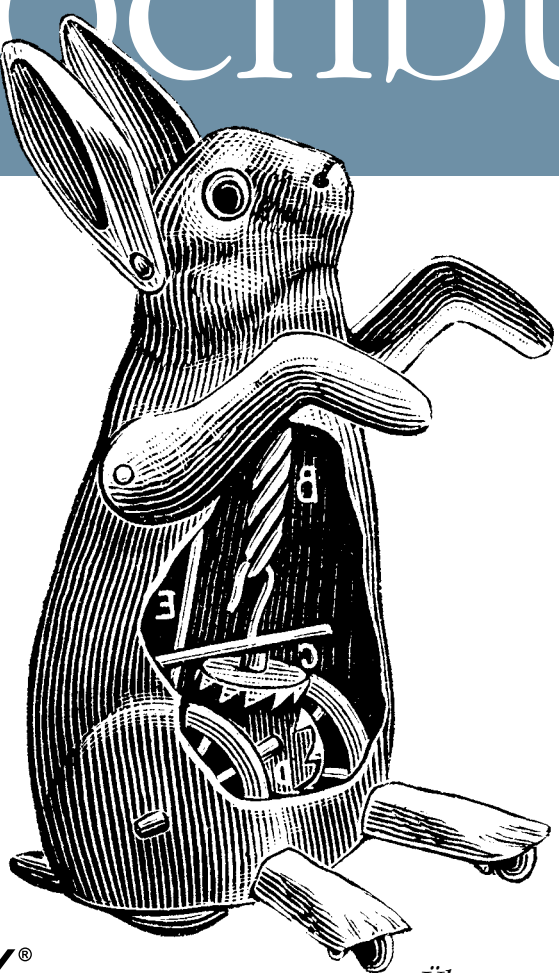
Arduino für Fortgeschrittene

Behandelt Arduino 1.0

Arduino Kochbuch



Michael Magolis

Übersetzung von Peter Klicman

O'REILLY®

Arduino Kochbuch

Michael Margolis

Deutsche Übersetzung von Peter Klicman

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

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

1. Auflage 2012

Die Originalausgabe erschien 2011 unter dem Titel
Arduino Cookbook, Second Edition, im Verlag O'Reilly Media, Inc.

Die Darstellung eines mechanischens Hasens im Zusammenhang mit dem Thema *Arduino* ist ein Warenzeichen von O'Reilly Media, Inc.

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

Lektorat: Volker Bombien
Fachliche Unterstützung: Markus Ulsaß, Hamburg
Korrektorat: Dr. Dorothée Leidig
Satz: Reemers Publishing Services GmbH, Krefeld, www.reemers.de
Umschlaggestaltung: Michael Oreal, Köln
Produktion: Karin Driesen, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; www.koeselbuch.de

ISBN-13 978-3-86899-353-0

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort

Dieses Buch wurde von Michael Margolis, zusammen mit Nick Weldin, geschrieben, um Sie die erstaunlichen Dinge entdecken zu lassen, die man mit Arduino machen kann.

Arduino ist eine Familie von Mikrocontrollern (kleinen Computern) und eine Umgebung zur Software-Entwicklung, die es Ihnen leicht macht, Programme (sog. *Sketches*) zu entwickeln, die mit der physikalischen Welt interagieren. Mit Arduino entwickelte Dinge können auf Berührungen, Töne, Wärme und Licht reagieren. Diese Technik, auch *physical computing* genannt, wird in den unterschiedlichsten Dingen, vom iPhone bis zur Automobilelektronik verwendet. Arduino ermöglicht es jedermann – auch Menschen ohne Programmier- oder Elektronikkenntnisse –, diese mächtige und komplexe Technik zu nutzen.

Leserkreis

Im Gegensatz zu den meisten technischen Kochbüchern wird keinerlei Erfahrung mit Soft- und Hardware vorausgesetzt. Dieses Buch richtet sich an Leser, die Computertechnik nutzen wollen, um mit ihrer Umgebung zu interagieren. Es ist für Leute gedacht, die eine schnelle Lösung für ihre Hard- und Softwareprobleme suchen. Die Rezepte bieten die Informationen, die Sie benötigen, um eine große Bandbreite von Aufgaben zu erledigen. Sie enthalten auch Details, die Ihnen dabei helfen, Lösungen an Ihre Bedürfnisse anzupassen. Ein auf 600 Seiten beschränktes Buch kann nicht den allgemeinen theoretischen Hintergrund vermitteln. Daher finden Sie überall im Buch Links auf externe Referenzen. Im Abschnitt »Was ausgelassen wurde« auf Seite XIV finden sich einige allgemeine Referenzen für diejenigen ohne Programmier- und Elektronikkenntnisse.

Wenn Sie keine Programmiererfahrung haben – vielleicht haben Sie eine gute Idee für ein interaktives Projekt, verfügen aber nicht über das notwendige Wissen, um es bauen zu können –, hilft Ihnen dieses Buch, das zu lernen, was Sie zum Schreiben funktionierender Programme brauchen. Dazu verwenden wir Beispiele, die über 200 gängige Aufgaben behandeln.

Wenn Sie über Programmiererfahrung verfügen, aber nicht mit Arduino vertraut sind, sorgt das Buch für ein schnelleres produktives Arbeiten, indem es demonstriert, wie man Arduino-spezifische Fähigkeiten in ein Projekt integriert.

Wenn Sie bereits mit Arduino vertraut sind, werden Sie den Inhalt nützlich finden, der Ihnen neue Techniken anhand praktischer Beispiele vermittelt. Das hilft Ihnen bei komplexeren Projekten, indem es zeigt, wie man Probleme mit Hilfe Ihnen möglicherweise noch nicht bekannter Techniken löst.

Erfahrene C/C++-Programmierer finden Beispiele für den Einsatz der auf niedriger Ebene angesiedelten AVR-Ressourcen (Interrupts, Timer, I2C, Ethernet etc.), die bei der Entwicklung von Anwendungen mit der Arduino-Umgebung helfen.

Organisation

Das Buch enthält Informationen, die ein breites Spektrum der Arduino-Fähigkeiten abdecken. Sie reichen von grundlegenden Konzepten und gängigen Aufgaben bis hin zu fortgeschrittenen Techniken. Jede Technik wird in einem Rezept erläutert, das zeigt, wie man eine bestimmte Fähigkeit implementiert. Sie müssen den Inhalt nicht der Reihe nach lesen. Nutzt ein Rezept eine Technik, die in einem anderen Rezept behandelt wird, finden Sie eine Referenz auf das andere Rezept, d.h., die Details werden nicht an mehreren Stellen wiederholt.

Kapitel 1 führt in die Arduino-Umgebung ein und erläutert die Arduino-Installation.

Die nächsten Kapitel führen in die Arduino-Software-Entwicklung ein. Kapitel 2, behandelt grundlegende Softwarekonzepte und Aufgaben. Kapitel 3 zeigt, wie man die gängigsten mathematischen Funktionen verwendet.

Kapitel 4 beschreibt, wie man den Arduino mit Ihrem Computer und anderen Geräten verbindet und mit ihnen kommuniziert. Seriell ist die für Arduino gängigste Methode der Ein- und Ausgabe, die im gesamten Buch von vielen Rezepten genutzt wird.

Kapitel 5 führt eine Reihe grundlegender Techniken zum Lesen digitaler und analoger Signale ein. Darauf aufbauend zeigt Kapitel 6, wie man Bauelemente nutzt, die es Arduino ermöglichen, Berührungen, Töne, Positionen, Wärme und Licht wahrzunehmen.

Kapitel 7 behandelt die Steuerung von Licht. Die Rezepte zeigen, wie man ein oder mehrere LEDs einschaltet und Helligkeit und Farbe kontrolliert. Dieses Kapitel erläutert, wie man Strichskalen und numerische LED-Displays ansteuert und wie man Muster und Animationen mit LED-Arrays erzeugt. Für Einsteiger enthält das Kapitel zusätzlich eine allgemeine Einführung in die digitale und analoge Ausgabe.

Kapitel 8 erklärt, wie man mit dem Arduino Dinge mittels Motoren bewegen kann. Es werden unterschiedlichste Motortypen behandelt: Spulen, Servomotoren, Gleichstrom- und Schrittmotoren.

Kapitel 9 zeigt, wie man mit de Arduino Töne über ein Ausgabegerät wie einen Lautsprecher erzeugt. Es behandelt einfache Töne und Melodien und das Abspeichern von WAV-Dateien und MIDI.

Kapitel 10 beschreibt Techniken, mit denen Sie mit nahezu jedem Gerät interagieren können, das irgendeine Form von Fernbedienung nutzt: Fernseher, Audiogeräte, Kameras, Garagentore, Haushaltsgeräte und Spielzeug. Es baut auf den Techniken zur Verbindung des Arduino mit anderen Bauelementen und Modulen auf.

Kapitel 11 behandelt die Verbindung mit Text- und LC-Displays. Es zeigt, wie man diese Geräte anbindet, um Text auszugeben, Wörter scrollt und hervorhebt und spezielle Symbole und Zeichen erzeugt.

Kapitel 12 behandelt die in Arduino fest integrierten zeitbezogenen Funktionen und stellt zusätzliche Techniken für Zeitverzögerungen, zur Zeitmessung und reale Zeit- und Datumsangaben vor.

Kapitel 13 behandelt die I2C-(Inter-Integrated Circuit-) und SPI-(Serial Peripheral Interface-)Standards. Diese Standards bieten eine einfache Möglichkeit, Informationen digital zwischen Sensoren und dem Arduino zu übertragen. Das Kapitel zeigt, wie man I2C und SPI nutzt, um gängige Bauelemente anzubinden. Es zeigt auch, wie man zwei oder mehr Arduino-Boards für Multiboard-Anwendungen über I2C miteinander verbindet.

Kapitel 14 behandelt die drahtlose Kommunikation mittels XBee und anderen Wireless-Modulen. Die Beispiele reichen vom einfachen Drahtlos-Ersatz für serielle Ports bis hin zu Mesh-Netzwerken, die mehrere Boards mit mehreren Sensoren verbinden.

Kapitel 15 beschreibt die vielen Möglichkeiten, wie Sie Arduino fürs Internet benutzen können. Es enthält Beispiele, die zeigen, wie man Web-Clients und -Server erstellt und nutzt und erläutert die gebräuchlichen Internet-Protokolle.

Arduino Softwarebibliotheken sind der übliche Ansatz, um die Arduino-Umgebung um zusätzliche Funktionen zu erweitern. Kapitel 16 erläutert, wie man Softwarebibliotheken nutzt und modifiziert. Es enthält auch eine Anleitung zur Entwicklung eigener Bibliotheken.

Kapitel 16 behandelt die fortgeschrittenen Programmiertechniken, und die Inhalte sind etwas technischer als die Rezepte in den vorherigen Kapiteln, weil sie Dinge abdecken, die ansonsten von freundlichen Arduino-Kumpels erledigt werden.

Die Rezepte aus diesem Kapitel können dazu eingesetzt werden, um einen Sketch effizienter zu schreiben, sie können Ihnen dabei helfen, die Performance zu verbessern und den Code schlanker zu schreiben.

Die Anhänge sowie Kapitel 17 und 18 wurden nicht übersetzt. Sie stehen in Englisch als Download auf unserer Webseite zur Verfügung.

Anhang A enthält eine Übersicht der im Buch verwendeten Komponenten.

Anhang B erklärt, wie man Schaltpläne und Datenblätter verwendet.

Anhang C bietet eine kurze Einführung in die Verwendung von Steckbrettern, den Anschluss und Einsatz externer Stromversorgungen und Batterien sowie der Nutzung von Kondensatoren zur Entstörung.

Anhang D enthält Tipps zur Behebung von Compiler- und Laufzeitproblemen.

Anhang E behandelt Probleme mit elektronischen Schaltungen.

Anhang F enthält Tabellen mit den Funktionen der einzelnen Pins bei Standard-Arduino-Boards.

