen und das Ergebnis an das Terminal gesendet.

Nach dem Upload erkennen Sie im Serial Monitor Werte, die aber relativ wild flackern. Dies liegt daran, dass der Port offen ist und Störfelder erfasst. Was Sie tun können: Stellen Sie eine Drahtverbindung vom analogen Pin A0 zu einem GND-Pin her. Nun wird 0.00 angezeigt, also 0 Volt. Wenn Sie das Drahtende statt in den GND-Pin in den 5-V-Pin stecken, werden Sie 5.00 lesen können. Dies ist das Maximum, das der Controller mit der gegenwärtigen Einstellung messen kann. Probieren Sie ruhig einmal aus, was passiert, wenn Sie anstelle des 5-V-Pins den 3,3-V-Pin benutzen.

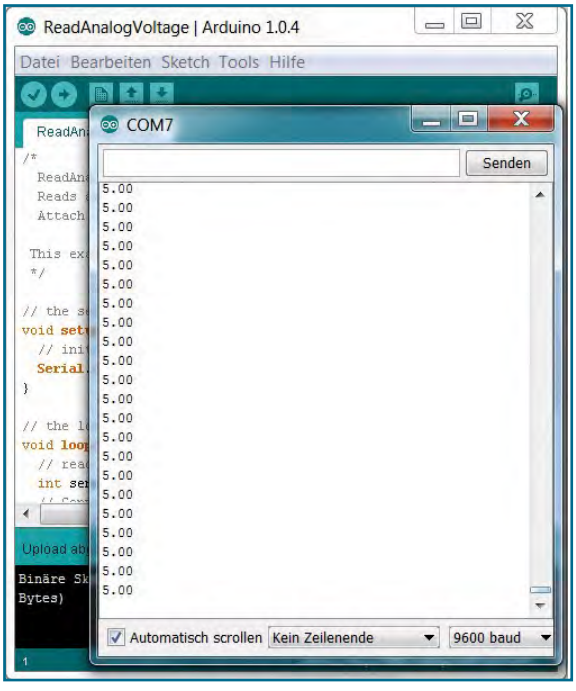


Abb. 3: Das Programm mit offenem Serial Monitor

Blinkfrequenz steuern

Im folgenden Programm wird wieder eine Spannung gemessen. Der Controller wertet das Ergebnis diesmal selbstständig aus und berechnet die Blinkfrequenz der L-LED. Laden Sie das Beispiel in Ihren Controller:
Datei > Beispiele > 03.Analog >AnalogInput

Wenn noch eine Drahtbrücke vom A0- zum 5-V-Pin gesteckt ist, blinkt die LED ca. einmal pro Sekunde. Benutzen Sie den 3.3-V-Pin, blinkt die LED deutlich schneller. Bei offenem Pin ändert sich die Blinkfrequenz ständig und unregelmäßig, und beim GND-Pin bleibt die LED vollständig aus oder leuchtet nur schwach.

Der Controller im Hintergrund hat aus der gemessenen Spannung eine Zeit berechnet, nach der der Zustand der LED wechselt. Mit einem Lichtsensor könnte man so z. B. eine automatisierte Steuerung von Warnsignalen programmieren. Doch dazu später mehr.