Anhang G enthält Tabellen mit den ASCII-Zeichen.

Anhang H erklärt, wie man Code für ältere Releases anpasst, damit er unter Arduino 1.0 korrekt läuft.

Was ausgelassen wurde

Das Buch bietet nicht genug Platz, um Elektronik in Theorie und Praxis zu erläutern, auch wenn Anleitungen für den Bau der in den Rezepten verwendeten Schaltungen gegeben werden. Für genauere Informationen sei der Leser auf das im Internet zahlreich vorhandene Material oder auf die folgenden Bücher verwiesen:

- *Make: Elektronik* (ISBN 978-3-89721-601-3) von Charles Platt
- *Arduino für Einsteiger* (ISBN 978-3-86899-232-8) von Massimo Banzi
- *Die elektronische Welt mit Arduino entdecken* (ISBN 978-3-89721-319-7) von Erik Bartmann
- *Making Things Talk* (ISBN 978-3-86899-162-8) von Tom Igoe

Dieses Kochbuch erklärt, wie man Code schreibt, der bestimmte Arbeiten erledigt, es ist jedoch keine Einführung in die Programmierung. Wichtige Programmierkonzepte werden kurz erklärt, doch der Platz reicht nicht aus, um auf die Details einzugehen. Wenn Sie mehr über die Programmierung lernen wollen, sei auf das Internet verwiesen.

Nicht in die deutsche Übersetzung aufgenommen wurden die Kurzkapitel »Advanced Coding and Memory Handling«, »Using the Controller Chip Hardware« sowie die Anhänge, um einen akzeptablen Verkaufspreis für das Buch zu gewährleisten.

Diese Kapitel sind als Originalkapitel in PDF-Form von unserer Webseite www.oreilly.de downloadbar.

Code-Stil (Über den Code)

Für das gesamte Buch wurde der Code maßgeschneidert, um das Thema des jeweiligen Rezepts ganz deutlich zu machen. Infolgedessen wurden gängige Kürzel vermieden, insbesondere in den frühen Kapiteln. Erfahrene C-Programmierer verwenden häufig mächtige, aber sehr knappe Ausdrücke, die für Anfänger etwas schwer verständlich sind. Zum Beispiel werden in den frühen Kapiteln Variablen mit expliziten Ausdrücken inkrementiert, die für Nichtprogrammierer leicht verständlich sind:

```
result = result + 1; // Zähler inkrementieren
```

und nicht in der von erfahrenen Programmierern üblicherweise genutzten Kurzform, die das Gleiche macht:

```
result++; // Inkrement mittels Postinkrement-Operator
```

Es steht Ihnen natürlich frei, den von Ihnen bevorzugte Stil zu verwenden. Anfängern sei versichert, dass die Kurzform keinerlei Vorteil bei Performance oder Codegröße bringt.

Einige Programmierausdrücke sind so gängig, dass wir ihre Kurzform verwenden. Zum Beispiel werden Schleifen immer wie folgt geschrieben:

```
for(int i=0; i < 4; i++)
```

was mit folgendem identisch ist:

```
int i;
for(i=0; i < 4; i = i+1)
```

Weitere Details zu diesen und anderen im Buch verwendeten Ausdrücken finden Sie in Kapitel 2.

Gute Programmierpraxis verlangt, dass die verwendeten Werte gültig sind, d.h., dass man sie prüft, bevor man sie in Berechnungen nutzt. Damit sich der Code aber auf das eigentliche Rezept konzentriert, haben wir nur sehr wenige Fehlerprüfungen eingefügt.

Arduino-Version

Diese Ausgabe wurde für Arduino 1.0 aktualisiert. Der gesamte Code wurde mit dem neuesten Arduino 1.0 Release Candidate getestet, der zur Drucklegung verfügbar war (RC2). Der Download-Code für diese Ausgabe wird bei Bedarf online aktualisiert, um die finale Release 1.0 zu unterstützen. Besuchen Sie also die Buch-Website (*http://shop.oreilly.com/product/0636920022244.do*), um den neuesten Code zu erhalten. Der Download enthält eine Datei namens *changelog.txt*, die den Code beschreibt, der sich von der gedruckten Version unterscheidet.

Obwohl viele Sketches mit früheren Arduino-Releases laufen, müssen Sie die Endung von *.ino* in *.pde* ändern, um den Sketch in eine Pre-1.0-IDE zu laden. Wenn Sie nicht auf Arduino 1.0 migriert sind und gute Gründe haben, bei einer älteren Release zu bleiben, können Sie den Beispielcode der ersten Ausgabe nutzen (verfügbar unter *http://shop.oreilly.com/product/9780596802486.do*), der mit den Releases 0018 bis 0022 getestet wurde. Beachten Sie, dass viele Rezepte der zweiten Ausgabe erweitert wurden, weshalb wir ein Upgrade auf Arduino 1.0 empfehlen. Hilfe zur Migration älteren Codes finden Sie in Anhang H (steht als Download bereit).

Dort finden Sie auch einen Link zum Fehlerverzeichnis. Das Fehlerverzeichnis gibt Ihnen die Möglichkeit, uns über (Druck-)Fehler und andere Probleme mit dem Buch zu informieren. Fehler sind auf der Seite sofort sichtbar und werden von uns bestätigt, sobald wir sie überprüft haben. O'Reilly kann die Fehler in zukünftigen Auflagen und auf Safari korrigieren.

Wenn Sie Probleme haben, Beispiele ans Laufen zu bekommen, überprüfen Sie in der Datei *changelog.txt* des aktuellsten Downloads, ob der Sketch aktualisiert wurde. Falls das Ihr Problem nicht löst, sehen Sie sich Anhang D (steht als Download bereit) an, das die Behebung von Softwareproblemen behandelt. Falls Sie mehr Hilfe benötigen, ist das Arduino-Forum ein guter Ort, um Fragen zu stellen: *http://www.arduino.cc*.

Wenn Sie dieses Buch mögen – oder auch nicht –, sollten es die Leute unbedingt erfahren. Amazon-Rezensionen sind eine beliebte Möglichkeit, Ihre Zufriedenheit und andere Kommentare mit anderen zu teilen. Sie können auch auf der O'Reilly-Site zu diesem Buch einen Kommentar hinterlassen.

Verwendete Konventionen

In diesem Buch werden die folgenden typographischen Konventionen verwendet:

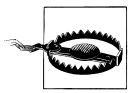
Kursivschrift
: wird für Pfad-, Datei- und Programmnamen, Internetadressen, Domainnamen und URLs verwendet, sowie für neue Begriffe, wenn sie zum ersten Mal im Text auftauchen.

`Nichtproportionalschrift`
: wird für Kommandozeilen und Optionen verwendet, die Sie wortwörtlich eingeben müssen. Ebenso bei Namen und Schlüsselwörtern in Programmen, einschließlich Methoden-, Variablen- und Klassennamen sowie HTML-Tags.

`Nichtproportionalschrift fett`
: wird für Hervorhebungen im Programmcode verwendet.

`Nichtproportionalschrift kursiv`
: wird für Text verwendet, der durch Benutzereingaben ersetzt werden muss.

Zeigt einen Tipp, eine Empfehlung oder einen allgemeinen Hinweis an.



Zeigt eine Warnung an.

Verwendung der Codebeispiele

Dieses Buch soll Ihnen bei der Arbeit mit Arduino helfen. Den Code, den wir hier zeigen, dürfen Sie generell in Ihren Programmen und Dokumentationen verwenden. Sie brauchen uns nicht um Genehmigung zu bitten, sofern Sie nicht große Teile des Codes reproduzieren. Wenn Sie zum Beispiel ein Programm schreiben, das mehrere Codeabschnitte aus diesem Buch verwendet, brauchen Sie unser Einverständnis nicht. Doch wenn Sie eine CD-ROM mit Codebeispielen aus O'Reilly-Büchern verkaufen wollen, müssen Sie sehr

wohl eine Erlaubnis einholen. Eine Frage mit einem Zitat und einem Codebeispiel aus diesem Buch zu beantworten, erfordert keine Erlaubnis, aber es ist nicht ohne Weiteres gestattet, große Teile unseres Textes oder Codes in eine eigene Produktdokumentation aufzunehmen.

Wir freuen uns über eine Quellenangabe, verlangen sie aber nicht zwingend. Zu einer Quellenangabe gehören normalerweise der Titel, der Autor, der Verlag und die ISBN, zum Beispiel: »*Arduino Kochbuch*, von Michael Margolis mit Nick Weldin (O'Reilly). Copyright 2012 Michael Margolis, Nicholas Weldin, 978-86899-353-0.«

Wenn Sie das Gefühl haben, dass Ihr Einsatz unserer Codebeispiele über die Grenzen des Erlaubten hinausgeht, schreiben Sie uns bitte eine E-Mail an *permissions@oreilly.com*.

Danksagungen

Nick Weldins Beitrag war für die Fertigstellung dieses Buches von unschätzbarem Wert. Das Buch war zu 90 Prozent fertig, als Nick an Bord kam – und ohne sein Können und seinen Enthusiasmus wären es wohl immer noch nur 90 Prozent. Seine Erfahrung mit Arduino-Workshops für die unterschiedlichsten Anwender macht die Ratschläge in diesem Buch für unseren breiten Leserkreis nutzbar. Danke, Nick, für dein Wissen und dein geniales, kollaboratives Wesen.

Simon St. Laurent war der Lektor bei O'Reilly, der als Erster Interesse an diesem Buch bekundet hat. Und letztlich war er auch derjenige, der es zusammengehalten hat. Seine Unterstützung und Aufmunterung hielten uns bei der Stange, während wir die Unmengen an Material durchgingen, die nötig waren, um dem Thema Genüge zu tun.

Brian Jepson half mir dabei, mit dem Schreiben dieses Buches anzufangen. Sein umfassendes Wissen in Bezug auf Arduino und sein Bemühen, Technik mit einfachen Worten zu vermitteln, setzten einen hohen Standard. Er war die ideale führende Hand, um dieses Buch zu formen und Technik für die Leser wirklich zugänglich zu machen. Wir sind Brian auch für den XBee-Inhalt in Kapitel 14 dankbar.

Brian Jepson und Shawn Wallace waren die Betreuer für diese zweite Ausgabe und lieferten wertvolle Hinweise zur Verbesserung der Genauigkeit und Klarheit des Inhalts.

Audrey Doyle arbeitete unermüdlich daran, Schreibfehler und grammatikalische Fehler aus dem ursprünglichen Manuskript zu tilgen und die allzu verwickelten Ausdrücke zu entwirren.

Philip Lindsay arbeitete in der ersten Auflage am Inhalt von Kapitel 15 mit. Adrian McEwen, der führende Entwickler von zahlreichen Ethernet-Erweiterungen für Arduino 1.0, steuerte wertvolle Hinweise für dieses Kapitel bei, um alle Release-Neuerungen zu erfassen.

Mikal Hart schrieb die Rezepte zu GPS und der seriellen (Software-)Schnittstelle. Mikal war dafür die natürliche Wahl – nicht nur, weil er die Bibliotheken geschrieben hat, sondern auch weil er ein Arduino-Enthusiast ist, mit dem die Zusammenarbeit ein Vergnügen ist.

Arduino wird durch die Kreativität des Arduino-Kernentwicklerteams möglich: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino und David Mellis. Im Namen aller Arduino-Nutzer möchte ich unseren Dank für ihre Bemühungen ausdrücken, diese faszinierende Technik einfach nutzbar zu machen, aber auch für ihren Großmut, sie frei zugänglich zu machen.

Ein besonderer Dank geht an Alexandra Deschamps-Sonsino, deren Tinker London Workshops wichtige Erkenntnisse über die Bedürfnisse der Benutzer lieferten. Dank auch an Peter Knight, der alle Arten cleverer Arduino-Lösungen sowie die Basis für eine Reihe von Rezepten in diesem Buch bereitgestellt hat .

Im Namen aller, die von Benutzern beigesteuerte Arduino-Bibliotheken heruntergeladen haben, möchte ich den Autoren danken, die großzügig ihr Wissen geteilt haben.

Die Verfügbarkeit eines großen Spektrums an Hardware macht Arduino so spannend – Dank dafür gebührt den Anbietern, die eine große Menge toller Bauelemente vorhalten und unterstützen. Die nachfolgenden Firmen haben die in diesem Buch verwendete Hardware bereitgestellt: SparkFun, Maker Shed, Gravitech und NKC Electronics. Weitere hilfreiche Anbieter waren Modern Device, Liquidware, Adafruit, MakerBot Industries, Mindkits, Oomlout und SK Pang.

Nick möchte allen danken, die in Tinker London involviert waren, insbesondere Alexandra, Peter, Brock Craft, Daniel Soltis und all den Leuten, die über Jahre bei den Workshops geholfen haben.

Nicks abschließender Dank gilt seiner Familie: Jeanie, Emily und Finn, die es ihm erlaubten, diese Sache während der Sommerferien durchzuziehen (und natürlich viel länger dauerte, als sie ursprünglich dachten), sowie an seine Eltern Frank und Eva, die ihn dazu erzogen, Dinge auseinanderzunehmen.

Zu guter Letzt möchte ich folgenden Leuten danken:

Joshua Noble, der mich O'Reilly vorstellte. Sein Buch *Programming Interactivity (http://oreilly.com/catalog/9780596154158/)* sei all denen wärmstens empfohlen, die ihr Wissen um Interaktivität erweitern wollen.

Robert Lacy-Thompson, der mir sehr früh bei der ersten Auflage seine Hilfe anbot.

Mark Margolis für seine Unterstützung und Hilfe als Diskussionspartner bei der Konzeption und Entwicklung dieses Buches.

Ich danke meinen Eltern, die mir halfen zu erkennen, dass die kreativen Künste und Technik keine distinktiven Entitäten sind und dass sie zu außergewöhnlichen Ergebnissen führen können, wenn man sie miteinander kombiniert.

Und schließlich wäre dieses Buch ohne die Unterstützung meiner Frau, Barbara Faden, weder begonnen noch fertiggestellt worden. Mein aufrichtiger Dank gilt ihrer Motivation, ihrem sorgfältigen Lesen des Manuskripts und ihre Beiträge dazu.

Hinweise zur Neuauflage

Die Neuauflage dieses Buchs folgt recht schnell auf die erste (die Erstauflage wurde nicht ins Deutsche übersetzt). Das wurde durch die Veröffentlichung von Arduino 1.0 nötig. Das genannte Ziel von 1.0 besteht darin, signifikante Änderungen einzuführen, die den Weg für zukünftige Verbesserungen ebnen. Leider funktioniert damit einiger Code nicht nicht mehr, der für ältere Software geschrieben wurde. Das hat dazu geführt, dass Code in vielen Kapiteln des Buches geändert werden musste. Die meisten Änderungen finden sich in Kapitel 15 und Kapitel 13, doch alle Rezepte dieser Ausgabe wurden auf 1.0 migriert und viele wurden dahingehend aktualisiert, dass sie neue Features dieser Release nutzen. Wenn Sie eine Release vor Arduino 1.0 nutzen, können Sie den Code der ersten Auflage des Buches herunterladen.

Anhang H (steht als Download bereit) wurde hinzugefügt, um die Änderungen zu beschreiben, die mit Arduino Release 1.0 eingeführt wurden. Er erläutert, wie man älteren Code an Arduino 1.0 anpasst.

Rezepte für nicht mehr weit verbreitete Bauelemente wurden für aktuelle Bauteile aktualisiert, und einige neue Sensoren und Drahtlosgeräte wurden hinzugefügt.

An die O'Reilly-Site gepostete Fehler wurden behoben. Wir danken den Lesern, die sich die Zeit genommen haben, uns darüber zu informieren.

Wir denken, dass Ihnen die Verbesserungen an Arduino 1.0 und an dieser Auflage des *Arduino Kochbuchs* gefallen werden. Die erste Auflage ist gut angekommen. Die konstruktive Kritik teilte sich zwischen Leuten auf, die es technischer haben wollten, und Leuten, die es sich weniger technisch wünschten. Bei einem Buch, bei dem wir auf nur ca. 600 Seiten beschränkt sind (damit es bezahl- und tragbar bleibt), scheint das für ein gutes Gleichgewicht zu sprechen.

KAPITEL 1

Erste Schritte

1.0 Einführung

Die Arduino-Umgebung wurde entworfen, um von Anfängern einfach genutzt werden zu können, die mit Software oder Elektronik keine Erfahrung haben. Mit Arduino können Sie Objekte entwickeln, die auf Licht, Töne und Bewegung reagieren oder sie kontrollieren. Arduino wurde für den Bau einer Vielzahl faszinierender Dinge verwendet, darunter Musikinstrumente, Roboter, Lichtskulpturen, Spiele, interaktive Möbel und sogar interaktive Kleidung.

Wenn Sie kein Einsteiger sind, können Sie gleich mit den Rezepten weitermachen, die Sie interessieren.

Arduino wird auf der ganzen Welt in vielen Bildungsprogrammen genutzt, insbesondere von Designern und Künstlern, die auf einfache Weise Prototypen herstellen wollen, ohne allzu tief in die technischen Details ihrer Schöpfungen einsteigen zu müssen. Da sie entworfen wurde, um von nicht technisch versierten Menschen genutzt zu werden, enthält die Software viele Codebeispiele, die demonstrieren, wie man die verschiedenen Fähigkeiten der Arduino-Boards nutzt.

Obwohl sie einfach zu nutzen ist, arbeitet die Arduino zugrunde liegende Hardware mit der gleichen »Perfektion«, die Ingenieure für den Aufbau eingebetteter Systeme nutzen. Für Leute, die bereits mit Mikrocontrollern gearbeitet haben, ist Arduino aufgrund der agilen Entwicklungsmöglichkeiten und der Möglichkeit zur schnellen Implementierung von Ideen ebenfalls interessant.

Arduino ist für seine Hardware bekannt, doch man benötigt auch Software, um diese Hardware programmieren zu können. Sowohl die Hardware als auch die Software wird »Arduino« genannt. Diese Kombination ermöglicht die Entwicklung von Projekten, die die physikalische Welt wahrnehmen und steuern können. Die Software ist frei, Open Source und plattformübergreifend. Die Boards kann man kostengünstig kaufen oder selbst zusammenbauen (die Hardware-Designs sind ebenfalls Open Source). Darüber hinaus gibt es eine aktive und unterstützende Arduino-Community, die weltweit über

die Arduino-Foren und das Wiki (bekannt als Arduino Playground) zugänglich ist. Die Foren und das Wiki bieten Beispiele für Projekte und Problemlösungen. Sie bieten Hilfe und Inspiration, wenn Sie Ihr eigenes Projekt vorantreiben wollen.

Die Rezepte in diesem Kapitel ermöglichen Ihnen den Einstieg. Sie zeigen Ihnen, wie man die Entwicklungsumgebung einrichtet und wie man einen Beispiel-Sketch kompiliert und ausführt.

Der Quellcode mit den Computer-Instruktionen zur Steuerung von Arduino-Funktionen wird in der Arduino-Community üblicherweise als *Sketch* bezeichnet. Das Wort *Sketch* wird im gesamten Buch für Arduino-Programmcode verwendet.

Der mit Arduino mitgelieferte Blink-Sketch ist ein Beispiel für die Rezepte in diesem Kapitel, auch wenn das letzte Rezept des Kapitels etwas weitergeht. Es lässt nicht nur die auf dem Board vorhandene LED blinken, sondern fügt noch Sound hinzu und liest Eingaben über zusätzliche Hardware ein. Kapitel 2 zeigt, wie man einen Sketch für Arduino strukturiert und führt in die Programmierung ein.

Wenn Sie mit den Arduino-Grundlagen bereits vertraut sind, können Sie mit den nachfolgenden Kapiteln weitermachen. Als Arduino-Einsteiger macht sich das Durcharbeiten dieser frühen Rezepte später mit besseren Ergebnissen bezahlt.

Arduino-Software

Software- Programme, sog. *Sketches*, werden auf einem Computer mit Hilfe der Arduino-Entwicklungsumgebung (Integrated Development Environment, kurz IDE) geschrieben. Die IDE ermöglicht es Ihnen, Code zu schreiben und zu bearbeiten und diesen Code dann in Instruktionen umzuwandeln, die die Arduino-Hardware versteht. Die IDE überträgt diese Instruktionen auch auf das Arduino-Board. Diesen Prozess bezeichnet man als *Hochladen* (engl. Uploading).

Arduino-Hardware

Auf dem Arduino-Board wird der von Ihnen geschriebene Code ausgeführt. Das Board selbst kann nur auf Strom reagieren und ihn steuern, weshalb spezielle Komponenten angeschlossen sind, die die Interaktion mit der realen Welt ermöglichen. Diese Komponenten können Sensoren sein, die bestimmte Aspekte der physikalischen Welt in Strom umwandeln, die das Board verarbeiten kann. Es können aber auch sog. Aktuatoren sein, die Strom vom Board erhalten und ihn in etwas umwandeln, was die Welt verändert. Beispiele für Sensoren sind Schalter, Beschleunigungsmesser und Ultraschall-Abstandssensoren. Aktuatoren sind Dinge wie Lampen und LEDs, Lautsprecher, Motoren und Displays.

Es gibt eine Vielzahl offizieller Boards, die mit der Arduino-Software verwendet werden können, sowie ein breites Spektrum an Arduino-kompatiblen Boards, die von Mitgliedern der Community hergestellt werden.

Die beliebtesten Boards enthalten einen USB-Stecker, der die Stromversorgung übernimmt und die Upload-Verbindung für ihre Software herstellt. Abbildung 1-1 zeigt ein einfaches Board, mit dem viele Leute anfangen: das Arduino Uno.

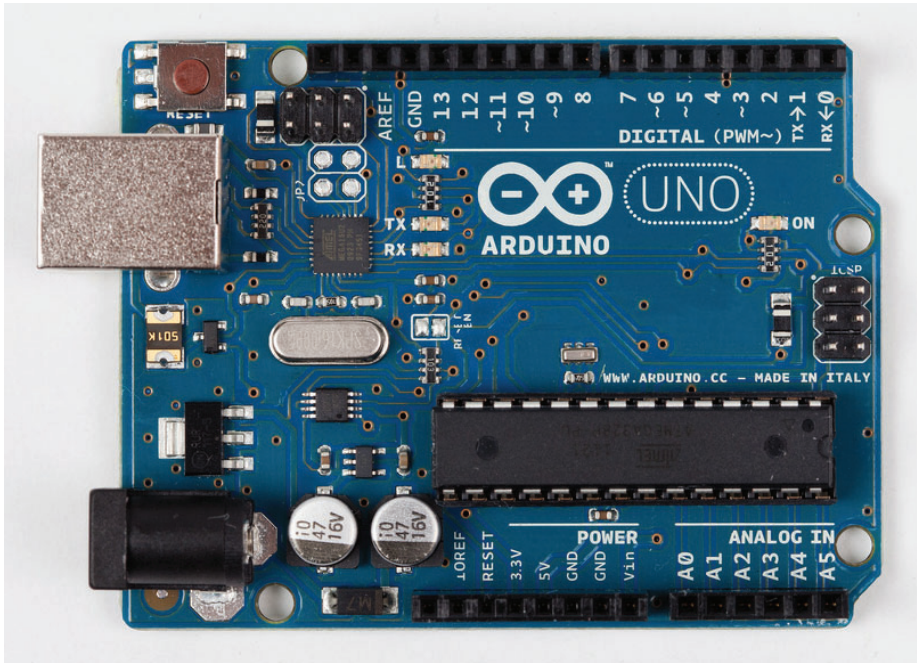


Abbildung 1-1: Einfaches Board: das Arduino Uno. Photo mit freundlicher Genehmigung von todo.to.it.