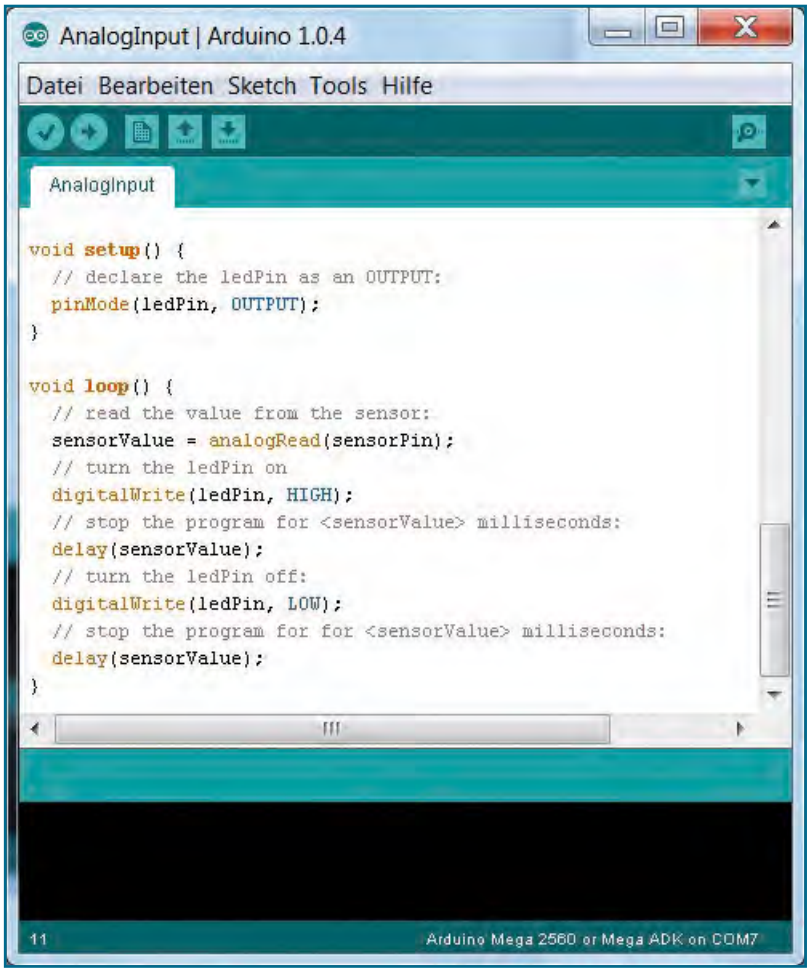


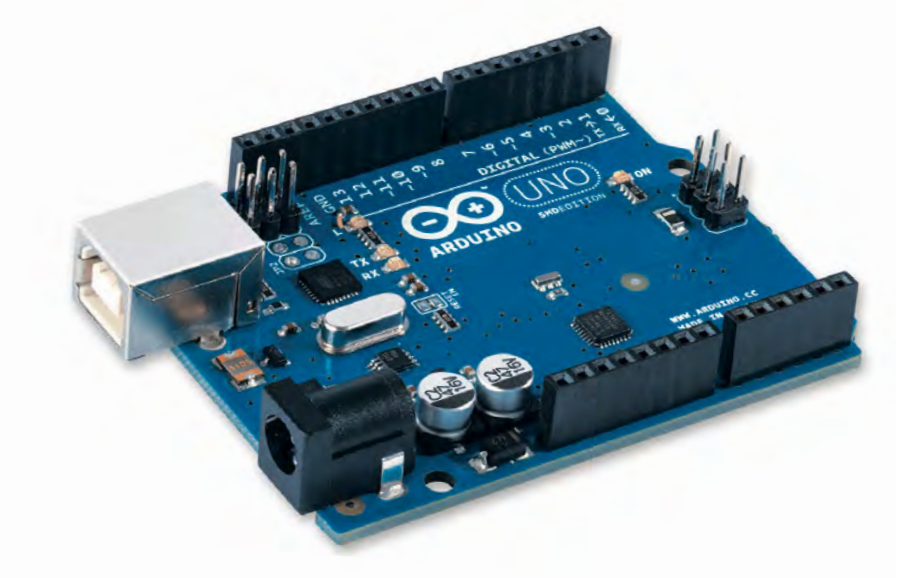
Abb. 4: Der Quelltext in der Programmierumgebung