Das Arduino Uno besitzt einen zweiten Mikrocontroller, der die gesamte USB-Kommunikation übernimmt. Der kleine SMD-Chip (ein ATmega8U2) ist nahe des USB-Steckers zu finden. Dieser Chip kann separat programmiert werden, so dass das Board unterschiedliche USB-Geräte emulieren kann (ein Beispiel finden Sie in Rezept 17.14). Das Arduino Leonardo ersetzt die ATmega8U2- und ATmega328-Controller durch einen einzelnen ATmega32u4-Chip, der das USB-Protokoll softwaremäßig emuliert. Die Arduino-kompatiblen Teensy- und Teensy+-Boards von PJRC (*http://www.pjrc.com/teensy/*) können ebenfalls USB-Geräte emulieren. Ältere Boards (und die meisten Arduino-kompatiblen Boards) verwenden einen Chip von FTDI, der eine Hardware-USB-Lösung bietet, mit der man die Verbindung mit dem seriellen Port des Computers herstellen kann.

Sie können Boards kaufen, die so klein wie eine Briefmarke sind, etwa das Arduino Mini und das Pro Mini. Größere Boards (wie das Arduino Mega) bieten mehr Anschlüsse und leistungsfähigere Prozessoren. Es gibt auch Boards für spezielle Anwendungen, etwa das

LilyPad, das man in Kleidung integrieren kann (»Wearable«-Anwendungen), das Fio für Wireless-Projekte, oder das Arduino Pro für Embedded-Anwendungen (eigenständige, häufig batteriebetriebene Projekte).

Jüngstes Mitglied ist das Arduino ADK, das über einen USB-Host-Sockel verfügt und mit dem Android Open Accessory Development Kit kompatibel ist (der offiziellen Methode, Hardware an Android-Geräte anzuschließen). Das Leonardo-Board verwendet einen Controller-Chip (den ATmega32u4), der unterschiedliche HID-Geräte repräsentieren kann. Das Ethernet-Board enthält eine Ethernet-Schnittstelle und eine Power-Over-Ethernet-Option, d.h., man kann das Board über ein einziges Kabel anbinden und mit Strom versorgen.

Es gibt noch weitere Arduino-kompatible Boards, einschließlich der folgenden:

- Arduino Nano, ein kleines, USB-fähiges Board von Gravitech (*http://store.gravitech.us/arna30wiatn.html*)
- Bare Bones Board, ein preiswertes Board mit oder ohne USB von Modern Device (*http://www.moderndevice.com/products/bbb-kit*)
- Boarduino, ein preiswertes, für Steckbretter geeignetes Board von Adafruit Industries (*http://www.adafruit.com/*)
- Seeeduino, eine flexible Variante des Standard-USB-Boards von Seeed Studio Bazaar (*http://www.seeedstudio.com/*)
- Teensy und Teensy++, kleine, aber extrem vielseitige Boards von PJRC (*http://www.pjrc.com/teensy/*)

Eine Liste Arduino-kompatibler Boards finden Sie unter *http://www.freeduino.org/*.

Siehe auch

Übersicht der Arduino-Boards: *http://www.arduino.cc/en/Main/Hardware*.

Online-Leitfäden für den Arduino-Einstieg finden Sie unter *http://arduino.cc/en/Guide/Windows* für Windows, *http://arduino.cc/en/Guide/MacOSX* für Mac OS X und *http://www.arduino.cc/playground/Learning/Linux* für Linux.

Eine Liste von über einhundert Boards, die mit der Arduino-Entwicklungsumgebung genutzt werden können, finden Sie unter *http://jmsarduino.blogspot.com/2009/03/comprehensive-arduino-compatible.html*

1.1 Installation der integrierten Entwicklungsumgebung (IDE)

Problem

Sie möchten die Arduino-Entwicklungsumgebung auf Ihrem Computer installieren.

Lösung

Die Arduino-Software für Windows, Mac und Linux kann von *http://arduino.cc/en/Main/Software* heruntergeladen werden.

Der Windows-Download ist eine ZIP-Datei. Entpacken Sie die Datei in ein geeignetes Verzeichnis – *Programme/Arduino* ist eine gute Wahl.

Ein freies Utility zum Entpacken von Dateien namens 7-Zip kann von *http://www.7-zip.org/* heruntergeladen werden.

Das Entpacken der Datei erzeugt einen Ordner namens *Arduino-00<nn>* (dabei ist *<nn>* die Versionsnummer der heruntergeladenen Arduino-Release). Das Verzeichnis enthält neben verschiedenen Dateien und Ordnern auch eine ausführbare Datei namens *Arduino.exe*. Klicken Sie *Arduino.exe* doppelt an und der »Splash Screen« (siehe Abbildung 1-2) sollte erscheinen, gefolgt vom Haupt-Programmfenster (siehe Abbildung 1-3). Haben Sie Geduld: Es kann einige Zeit dauern, bis die Software geladen ist.

Abbildung 1-2: Arduino Splash Screen (Version 1.0 unter Windows 7)

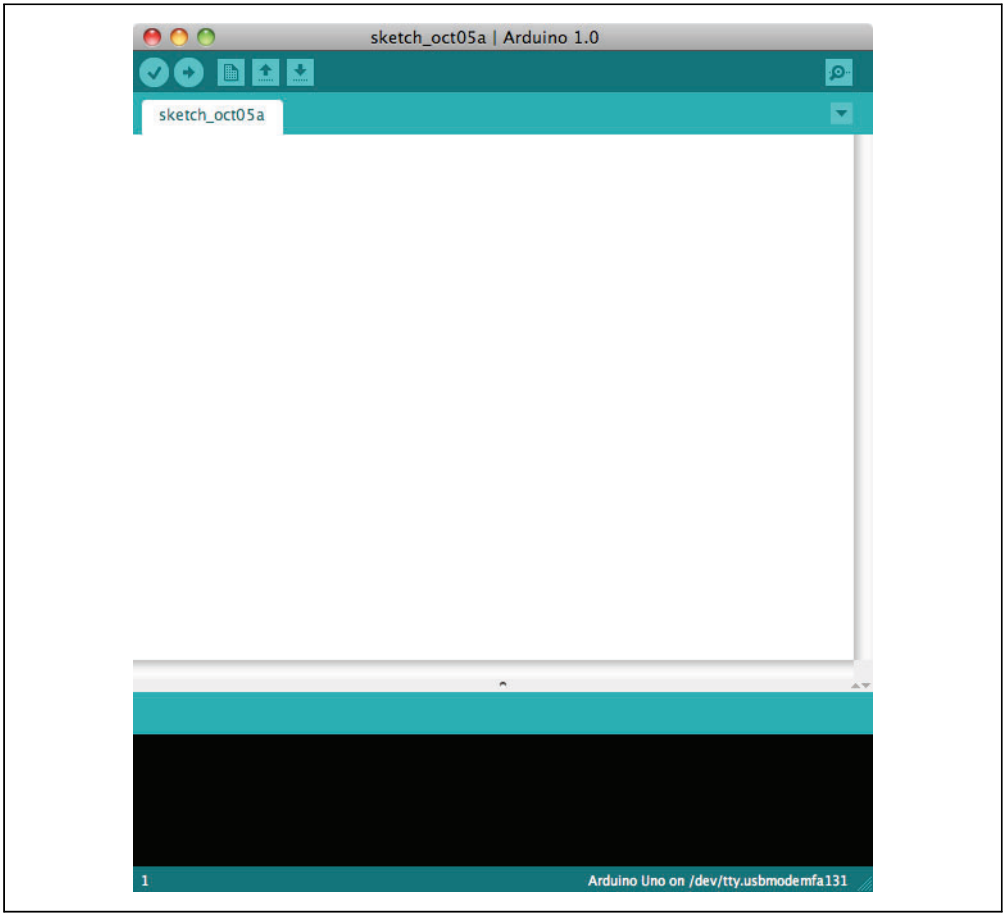


Abbildung 1-3: IDE-Hauptfenster (Arduino 1.0 auf einem Mac)

Der Arduino-Download für den Mac ist ein Disk-Image (*.dmg*). Klicken Sie die Datei doppelt an, nachdem der Download abgeschlossen ist. Das Image wird gemountet (und erscheint wie ein Speicherstick auf dem Desktop). Innerhalb des Disk-Images befindet sich die Arduino-Anwendung. Kopieren Sie sie an einen geeigneten Ort – der Ordner *Programme* ist eine gute Wahl. Sobald Sie die Datei kopiert haben, klicken Sie sie doppelt an (sie aus dem Disk-Image auszuführen, ist keine gute Idee). Der Splash Screen erscheint, gefolgt vom Haupt-Programmfenster.

Die Linux- Installation ist von der verwendeten Linux-Distribution abhängig. Informationen finden Sie im Arduino-Wiki (*http://www.arduino.cc/playground/Learning/Linux*).

Damit die Arduino-Entwicklungsumgebung mit dem Board kommunizieren kann, müssen Sie Treiber installieren.

Unter Windows, verbinden Sie Ihren PC und das Arduino-Board über ein USB-Kabel und warten, dass der »Neue Hardware«-Assistent erscheint. Wenn Sie ein Uno-Board ver-

wenden, lassen Sie den Assistenten versuchen, die Treiber zu suchen und installieren. Der Versuch schlägt fehl (keine Sorge, das ist das erwartete Verhalten). Um das zu beheben, wechseln Sie nun nach Startmenü→Systemsteuerung→System und Sicherheit. Klicken Sie auf System und öffnen Sie den Gerätemanager. In der dargestellten Liste wählen Sie dann den Eintrag in COM und LPT namens `Arduino UNO (COM nn)`. `nn` ist die Nummer, die Windows dem für das Board erzeugten Port zugewiesen hat. Daneben sehen Sie eine Warnung, da die richtigen Treiber noch nicht zugewiesen wurden. Klicken Sie den Eintrag mit der rechten Maustaste an und wählen Sie Treibersoftware aktualisieren. Wählen Sie dann die Option »Browse my computer for driver software« und bewegen Sie sich in den *Drivers*-Ordner im eben entpackten Arduino-Ordner. Wählen Sie die Datei `Arduino-UNO.inf`, und Windows sollte den Installationsprozess abschließen.

Wenn Sie ein älteres Board (das FTDI-Treiber verwendet) mit Windows Vista oder Windows 7 nutzen und online sind, können Sie den Assistenten nach Treibern suchen lassen, und sie sollten automatisch installiert werden. Unter Windows XP (oder wenn Sie keinen Internetzugang haben) müssen Sie die Lage des Treibers angeben. Bewegen Sie sich in der Dateiauswahl ins Verzeichnis *FTDI USB Drivers*. Sie finden es in dem Verzeichnis, in dem Sie die Arduino-Dateien entpackt haben. Sobald der Treiber installiert ist, erscheint wieder der »Neue Hardware«-Assistent mit der Meldung, eine neue serielle Schnittstelle sei gefunden worden. Folgen Sie nun den Anweisungen von vorhin.

Es ist wichtig, dass Sie diese Schritte zur Installation des Treibers zweimal durchgehen, da die Software anderenfalls nicht mit dem Board kommunizieren kann.

Auf dem Mac sollten neuere Arduino-Boards wie das Uno ohne zusätzliche Treiber genutzt werden können. Wenn Sie das Board zum ersten Mal anschließen, erscheint ein Hinweis, dass eine neue Netzwerkschnittstelle gefunden wurde. Bei älteren Boards (die FTDI-Treiber benötigen), müssen Sie Treibersoftware installieren. Im Disk-Image finden Sie ein Paket namens *FTDIUSBSerialDriver* mit einer Reihe von Zahlen dahinter. Klicken Sie das Paket an und der Installer führt Sie durch den Prozess. Sie müssen das Administrationspasswort kennen, um den Vorgang abschließen zu können.