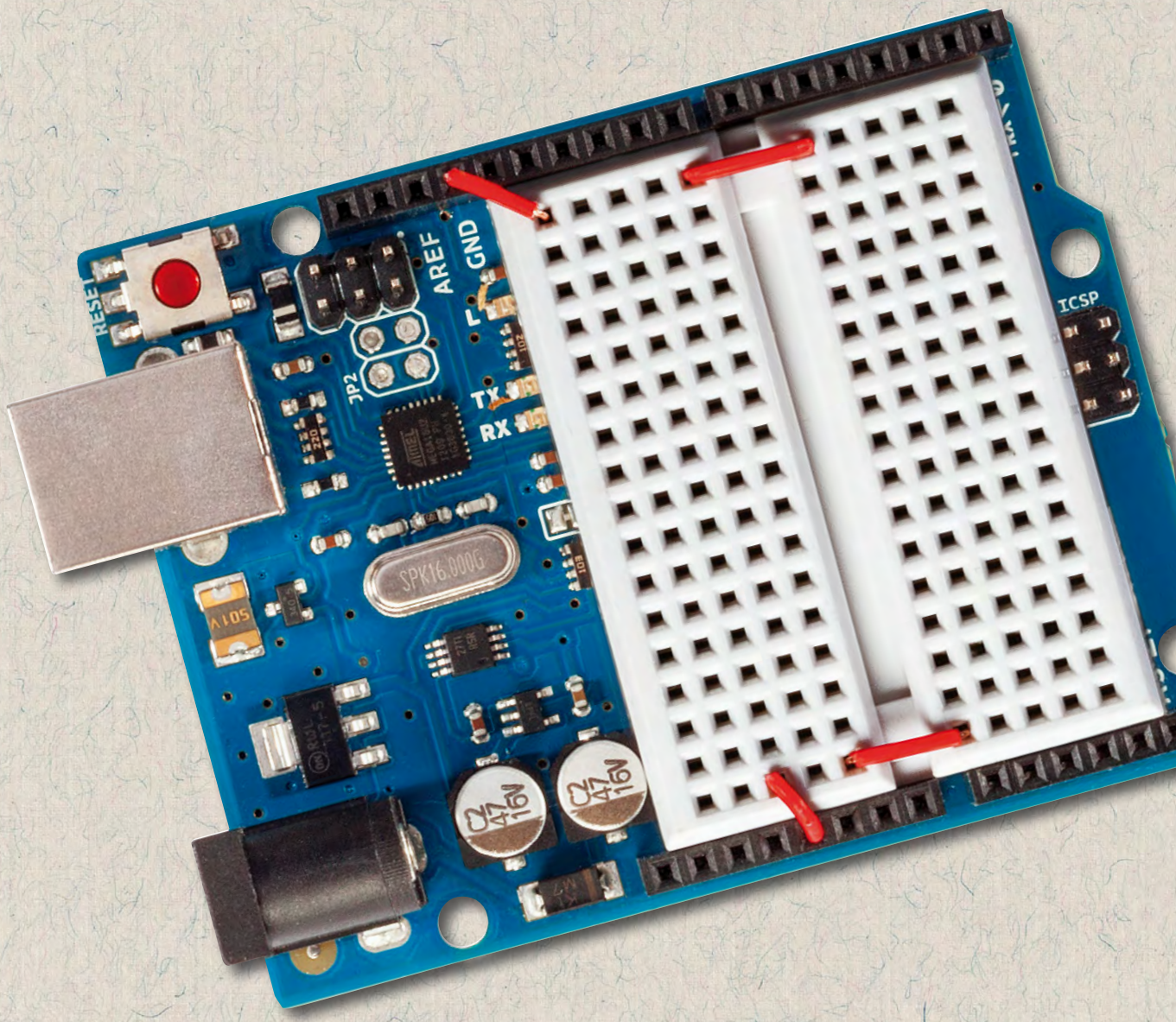
Fabian Kainka

Das Franzis Starterpaket ARDUINO™ UNO

Das Handbuch

Inhalt




RESET
AREF
GND
JP2
TX
RX
ICSP
SPK16.000G
C47
16V
C47
16V

DO IT YOURSELF

Vorbereitungen
Projekte zum Nachbauen
Tipps und Hilfestellungen

Vorbereitungen

So bereiten Sie sich auf die praktischen Versuche mit dem Arduino vor.

Nach dem theoretischen Einstieg in die Arduino-Welt ist es nun an der Zeit, sich mit den praktischen Grundlagen zu beschäftigen. Als ersten Schritt sollten Sie sich ein praktikables System für die Hardwareaufbauten aneignen. Es ist natürlich nicht sehr klug, gleich den ersten Versuch auf ein Shield zu löten, ganz besonders, wenn man noch Einsteiger ist. Ich persönlich habe mir ein kleines Steckboard besorgt, das zufällig genau zwischen die Anschlüsse des Arduino passt. Der Vorteil: Dieses System ist sehr kompakt. Der Nachteil: Die Pin-Beschriftungen sind verdeckt.

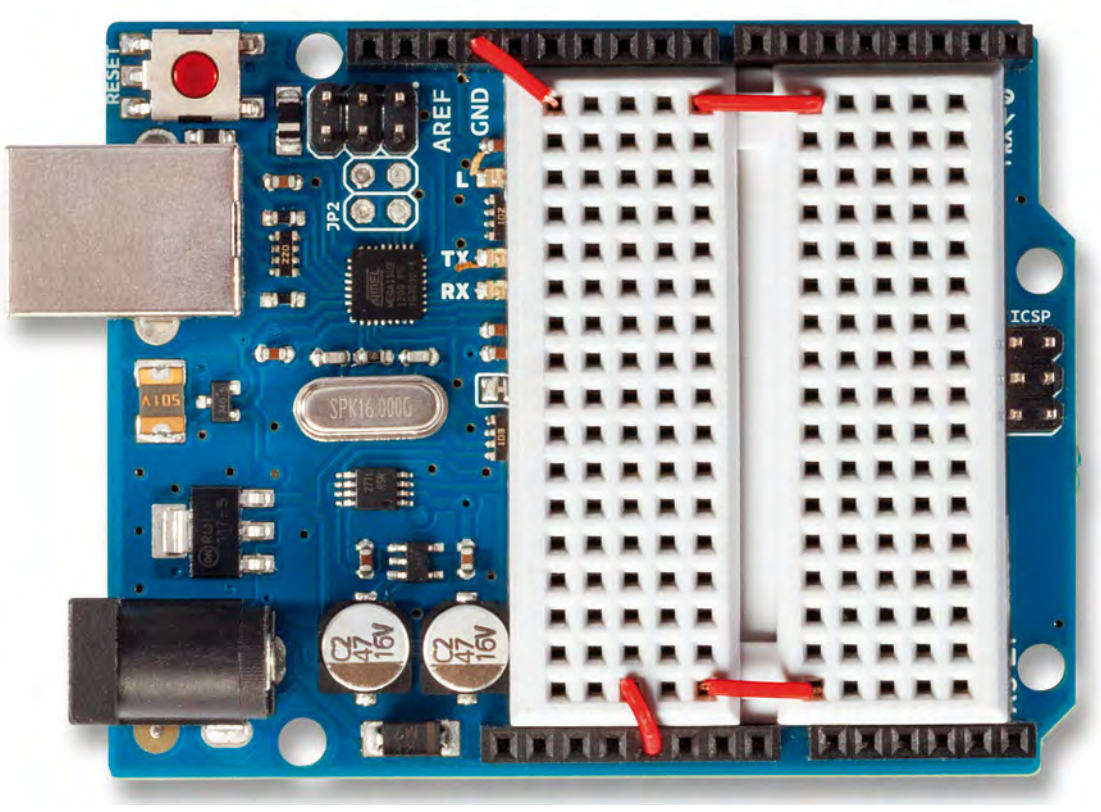


Abb 19: Aufbau mit kleinem Steckboard

Wichtig zum Verständnis meiner Aufbauten ist, dass auf diesem Board die Querkontakte in 5er-Blöcken verbunden sind. Außerdem habe ich mir, wie man auf dem Bild sehen kann, auf die unteren beiden Kontaktblöcke die Betriebsspannung von + 5 V und auf die oberen beiden eine Masseleitung gelegt. Dadurch ist der Zugang zu diesen wichtigen Versorgungsleisten vereinfacht.

Wichtig zum Verständnis: Auf diesem Board sind die Querkontakte in 5er-Blöcken verbunden.

Es gibt natürlich auch andere Möglichkeiten, eine handliche Entwicklungsumgebung zu schaffen. Auf dem nächsten Bild sehen Sie eine Variante mit einem größeren Steckboard. Auch hier wurden 5 V und Masse auf durchgehende Kontakte gelegt und somit leicht zugänglich gemacht.

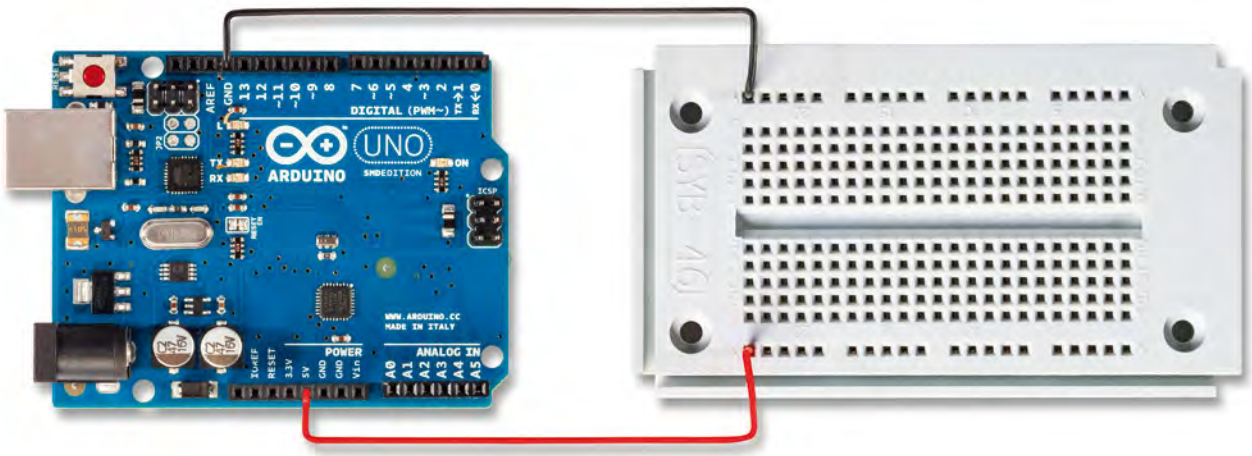


Abb 20: Aufbau mit einem großen Steckboard

Wofür auch immer Sie sich entscheiden: Wichtig ist, dass Sie den Überblick über Ihr System nicht verlieren und schnell und unkompliziert Bauteile austauschen oder Schaltungen erweitern können. Dadurch hält sich der Programmierstress in Grenzen, und der Spaßfaktor bleibt erhalten.

Alle folgenden Programme und Aufbaupläne können Sie auch von dieser Webseite herunterladen: http://fkainka.de/Arduino/Beispiele

Projekte zum Nachbauen

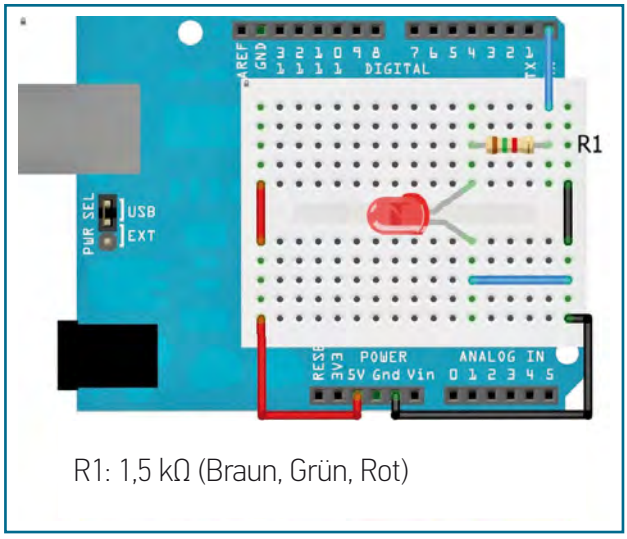
Die erste LED

So berechnen Sie den Widerstand.

Genug vorbereitet! Jetzt geht es endlich los mit dem ersten eigenen Programm: Zunächst schließen Sie eine LED an Pin0 an. Vor die LED gehört natürlich immer ein Vorwiderstand, um den Strom zu begrenzen. Der Pin hat, wenn er HIGH geschaltet ist, einen Pegel von + 5 V. Außerdem liefert er bis zu 40 mA. Also berechnen Sie den Widerstand wie folgt:

Spannung am Widerstand:	U = 5 V – 1,7 V (Spannungsabfall an der LED) = 3,3 V
Strom am Widerstand:	I = erlaubter Strom an der LED = 20 mA
Widerstand:	R = U / I => R = 3,3 V/20 mA = 165 Ω

Das bedeutet also, dass Sie mindestens einen Widerstand von 165 Ω einsetzen sollten, um die LED nicht zu überlasten. Ich habe 1,5 kΩ (Farbringe: Braun, Grün, Rot) verwendet. Damit fließen durch die LED nur 2,2 mA. Das ist aber immer noch ausreichend, damit sie hell leuchtet. Auf dem Bild sieht man, wie die LED auf einem Steckbrett angeschlossen werden könnte.