Unter Linux ist der Treiber bei den meisten Distributionen bereits installiert. Informationen zu Ihrer Distribution finden Sie unter dem Linux-Link, der in der Kapiteleinführung genannt wurde.

Diskussion

Falls die Software nicht startet, besuchen Sie den Fehlersuche-Bereich der Arduino-Website unter *http://arduino.cc/en/Guide/Troubleshooting*. Hier finden Sie Hinweise zur Lösung von Installationsproblemen.

Siehe auch

Online-Leitfäden für den Arduino-Einstieg finden Sie unter *http://arduino.cc/en/Guide/Windows* für Windows, *http://arduino.cc/en/Guide/MacOSX* für Mac OS X und *http://www.arduino.cc/playground/Learning/Linux* für Linux.

1.2 Das Arduino-Board einrichten

Problem

Sie möchten ein Arduino-Board einschalten und sicherstellen, dass es funktioniert.

Lösung

Verbinden Sie das Board mit einem USB-Port Ihres Computers und stellen Sie sicher, dass die grüne Betriebs-LED leuchtet. Standard Arduino-Boards (Uno, Duemilanove und Mega) haben eine grüne Betriebs-LED in der Nähe des Reset-Tasters.

Eine orange LED nahe der Mitte der Platine (»Pin 13 LED« in Abbildung 1-4) sollte an- und ausgehen, sobald das Board eingeschaltet ist. Boards werden werksseitig mit vorinstallierter Software ausgeliefert, die die LED ein- und ausschaltet. Auf diese Weise lässt sich einfach prüfen, ob das Board funktioniert.

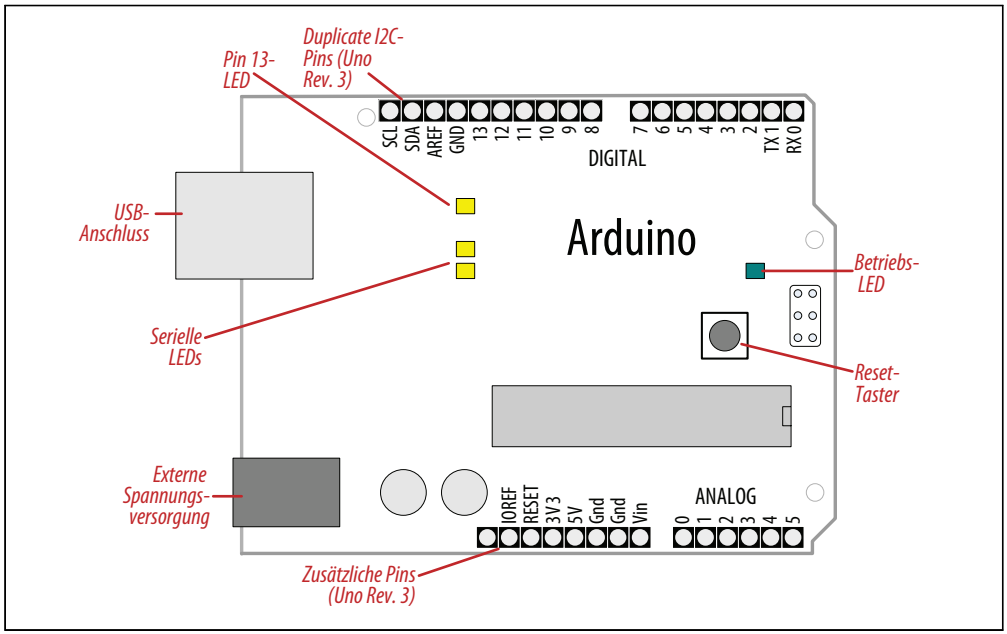


Abbildung 1-4: Einfache Arduino-Board (Duemilanove und Uno)

Bei neuen Boards wie dem Leonardo befinden sich die LEDs in der Nähe des USB-Anschlusses (siehe Abbildung 1-5). Neue Boards haben auch doppelte Pins für I2C (`SCL` und `SDA`). Diese Boards besitzen auch einen Pin namens `IOREF`, mit dessen Hilfe die Betriebsspannung des Chips bestimmt werden kann.

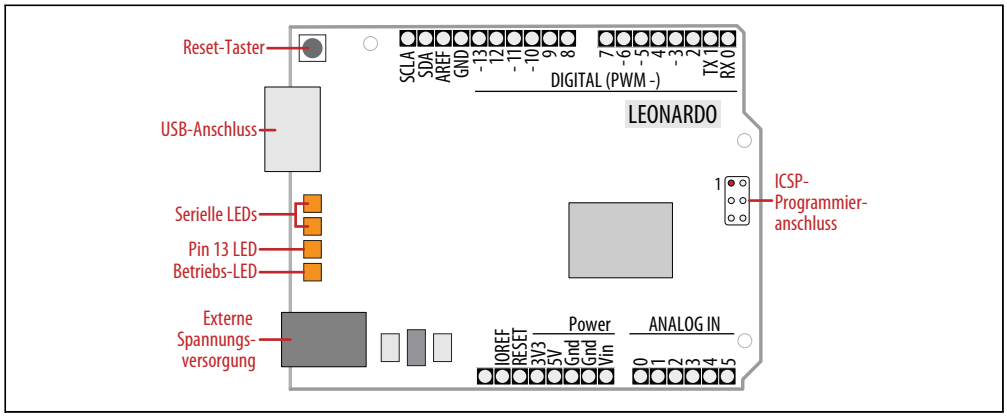


Abbildung 1-5: Leonardo-Board

Der neue Standard der aktuellen Boards verfügt über drei zusätzliche Anschlüsse im Anschlusslayout. Das hat keinen Einfluss auf ältere Shields, die in den neuen Boards genauso laufen wie in den alten. Die neuen Anschlüsse bestehen aus dem Pin IOREF, mit dem die analoge Referenzspannung ermittelt werden kann (so dass die analogen Eingangswerte mit der Stromversorgung abgeglichen werden können), sowie aus den Pins SCL and SDA, die eine konsistenteVerbindung für I2C-Geräte ermöglichen. Die Lage der I2C-Pins ist bei älteren Boards eine andere, da die Chips unterschiedlich konfiguriert sind. Für das neue Layout entwickelte Shields sollten mit jedem Board laufen, das die neue Lage der Pins nutzt. Ein weiterer Pin (neben dem IOREF-Pin) wird momentan nicht genutzt, ermöglicht aber die zukünftige Implementierung neuer Features, ohne das Pin-Layout erneut ändern zu müssen.

Diskussion

Leuchtet die Betriebs-LED nicht, wenn das Board mit dem Computer verbunden ist, erhält das Board wahrscheinlich keinen Strom.

Die blinkende LED (die mit dem digitalen Ausgang an Pin 13 verbunden ist) wird durch Code gesteuert, der auf dem Board läuft (bei neuen Boards ist der Blink-Sketch vorinstalliert). Wenn die LED an Pin 13 blinkt, wird der Sketch korrekt ausgeführt, was wiederum bedeutet, dass der Chip auf dem Board funktioniert. Wenn die grüne Betriebs-LED leuchtet, die LED an Pin 13 aber nicht blinkt, kann es sein, dass der Code werksseitig nicht auf dem Chip installiert wurde. Folgen Sie den Anweisungen in Rezept 1.3, um den Blink-Sketch auf das Board zu laden und die Funktionstüchtigkeit des Boards zu über-

prüfen. Wenn Sie kein Standard-Board verwenden, gibt es möglicherweise keine feste LED an Pin 13. Dann müssen Sie die Details des Boards in der Dokumentation nachlesen. Beim Leonardo-Board sieht es so aus, als würde die LED »atmen«, wenn das Board funktioniert.

Siehe auch

Online-Leitfäden für den Arduino-Einstieg finden Sie unter *http://arduino.cc/en/Guide/Windows* für Windows, *http://arduino.cc/en/Guide/MacOSX* für Mac OS X und *http://www.arduino.cc/playground/Learning/Linux* für Linux.

Hilfe bei der Fehlersuche finden Sie unter *http://arduino.cc/en/Guide/Troubleshooting*.

1.3 Einen Arduino-Sketch mit der integrierten Entwicklungsumgebung (IDE) bearbeiten

Problem

Sie wollen einen Sketch bearbeiten und für den Upload auf das Board vorbereiten.

Lösung

Verwenden Sie die Arduino-IDE, um Sketches anzulegen, zu öffnen und zu ändern. (Sketches legen fest, was das Board machen soll.) Sie können diese Aktionen über die Buttons am oberen Rand durchführen (siehe Abbildung 1-6) oder die Menüs und Tastaturkürzel (siehe Abbildung 1-7) nutzen.

Im Sketcheditor betrachten und editieren Sie den Code eines Sketches. Er unterstützt gängige Textbearbeitungs-Tasten wie Ctrl-F (⌘+F auf dem Mac) für die Suche, Ctrl-Z (⌘+Z auf dem Mac) für Undo, Ctrl-C (⌘+C auf dem Mac) für das Kopieren markierten Textes und Ctrl-V (⌘+V auf dem Mac) für das Einfügen von Text.

Abbildung 1-7 zeigt, wie man den Blink-Sketch lädt (der Sketch ist auf neuen Arduino-Boards vorinstalliert).

Nachdem Sie die IDE gestartet haben, wechseln Sie in das Menü File → Examples und wählen 1. Basics→Blink (siehe Abbildung 1-7). Der Code, der die eingebaute LED blinken lässt, erscheint im Sketch-Editor (siehe Abbildung 1-6).

Bevor der Code auf das Board übertragen werden kann, muss er in Instruktionen umgewandelt werden, die vom Arduino-Controller-Chip gelesen und ausgeführt werden können. Diesen Vorgang bezeichnet man als *Kompilierung*. Dazu klicken Sie den Compiler-Button (den oben links mit dem Häkchen) an, oder wählen Sketch→Verify/Compile (Ctrl-R; ⌘+R auf dem Mac).

Im Nachrichtenbereich unter dem Editor sollte die Meldung »Compiling sketch...« und eine Fortschrittsanzeige erscheinen. Nach ein oder zwei Sekunden erscheint die Meldung »Done Compiling«. Der schwarze Konsolenbereich enthält zusätzlich die folgende Meldung:

```
Binary sketch size: 1026 bytes (of a 32256 byte maximum)
```

Die genaue Meldung hängt von Ihrem Board und der Arduino-Version ab. Sie gibt an, wie groß der Sketch ist und welche Größe Ihr Board maximal akzeptiert.