R1: 1,5 kΩ (Braun, Grün, Rot)

Abb 21: Aufbau der ersten LED

Achten Sie darauf, dass der negative Pol der LED (die Kathode) in Richtung GND zeigt. Sie erkennen die Kathode in den meisten Fällen an folgenden Merkmalen:

Die wichtigsten Merkmale einer Kathode.

Merkmale einer Kathode

- kürzeres Kontaktbeinchen
- Abflachung am Gehäuserand
- größere Kontaktfläche im Inneren der LED

In der Programmierumgebung müssen Sie jetzt schrittweise Folgendes tun:

1. Die grundlegende Programmstruktur anlegen.
2. Den Pin der LED als Ausgang definieren.
3. Den besagten Pin auf HIGH schalten.

Drei Schritte in der Programmierumgebung.

Heraus kommt folgendes Programm:

Programm LED 0

```
setup() {
pinMode (0, OUTPUT);                    //Pin0 als Ausgang definieren
}


void loop() {
digitalWrite(0,HIGH);                   //Pin0 auf HIGH setzen
}
```

Kompilieren und uploaden Sie das Programm. Die LED sollte jetzt leuchten.

Ist dies nicht der Fall, prüfen Sie anhand der folgenden Checkliste:

Checkliste

- Haben Sie den richtigen Pin getroffen?
- Haben Sie die LED richtig herum angeschlossen?
- Haben Sie die LED gegen Masse geschaltet?
- Wurde eine Fehlermeldung angezeigt?
- Haben Sie die Groß- und Kleinschreibung im Programm beachtet?

Wenn alles funktioniert, können Sie noch etwas tun, um das Programm übersichtlicher zu gestalten. Es ist nämlich möglich, Ihren Pins Namen zu geben. Das kann bei komplexeren Programmen sehr hilfreich sein.

Globale Variablen (die man in allen Prozeduren benutzen kann) definiert man grundsätzlich noch vor der Setup-Prozedur.

Hierfür wird eine Variable definiert. Globale Variablen (die man in allen Prozeduren benutzen kann) definiert man grundsätzlich noch vor der Setup-Prozedur. Möchten Sie eine Variable lokal (nur in genau dieser Prozedur) verwenden, müssen Sie sie zu Beginn der Prozedur deklarieren.

Diese Variable ist aber global und steht noch vor der Setup-Prozedur. Die Zeile, die dort eingefügt wird, lautet:

```
int led = 0;
```

Die Variable led.

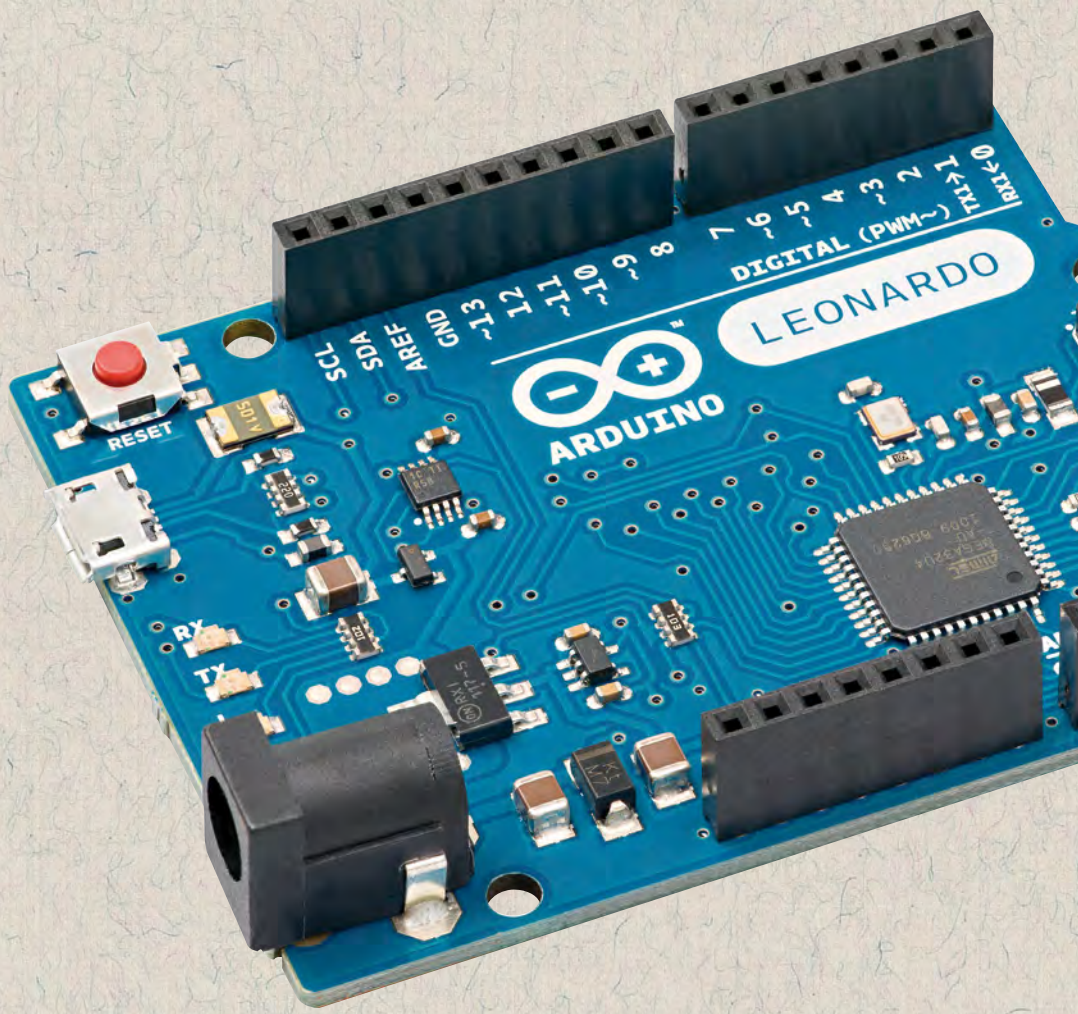
int steht für Integer, einen Variablentyp, der für Ganzzahlen steht. **led** ist der Variablenname, und nach dem Gleichheitszeichen steht der Wert, den Sie dieser Variablen beim Start zuweisen. Der Compiler sorgt nun beim Übersetzen des Programms dafür, dass alle **led**-Variablen durch den entsprechenden Wert ersetzt werden.