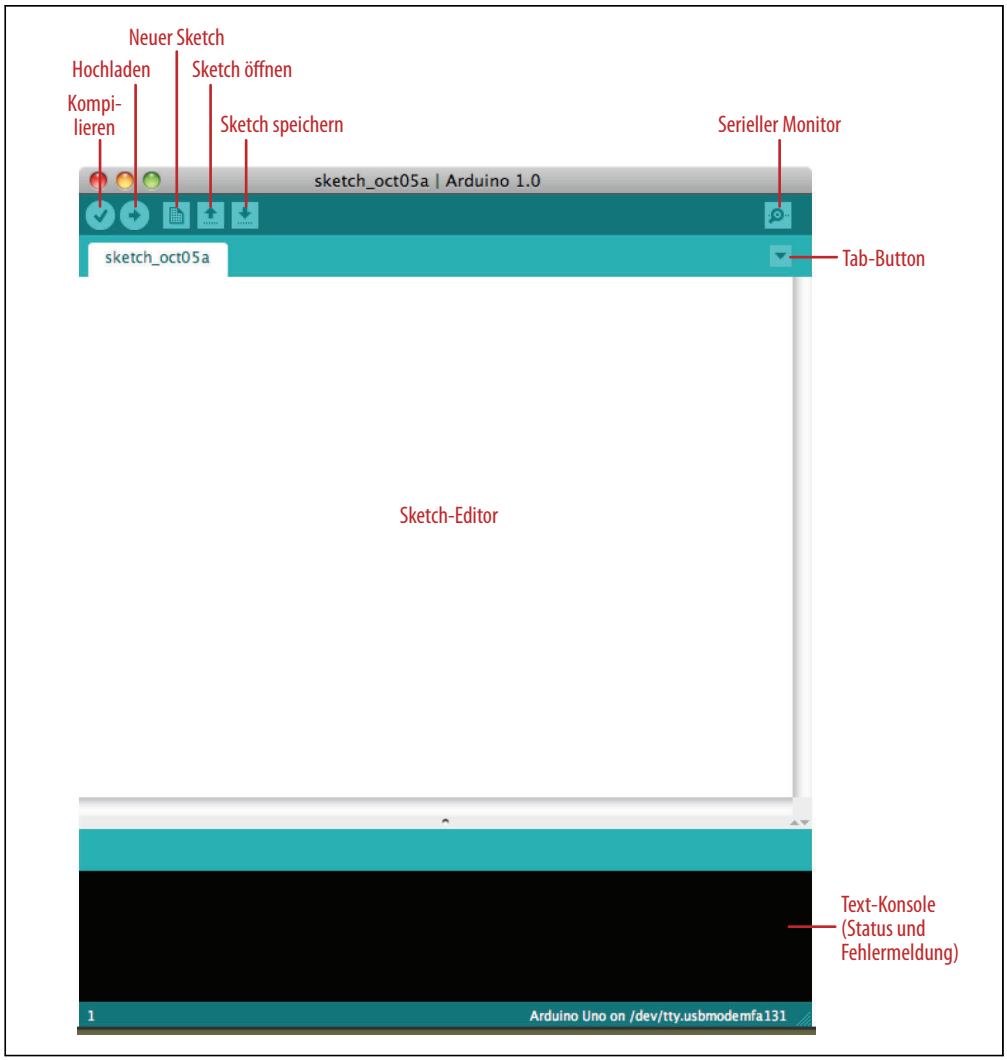


Abbildung 1-6: Arduino-IDE

Diskussion

Der Quellcode für Arduino wird *Sketch* genannt. Der Prozess, der einen solchen Sketch in eine Form umwandelt, die auf dem Board funktioniert, nennt man *Kompilierung*. Die IDE nutzt hinter den Kulissen eine Reihe von Kommandozeilen-Werkzeugen, um einen Sketch zu kompilieren. Weitere Informationen hierzu finden Sie im Rezept 17.1.

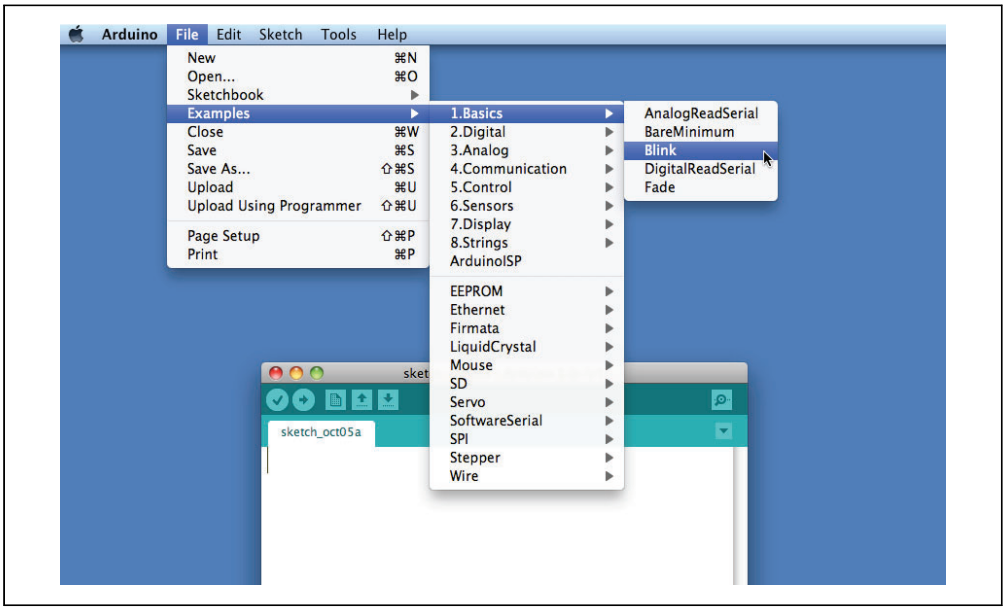


Abbildung 1-7: IDE-Menü (Auswahl des Blink-Beispiel-Sketches)

Die abschließende Meldung, die die Größe des Sketches angibt, sagt Ihnen, wie viel Programmspeicher benötigt wird, um die Controller-Instruktionen auf dem Board zu speichern. Ist der kompilierte Sketch größer als der auf dem Board verfügbare Speicher, erscheint die folgende Fehlermeldung:

```
Sketch too big; see http://www.arduino.cc/en/Guide/Troubleshooting#size
 for tips on reducing it.
```

In diesem Fall müssen Sie den Sketch kürzen, damit er auf das Board passt, oder Sie müssen sich ein Board mit einer höheren Kapazität besorgen.

Wenn der Code selbst fehlerhaft ist, gibt der Compiler ein oder mehrere Fehlermeldungen im Konsolenfenster aus. Diese Meldungen helfen bei der Identifikation des Fehlers. Tipps zur Behebung von Softwarefehlern finden Sie im Anhang D (steht als Download bereit).

Um das versehentliche Überschreiben der Beispiele zu verhindern, erlaubt es die Arduino-IDE nicht, Änderungen an den Beispiel-Sketches zu speichern. Sie müssen sie zuerst mit der Menüoption Save As umbenennen. Selbst geschriebene Sketches können mit dem Save-Button gespeichert werden (siehe Rezept 1.5).

Während Sie einen Sketch entwickeln und verändern, sollten Sie die Menüoption File → Save regelmäßig nutzen und verschiedene Namen oder Versionsnummern verwenden, so dass Sie bei der Implementierung bei Bedarf auf eine ältere Version zurückgreifen können.

Auf das Board hochgeladener Code kann nicht wieder auf den Computer heruntergeladen werden. Stellen Sie also sicher, dass Ihr Sketch-Code auf dem Computer gespeichert ist. Sie können Änderungen an den Beispieldateien auch nicht speichern, sondern müssen Save As nutzen und der geänderten Datei einen anderen Namen geben.

Siehe auch

Rezept 1.5 zeigt einen Beispiel-Sketch. Anhang D (steht als Download bereit) gibt Hinweise zur Fehlersuche bei Software-Problemen.

1.4 Den Blink-Sketch hochladen und ausführen

Problem

Sie wollen Ihren kompilierten Sketch an das Arduino-Board senden und ausführen.

Lösung

Verbinden Sie Ihr Arduino-Board über ein USB-Kabel mit Ihrem Computer. Laden Sie den Blink-Sketch wie in Rezept 1.3 beschrieben in die IDE.

Als nächstes wählen Sie Tools→Board aus dem Dropdown-Menü und wählen den Namen des angeschlossenen Boards (falls Sie ein Standard-Uno-Board verwenden, ist es wahrscheinlich der erste Eintrag in der Board-Liste).

Nun wählen Sie Tools→Serial Port. Es erscheint eine Dropdown-Liste der auf Ihrem Computer verfügbaren seriellen Ports. Jeder Rechner hat eine andere Kombination serieller Ports, je nachdem welche anderen Geräte auf dem Computer verwendet werden.

Unter Windows wird eine Liste durchnummerierter COM-Einträge ausgegeben. Gibt es nur einen Eintrag, dann wählen Sie ihn aus. Bei mehreren Einträgen ist Ihr Board wahrscheinlich der letzte Eintrag.

Bei einem Mac wird Ihr Board zweimal aufgeführt, wenn es sich um ein Uno-Board handelt:

```
/dev/tty.usbmodem-XXXXXXX
/dev/cu.usbmodem-XXXXXXX
```

Ein älteres Board wird wie folgt aufgeführt:

```
/dev/tty.usbserial-XXXXXXX
/dev/cu.usbserial-XXXXXXX
```

Jedes Board besitzt einen anderen Wert für *XXXXXXX*. Wählen Sie einen beliebigen Eintrag.

Klicken Sie den Upload-Button an (in Abbildung 1-6 der zweite Button von links), oder wählen Sie File→Upload to I/O board (Ctrl-U, ⌘+U auf einem Mac).

Die Software kompiliert den Code wie in Rezept 1.3 beschrieben. Nachdem die Software kompiliert wurde, wird sie auf das Board hochgeladen. Wenn Sie auf das Board schauen, sehen Sie, dass die LED aufhört zu blinken und zwei LEDs (die »Seriell«-LEDs in Abbildung 1-4) direkt unter der eben noch blinkenden LED für einige Sekunden flackern, während der Code hochgeladen wird. Die ursprüngliche LED fängt dann wieder an zu blinken, sobald der Code ausgeführt wird.

Diskussion

Damit die IDE kompilierten Code an das Board senden kann, muss das Board mit dem Computer verbunden sein und Sie müssen der IDE mitteilen, welches Board und welchen seriellen Port Sie verwenden.

Wenn ein Upload beginnt, wird ein auf dem Board laufender Sketch angehalten (wenn der Blick-Sketch läuft, hört die LED auf zu blinken). Der neue Sketch wird auf das Board hochgeladen und ersetzt den vorhandenen Sketch. Der neue Sketch wird ausgeführt, sobald das Hochladen erfolgreich abgeschlossen wurde.

Ältere Arduino-Boards (und einige kompatible) unterbrechen einen laufenden Sketch zur Initiierung des Uploads nicht. In diesem Fall müssen Sie die Reset-Taste auf dem Board drücken, sobald die Kompilierung abgeschlossen ist (d.h., wenn Sie die Meldung zur Größe des Sketches sehen). Möglicherweise brauchen Sie einige Versuche, bis Sie das richtige Timing zwischen dem Ende der Kompilierung und dem Drücken des Reset-Tasters hinbekommen.

Die IDE gibt eine Fehlermeldung aus, wenn der Upload nicht erfolgreich war. Die typischen Probleme sind falsch gewählte Boards oder serielle Ports, oder nicht korrekt angeschlossene Boards. Das gewählte Board und der serielle Port werden in der Statusleiste am unteren Rand des Arduino-Fensters angezeigt.

Wenn Sie Schwierigkeiten haben, den richtigen Port unter Windows zu bestimmen, trennen Sie das Board vom Computer, wählen Tools→Serial Port und sehen nach, welcher COM-Port nicht mehr in der Liste steht. Ein anderer Ansatz besteht darin, die Ports nacheinander auszuwählen, bis die LEDs des Boards zu flackern beginnen (was bedeutet, dass der Code hochgeladen wird).

Siehe auch

Die Arduino-Seite zur Fehlersuche: *http://www.arduino.cc/en/Guide/Troubleshooting*.

1.5 Einen Sketch erstellen und speichern

Problem

Sie möchten einen Sketch erstellen und auf dem Computer speichern.

Lösung

Um ein Editor-Fenster zu öffnen, in das Sie einen neuen Sketch eingeben können, starten Sie die IDE (siehe Rezept 1.3), wechseln ins File-Menü und wählen New. Geben Sie den folgenden Code im Sketch-Editor ein (es ähnelt dem Blink-Sketch, aber die Blinkdauer ist doppelt so lang):

```
const int ledPin = 13;   // Mit Pin 13 verbundene LED

void setup()
{
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(2000);          // Zwei Sekunden warten
 digitalWrite(ledPin, LOW);   // LED ausschalten
 delay(2000);          // Zwei Sekunden warten
}
```

Kompilieren Sie den Code durch Anklicken des Compiler-Buttons (oben links mit dem Häkchen), oder wählen Sie Sketch→Verify/Compile (siehe Rezept 1.3).

Laden Sie den Code über den Upload-Button oder über File→Upload to I/O board (siehe Rezept 1.4) hoch. Nach dem Hochladen sollte die LED in einem Intervall von zwei Sekunden blinken.