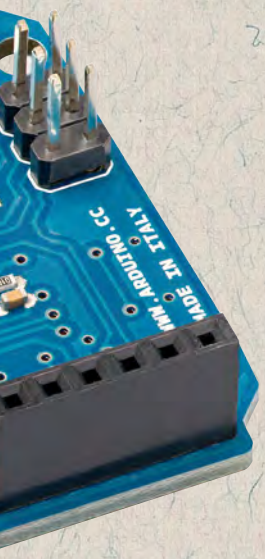
Fabian Kainka

Das Franzis Starterpaket ARDUINO™ LEONARDO

Das Handbuch

Inhalt


SCL
SDA
AREF
GND
~13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
LEONARDO
ARDUINO
RESET
RX
TX



1

SCHNELLSTART

Starthilfe
Drei Beispiele

Starthilfe

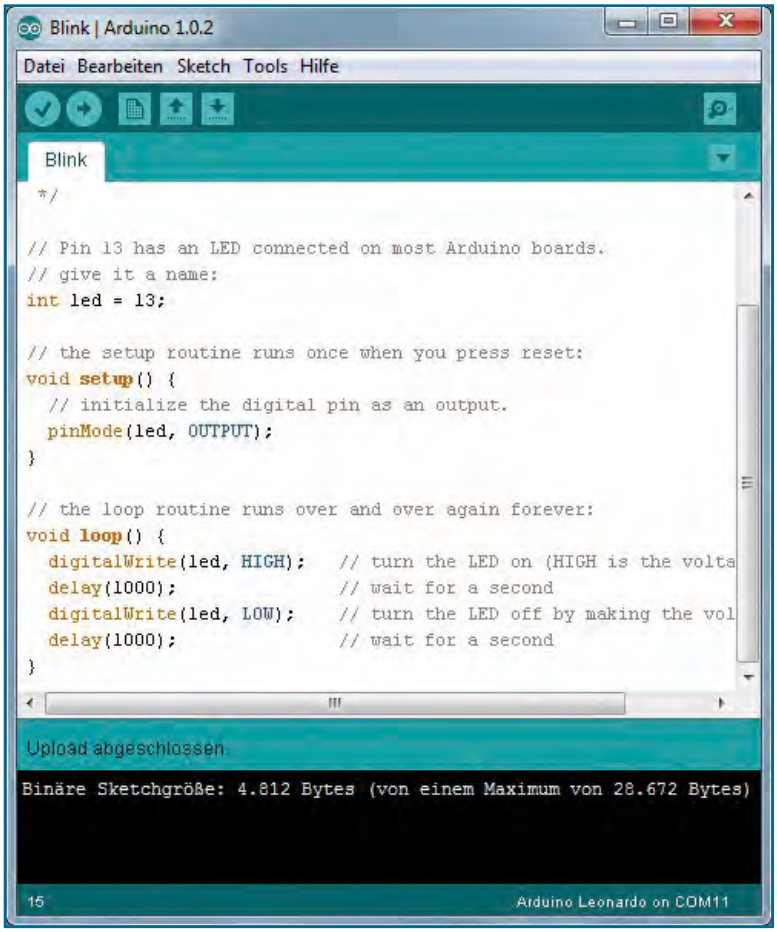


Abb. 1: *Die Programmierumgebung nach dem erfolgreichen Upload*

Möchten Sie jetzt gleich loslegen und ein Programm auf Ihrem Arduino ausprobieren? Wenn ja, finden Sie an dieser Stelle zunächst einmal eine Schnellstartanleitung und ein paar Beispiele, bevor Sie sich den Details zuwenden.

Sollten Sie beim Schnellstart auf Probleme stoßen, die sich daraus ergeben, dass Sie noch nicht so fit in der Materie sind, finden Sie später noch eine ausführlichere Anleitung für die Installation. Aber mit etwas Vorerfahrung reichen die folgenden Schritte für den ersten Erfolg.

So gehen Sie vor

1. Aktuelle Software laden und installieren. Sie finden sie unter http://www.arduino.cc/en/Main/software
2. Controller anschließen.
3. Treiber (nur unter Windows) installieren (Treiber finden Sie im Zip-Ordner).
4. Arduino-Programmierumgebung starten.
5. Unter *Tools > Board* die verwendete Hardware (Arduino Leonardo) auswählen.
6. Unter *Tools > Serieller Port* den entsprechenden COM-Port wählen.
7. Unter *Datei > Beispiele > 01.Basics > Blink* Beispiel laden.
8. Den Upload-Button klicken.
9. Auf die Meldung „Upload abgeschlossen“ warten.
10. Die LED „L“ auf dem Board blinkt – fertig!

Herzlichen Glückwunsch! Auf Ihrem Arduino läuft nun ein funktionsfähiges Programm.

Drei Beispiele

Für die folgenden drei Beispiele benötigen Sie keine zusätzliche Hardware, nur ein kleines Stück Verbindungsdraht. Anhand der Beispiele lässt sich ein erster Eindruck von den Möglichkeiten des Arduino-Controllers gewinnen. Die Beispiele sind in die Arduino-Software integriert und werden nur minimal variiert. Dadurch können Ihnen kaum Fehler unterlaufen, und Sie sammeln erste Erfahrungen mit Ihrem Arduino-Controller.

Tipp: Falls doch etwas schiefgeht, finden Sie Hilfe im Anhang.

Fade

Dieses Beispiel finden Sie unter:
Datei > Beispiele > 01.Basics > Fade
Um dieses Programm zu nutzen, ohne eine LED anzuschließen, nehmen Sie eine kleine Veränderung vor. In Zeile 10 steht folgender Text:

```
int led = 9;          //the pin that the LED is attached to
```

Die Voreinstellung dieses Programms bezieht sich auf Pin9. Die L-LED ist aber an Pin13 angeschlossen. Also machen Sie aus der 9 eine 13:

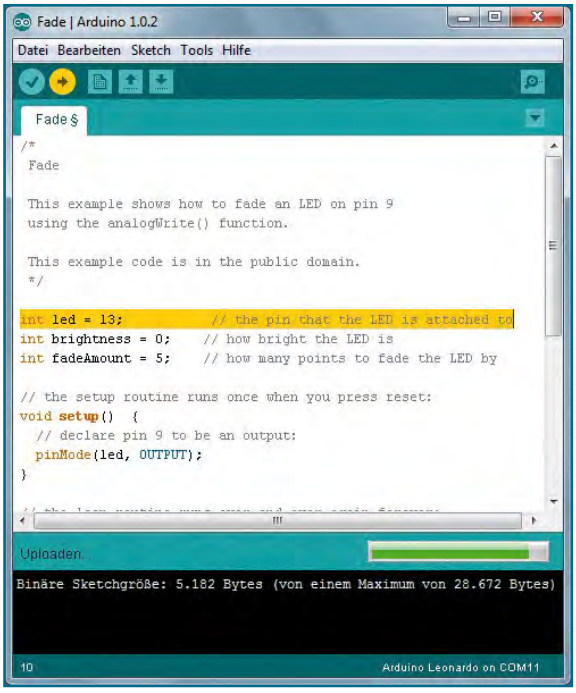


Abb. 2: *In der markierten Zeile sehen Sie die Änderung.*

Nach dem Upload blinkt die LED. Anstatt wie im vorherigen Programm an- oder auszugehen, gewinnt die LED langsam an Helligkeit. Am hellsten Punkt kehrt sich der Verlauf um. Diesen Vorgang nennt man Fade. Mithilfe der PWM wird die mittlere Helligkeit der LED verändert. Mehr zu PWM später.

PC-Hack

Speichern Sie alle offenen Dokumente und schließen Sie alle Downloads ab, bevor Sie das folgende Programm in Ihren Controller laden. Wenn Sie sicher sind, dass keine wichtigen Daten verlorengehen können, öffnen Sie das Programm:
Datei > Beispiele > 09.USB > Keyboard > Keyboard-Logout

In Zeile 33 dieses Programms (siehe Bild) lautet die Voreinstellung OSX. Falls Sie nicht mit einem Mac arbeiten, geben Sie hier Ihr Betriebssystem an. Wenn Sie einen Windows-PC nutzen, tragen Sie statt OSX an dieser Stelle WINDOWS oder die Zahl 1 ein. Wenn Ihr Betriebssystem Linux ist, geben Sie UBUNTU oder die Zahl 2 ein. Anschließend kompilieren Sie das Programm und laden es auf den Controller. Nun benötigen Sie noch ein Stück Schaltdraht. Das eine Ende des Drahts stecken Sie in einen GND-Pin, das andere Ende in den Digital-Pin2 (nicht A2).