Sie können den Sketch auf dem Computer sichern, indem Sie den Save-Button anklicken oder File→Save auswählen.

Sie können den Sketch unter einem neuen Namen sichern, indem Sie die Menüoption Save As wählen. Es erscheint eine Dialogbox, in die Sie den neuen Dateinamen eintragen können.

Diskussion

Wenn Sie eine Datei mit der IDE sichern, erscheint eine Standard-Dialogbox des Betriebssystems. Per Voreinstellung wird der Sketch in einem Ordner namens *Arduino* unter

Eigene Dateien (bzw. dem *Dokumenten*-Ordner auf einem Mac) gespeichert. Sie können den voreingestellten Sketch-Namen durch einen sinnvollen Namen ersetzen, der den Zweck des Sketches widerspiegelt.

Der Standardname besteht aus dem Wort *Sketch*, gefolgt vom aktuellen Datum. Bei *a* beginnende Zeichenfolgen werden verwendet, um am gleichen Tag angelegte Sketches zu unterscheiden. Den Standardnamen durch etwas Sinnvolles zu ersetzen, hilft Ihnen dabei, den Zweck des Sketches zu erkennen, wenn Sie ihn später wieder bearbeiten.

Wenn Sie von der IDE nicht erlaubte Zeichen verwenden (z.B. ein Leerzeichen), ersetzt die IDE sie automatisch durch gültige Zeichen.

Arduino-Sketches werden als reine Textdateien mit der Erweiterung *.ino* gespeichert. Ältere Versionen der IDE verwenden die Erweiterung *.pde*, die auch von Processing genutzt wird. Sie werden automatisch in einem Ordner gesichert, der den gleichen Namen hat wie der Sketch.

Sie können Ihre Sketches in jedem beliebigen Ordner Ihres Computers sichern, doch wenn Sie den Standardordner nutzen (den *Arduino*-Ordner unterhalb Ihrer *Dokumente*), erscheinen die Sketches automatisch im Sketchbook-Menü der Arduino-Software und sind leichter zu finden.

Wenn Sie eines der Beispiele des Arduino-Downloads bearbeitet haben, können Sie die Änderungen nicht unter dem gleichen Dateinamen sichern. Auf diese Weise bleiben die Standard-Beispiele unangetastet. Wenn Sie das modifizierte Beispiel sichern wollen, müssen Sie einen anderen Ort für den Sketch wählen.

Wenn Sie Änderungen vorgenommen haben und einen Sketch schließen, erscheint eine Dialogbox, die Sie fragt, ob Sie die Änderungen sichern wollen.

Das Symbol § hinter dem Namen des Sketches in der oberen Leiste des IDE-Fensters zeigt an, dass der Sketch-Code geändert, aber noch nicht auf dem Computer gesichert wurde. Das Symbol wird entfernt, wenn der Sketch gesichert wird.

Die Arduino-Software bietet keinerlei Versionskontrolle. Wenn Sie also zu einer älteren Version Ihres Sketches zurückkehren wollen, müssen Sie Save As regelmäßig verwenden und jeder Version des Sketches einen etwas anderen Namen geben.

Die häufige Kompilierung während der Bearbeitung ist eine gute Möglichkeit, den von Ihnen geschriebenen Code auf Fehler zu überprüfen. Fehler aufzuspüren und zu korrigieren, wird auf diese Weise einfacher, weil sie direkt mit Ihren aktuellen Eingaben zusammenhängen.

Sobald ein Sketch auf ein Board hochgeladen wurde, gibt es keine Möglichkeit, ihn wieder auf den Rechner herunterzuladen. Sichern Sie also alle Änderungen an Ihren Sketches, die Sie behalten wollen.

Wenn Sie versuchen, einen Sketch zu sichern, der nicht in einem Ordner liegt, der den gleichen Namen wie der Sketch hat, informiert Sie die IDE darüber, dass das so nicht geht, und empfiehlt, OK anzuklicken, um einen Ordner mit dem gleichen Namen für den Sketch anzulegen.

Sketches müssen in einem Ordner liegen, der den gleichen Namen hat wie der Sketch. Die IDE legt den Ordner automatisch an, wenn Sie einen neuen Sketch sichern.

Mit älterer Arduino-Software entwickelte Sketches verwenden eine andere Dateierweiterng (*.pde*). Die IDE öffnet sie und erzeugt eine Datei mit einer neuen Erweiterung (*.ino*), wenn Sie sie sichern. Für ältere Versionen der IDE enwickelter Code kompiliert unter der Version 1.0 möglicherweise nicht. Die meisten Änderungen, die notwendig sind, um älteren Code ans Laufen zu kriegen, sind aber einfach vorzunehmen. Details finden Sie in Anhang H (steht als Download bereit).

Siehe auch

Der Code in diesem Rezept (und im Rest des Buches) verwendet den Ausdruck `const int`, um für Konstanten sinnvolle Namen (`ledPin`) anstelle von Zahlen (`13`) zu nutzen. Mehr zur Verwendung von Konstanten finden Sie in Rezept 17.5 (steht als Download bereit).

1.6 Arduino verwenden

Problem

Sie möchten ein Projekt beginnen, das einfach zu bauen ist und gleichzeitig Spaß macht.

Lösung

Dieses Rezept bietet einen Vorgeschmack auf einige der Techniken, die in späteren Kapiteln detailliert behandelt werden.

Der Sketch basiert auf dem Code für die blinkende LED aus dem vorherigen Rezept. Anstelle eines festen Zeitintervalls wird die Dauer aber über einen Sensor, einen lichtempfindlichen Widerstand, bestimmt (siehe Rezept 6.2). Verdrahten Sie den lichtempfindlichen Widerstand wie in Abbildung 1-8 gezeigt.

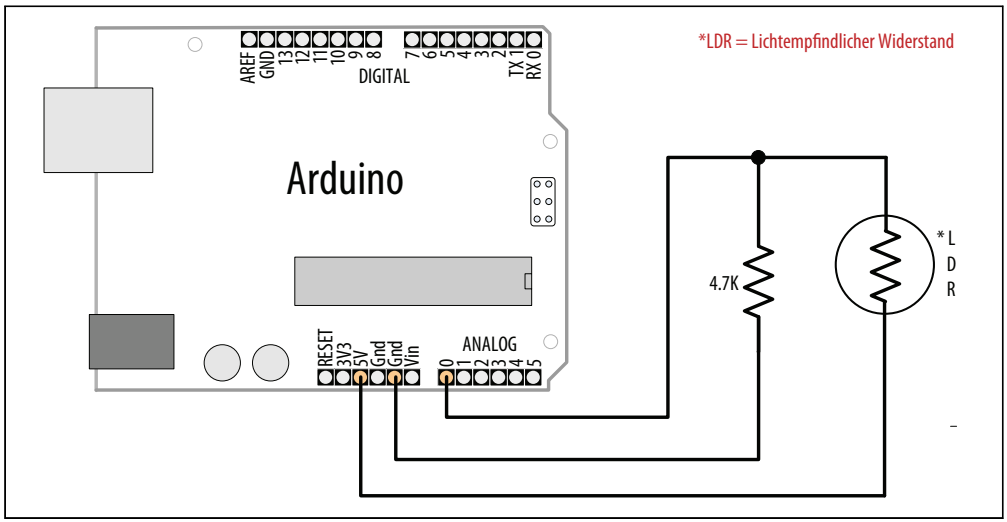


Abbildung 1-8: Arduino mit lichtempfindlichem Widerstand

Wenn Sie nicht damit vertraut sind, einen Schaltkreis nach einem Schaltplan aufzubauen, sehen Sie sich Anhang B (steht als Download bereit) an. Hier wird Schritt für Schitt gezeigt, wie man eine Schaltung auf einem Steckbrett aufbaut.

Der folgende Sketch liest die Lichtintensität des lichtempfindlichen Widerstands ein, der mit dem analogen Pin 0 verbunden ist. Diese Lichtintensität verändert die Blinkgeschwindigkeit der internen LED an Pin 13:

```
const int ledPin =  13;   // Mit Pin 13 verbundene LED
const int sensorPin = 0;  // Mit Analogeingang 0 verbundener Sensor

void setup()
{
 pinMode(ledPin, OUTPUT); // LED-Pin als Ausgang aktivieren
}

void loop()
{
 int rate = analogRead(sensorPin);  // Analogen Eingang einlesen
 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(rate);           // Wartezeit abhängig von Lichtintensität
 digitalWrite(ledPin, LOW);  // LED ausschalten
 delay(rate);
}
```

Diskussion

Der Wert des 4,7K-Widerstands ist unkritisch. Sie können alles zwischen 1K und 10K verwenden. Die Lichtintensität am lichtempfindlichen Widerstand ändert die Spannung an Analogpin 0. Der Befehl `analogRead` (siehe Kapitel 6) liefert einen Wert zwischen 200

(dunkel) und 800 (hell) zurück. Dieser Wert bestimmt, wie lange die LED an und aus bleibt. Das Blinkintervall wird also mit zunehmender Lichtstärke länger.

Sie können das Blinkintervall mit Hilfe der Arduino-Funktion `map` wie folgt skalieren:

```
const int ledPin =  13;   // Mit Pin 13 verbundene LED
const int sensorPin = 0;  // Mit Analogeingang 0 verbundener Sensor

// die nächsten  beiden Zeilen legen das minimale und maximale Zeitintervall fest
const int minDuration = 100; // Minimale Dauer
const int maxDuration = 1000; // Maximale Dauer


void setup()
{
 pinMode(ledPin, OUTPUT); // LED-Pin als Ausgang aktivieren
}

void loop()
{
 int rate = analogRead(sensorPin);  // Analogen Eingang einlesen
 // Die nächste Zeile skaliert das Blinkintervall auf die Minimal- und Maximalwerte
 rate = map(rate, 200,800,minDuration, maxDuration); // In Blinkintervall umwandeln
 rate = constrain(rate, minDuration,maxDuration);  // und Wert beschränken

 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(rate);          // Wartezeit abhängig von Lichtintensität
 digitalWrite(ledPin, LOW);  // LED ausschalten
 delay(rate);
}
```

In Rezept 5.7 wird detailliert beschrieben, wie man die `map`-Funktion zur Skalierung von Werten nutzt. Rezept 3.5 zeigt im Detail, wie man die `constrain`-Funktion verwendet, damit ein Wert einen bestimmten Wertebereich nicht über- oder unterschreitet.

Wenn Sie sich den Wert der rate-Variablen auf Ihrem Computer ansehen wollen, können Sie ihn über den seriellen Monitor ausgeben. Wie das geht, zeigt der nachfolgende, überarbeitete Code in der `loop()`-Funktion. Der Sketch gibt die Blinkrate über den seriellen Monitor aus. Sie öffnen den seriellen Monitor in der Arduino IDE, indem Sie das Icon rechts in der oberen Leiste anklicken (mehr zur Verwendung des seriellen Monitors finden Sie in Kapitel 4):

```
const int ledPin =  13;   // Mit Pin 13 verbundene LED
const int sensorPin = 0;  // Mit Analogeingang 0 verbundener Sensor

// die nächsten  beiden Zeilen legen das minimale und maximale Zeitintervall fest
const int minDuration = 100; // Minimale Dauer
const int maxDuration = 1000; // Maximale Dauer


void setup()
{
 pinMode(ledPin, OUTPUT); // LED-Pin als Ausgang aktivieren
 Serial.begin(9600);    // Seriellen Port initialisieren
}
```

```
void loop()
{
 int rate = analogRead(sensorPin);   // Analogen Eingang einlesen
 // Die nächste Zeile skaliert das Blinkintervall auf die Minimal- und Maximalwerte
 rate = map(rate, 200,800,minDuration, maxDuration); // In Blinkintervall umwandeln
 rate = constrain(rate, minDuration,maxDuration);   // und Wert beschränken

 Serial.println(rate);      // Intervall über seriellen Monitor ausgeben
 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(rate);           // Wartezeit abhängig von Lichtintensität
 digitalWrite(ledPin, LOW);   // LED ausschalten
 delay(rate);
}
```

Sie können mit dem lichtempfindlichen Widerstand auch die Tonlage eines Lautspechers steuern, wenn Sie ihn wie in Abbildung 1-9 anschließen.