Wenn alles korrekt eingegeben wurde, erscheint nun der Login-Screen Ihres Betriebssystems. Der Controller hat sich nach der Programmierung als HID (Human Interface Device = Eingabegerät) angemeldet, genauer gesagt als Tastatur. Mit der Verbindung von Pin2 zu GND haben Sie dem Controller das Signal gegeben, eine Tastenkombination an Ihren Rechner zu schicken. Diese Tastenkombination meldet den aktuellen Benutzer ab und lässt das Anmelde-Fenster erscheinen. Wenn Sie Ihren Controller mit der Reset-Taste neu starten, können Sie den gleichen Trick vollführen. Nebenbei bemerkt: Da für dieses Programm keine Treiber installiert werden müssen, können Sie anderen PC-Nutzern damit natürlich auch einen kleinen Streich spielen.

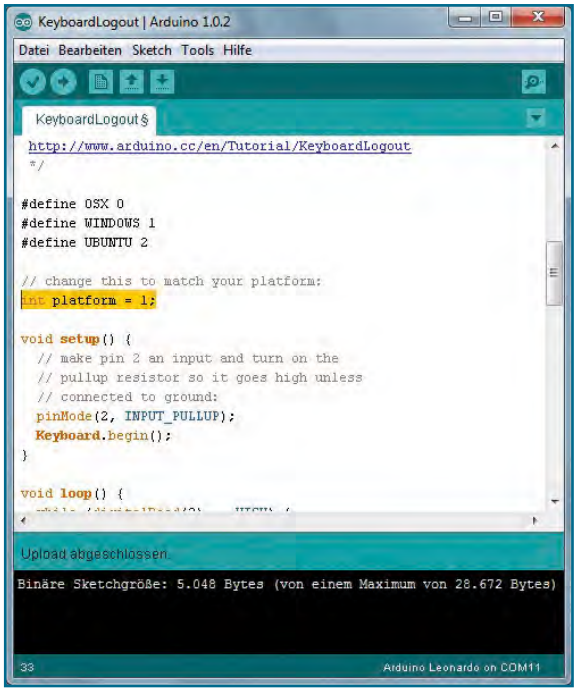


Abb. 3: In der markierten Zeile geben Sie Ihr Betriebssystem an.

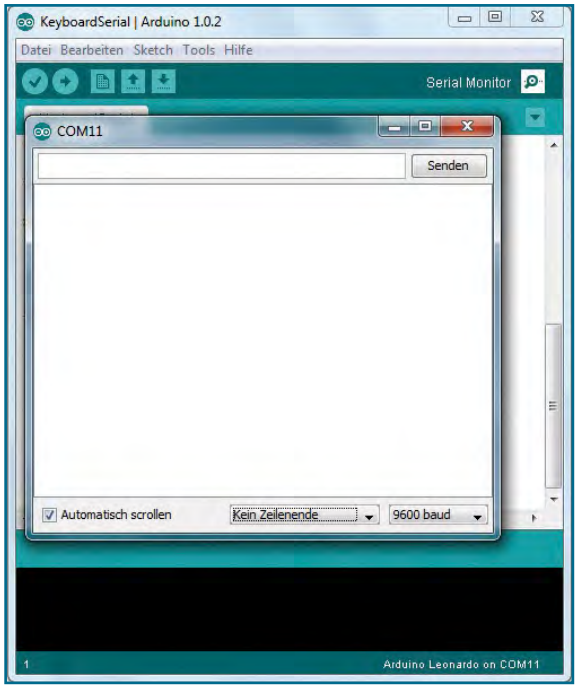


Abb. 4: Das Programm mit geöffnetem Serial-Monitor

Tastatur-Hack

Auch im folgenden Programm gibt sich der Controller wieder als Tastatur aus. Gleichzeitig empfängt er Daten über die serielle Schnittstelle, was für ein interessantes Wechselspiel zwischen Ein- und Ausgabe sorgt. Laden Sie das folgende Programm in Ihren Controller:
Datei > Beispiele > 09.USB > Keyboard > KeyboardSerial

Nun öffnen Sie den „Serial Monitor", den Sie rechts auf der Symbolleiste finden. Als Nächstes öffnet sich ein Fenster, ähnlich dem in Abb. 4. Dies ist das Terminal, mit dem Sie serielle Daten zum Controller senden oder von ihm empfangen. Geben Sie nun in die obere Zeile „a" ein und bestätigen Sie mit der Enter-Taste.

Sie wundern sich vermutlich, denn nun steht „b" in der Zeile. Drücken Sie erneut die Enter-Taste, wird aus „b" plötzlich „c". Dies funktioniert sogar mit einer ganzen Textzeile.

Was passiert mit dem Text? Zunächst wird der Buchstabe nach dem Auslösen der Enter-Taste über die serielle Schnittstelle an den Controller übertragen. Dieser verschiebt das Zeichen um 1, was aus „a" „b" macht. Anschließend sendet der Controller in seiner Funktion als Tastatur das neu berechnete Zeichen zurück. Da sich der Cursor noch in der Textzeile des Serial Monitor befindet, wird das Zeichen dort „eingetippt". Das Ergebnis ist verblüffend. Probieren Sie mal aus, was nach „z" kommt.