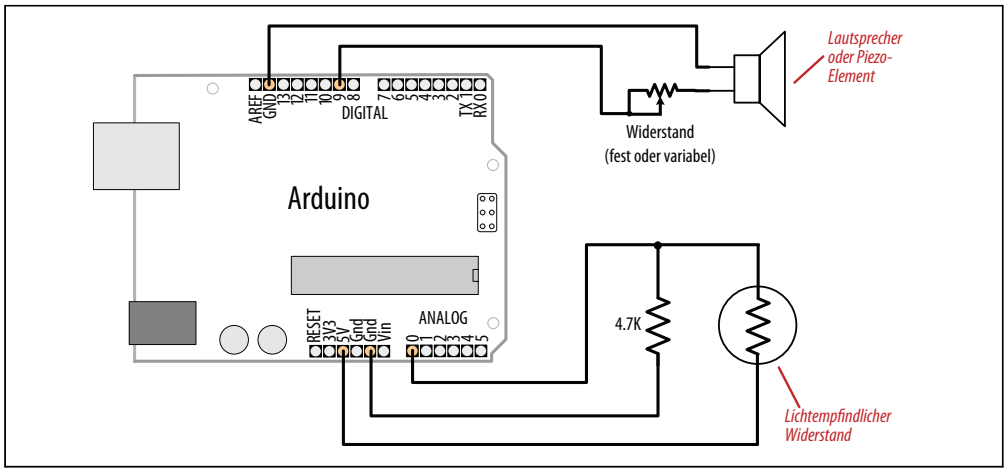


Abbildung 1-9: Verbinden eines Lautsprechers mit dem lichtempfindlichen Widerstand

Sie müssen das Ein/Aus-Intervall des Pins auf eine Frequenz im Audiospektrum erhöhen. Das wird im nachfolgenden Code erreicht, indem die minimale und maximale Dauer reduziert wird:

```
const int outputPin = 9;   // Mit digitalem Pin 9 verbundener Lautsprecher
const int sensorPin = 0;   // Mit Analogeingang 0 verbundener Sensor

const int minDuration = 1;  // 1ms an, 1ms aus (500 Hz)
const int maxDuration = 10; // 10ms an, 10ms aus (50 hz)

void setup()
{
 pinMode(outputPin, OUTPUT); // LED-Pin als Ausgang aktivieren
}

void loop()
{
```

```
 int sensorReading = analogRead(sensorPin);   // Analogeingang einlesen
 int rate = map(sensorReading, 200,800,minDuration, maxDuration);
 rate = constrain(rate, minDuration,maxDuration);   // Wert beschränken

 digitalWrite(outputPin, HIGH);  // LED einschalten
 delay(rate);            // Wartezeit abhängig von Lichtintensität
 digitalWrite(outputPin, LOW);   // LED ausschalten
 delay(rate);
}
```

Siehe auch

In Rezept 3.5 finden Sie Details zur Verwendung der `constrain`-Funktion.

Rezept 5.7 beschreibt die `map`-Funktion.

Wenn Sie Töne erzeugen wollen, finden Sie in Kapitel 9 eine umfassende Diskussion zur Audioausgabe mit dem Arduino.

KAPITEL 2

Den Sketch machen lassen, was Sie wollen

2.0 Einführung

Auch wenn es bei einem Arduino-Projekte zu einem großen Teil darum geht, das Arduino-Board mit der passenden Hardware zu verbinden, müssen Sie dem Board sagen können, was es denn genau damit anfangen soll. Dieses Kapitel führt in die Kernelemente der Arduino-Programmierung ein, zeigt Nicht-Programmierern, wie man gängige Sprachkonstrukte nutzt, und enhält eine Sprachübersicht für diejenigen Leser, die nicht mit C oder C++ (der von Arduino verwendeten Sprache) vertraut sind.

Damit die Beispiele interessant bleiben, müssen wir Arduino etwas machen lassen. Die Rezepte nutzen daher physikalische Fähigkeiten des Boards, die erst in späteren Kapiteln im Detail erläutert werden. Falls Ihnen der Code in diesem Kapitel nicht klar ist, sollten Sie zu den nachfolgenden Kapiteln springen, insbesondere zu Kapitel 4 für die serielle Ausgabe und Kapitel 5 zum Einsatz der digitalen und analogen Pins. Sie müssen aber nicht alle Codebeispiele begreifen, um zu verstehen, wie die speziellen Fähigkeiten genutzt werden, die im Fokus des jeweiligen Rezepts stehen. Hier einige gängige Funktionen, die in den nächsten paar Kapiteln behandelt werden:

`Serial.println(wert);`
: Gibt den Wert über den seriellen Monitor der Arduino-IDE aus, so dass Sie die Arduino-Ausgabe am Computer verfolgen können; siehe Rezept 4.1.

`pinMode(pin, modus);`
: Konfiguriert einen digitalen Pin als Eingang (lesen) oder Ausgang (schreiben). Siehe hierzu die Einführung zu Kapitel 5.

`digitalRead(pin);`
: Liest einen digitalen Wert (`HIGH` oder `LOW`) über einen als Eingang festgelegten Pin ein; siehe Rezept 5.1.

`digitalWrite(pin, wert);`
: Schreibt einen digitalen Wert (`HIGH` oder `LOW`) an einen als Ausgang festgelegten Pin; siehe Rezept 5.1.

2.1 Strukturierung eines Arduino-Programms

Problem

Sie sind Programmiereinsteiger und wollen die Bausteine eines Arduino-Programms verstehen.

Lösung

Programme für den Arduino werden üblicherweise als Sketches bezeichnet. Die ersten Nutzer waren Künstler und Designer, und der Begriff *Sketch* (Skizze/Entwurf) hebt die schnelle und einfache Möglichkeit hervor, Ideen realisieren zu können. Die Begriffe *Sketch* und *Programm* sind austauschbar. Sketches enthalten Code, also die Instruktionen, die das Board ausführen soll. Nur einmal auszuführender Code (etwa die Initialisierung des Boards für die Anwendung), muss in der `setup`-Funktion stehen. Code, der kontinuierlich ausgeführt werden soll, nachdem das Setup abgeschlossen wurde, kommt in die `loop`-Funktion. Hier ein typischer Sketch:

```
const int ledPin = 13;   // Mit Pin 13 verbundene LED

// Die setup()-Methode wird beim Start des Sketches einmal ausgeführt
void setup()
{
 pinMode(ledPin, OUTPUT);    // Digitalen Pin als Ausgang festlegen
}

// Die loop()-Methode wird immer und immer wieder ausgeführt
void loop()
{
 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(1000);          // Eine Sekunde warten
 digitalWrite(ledPin, LOW);   // LED ausschalten
 delay(1000);          // Eine Sekunde warten
}
```

Sobald die Arduino-IDE den Code hochgeladen hat (und bei jedem Einschalten des Boards), beginnt es am Anfang des Sketches und geht die Instruktionen nacheinander durch. Es führt den Code in `setup` einmal aus und geht dann den Code in `loop` (engl. Schleife) durch. Am Ende von `loop` (den die schließende geschweifte Klammer } markiert), wird zum Anfang von `loop` zurückgesprungen.

Diskussion

Dieses Beispiel lässt die LED fortlaufend blinken, indem es die Werte `HIGH` und `LOW` an den Pin schreibt. Mehr zur Verwendung der Arduino-Pins finden Sie in Kapitel 5. Beim Start des Sketches legt der Code in `setup` den Modus des Pins fest (damit er die LED ein- und ausschalten kann). Nachdem der Code in `setup` ausgeführt wurde, wird der Code in `loop`, der die LED blinken lässt, in einer Endlosschleife ausgeführt, solange das Arduino-Board eingeschaltet ist.

Sie müssen Folgendes nicht wissen, um Arduino-Sketches zu schreiben, doch erfahrene C/C++-Programmierer werden sich fragen, wohin der Einsprungpunkt `main()` verschwunden ist. Er ist da, wird aber durch die Arduino-Build-Umgebung versteckt. Der Build-Prozess erzeugt eine Zwischendatei, die den Sketch-Code und die folgenden zusätzlichen Anweisungen enthält:

```
int main(void)
{
  init();

  setup();

  for (;;)
    loop();

  return 0;
}
```

Zuerst wird die Funktion `init()` aufgerufen, die die Arduino-Hardware initialisiert. Danach wird die `setup()`-Funktion des Sketches aufgerufen. Zum Schluss wird Ihre `loop()`-Funktion immer und immer wieder ausgeführt. Da die `for`-Schleife niemals endet, wird die `return`-Anweisung nie ausgeführt.

Siehe auch

Rezept 1.4 erklärt, wie man einen Sketch auf ein Arduino-Board hochlädt.

Kapitel 17 (steht als Download bereit) und *http://www.arduino.cc/en/Hacking/BuildProcess* enthalten weitere Informationen zum Build-Prozess.

2.2 Einfache primitive Typen (Variablen) nutzen

Problem

Arduino kennt verschiedene Variablentypen, um Werte effizient repräsentieren zu können. Sie wollen wissen, wie man diese Arduino-Datentypen wählt und nutzt.

Lösung

Der Datentyp `int` (kurz für *Integer*, bei Arduino ein 16-Bit-Wert) ist in Arduino-Anwendungen die gängige Wahl für numerische Werte. Sie können aber Tabelle 2-1 nutzen, um den Datentyp zu bestimmen, der für Ihre Anwendung den geeigneten Wertebereich aufweist.

Tabelle 2-1: Arduino-Datentypen

Numerische Typen	Bytes	Wertebereich	Verwendung
`int`	2	–32768 bis 32767	Repräsentiert positive und negative ganze Zahlen.
`unsigned int`	2	0 bis 65535	Repräsentiert nur positive ganze Zahlen. Ansonsten wie `int`.

Tabelle 2-1: Arduino-Datentypen (Fortsetzung)

Numerische Typen	Bytes	Wertebereich	Verwendung
`long`	4	–2147483648 bis 2147483647	Repräsentiert sehr große positive und negative ganze Zahlen.
`unsigned long`	4	4294967295	Repräsentiert sehr große positive ganze Zahlen.
`float`	4	3.4028235E+38 bis –3.4028235E+38	Repräsentiert Fließkommazahlen. Wird für Messwerte genutzt um Werte von Messungen an reale Werte anzunähern.
`double`	4	Wie `float`	Bei Arduino ist `double` nur ein anderer Name für `float`.
`boolean`	1	`false` (0) oder `true` (1)	Repräsentiert boolesch Wahr/Falsch-Werte.
`char`	1	–128 bis 127	Repräsentiert ein einzelnes Zeichen. Kann auch einen vorzeichenbehafteten Wert zwischen –128 und 127 repräsentieren.
`byte`	1	0 bis 255	Wie `char`, aber für vorzeichenlose Werte.

Weitere Typen	Verwendung
`String`	Repräsentiert Arrays von `chars` (Zeichen). Wird üblicherweise für Text verwendet.
`void`	Wird nur bei Funktionsdeklarationen verwendet, die keinen Wert zurückliefern.

Diskussion

Solange man keine maximale Performance oder Speichereffizienz braucht, eignen sich als `int` deklarierte Variablen für numerische Werte, wenn sie den Wertebereich (in der ersten Spalte in Tabelle 2-1) nicht überschreiten und man keine Brüche (Fließkommazahlen) braucht. Die meisten offiziellen Arduino-Codebeispiele deklarieren numerische Variablen als `int`. Doch manchmal müssen Sie einen Typ wählen, der die speziellen Anforderungen Ihrer Anwendung erfüllt.

Manchmal braucht man negative Zahlen und manchmal nicht, weshalb es zwei Varianten numerischer Variablen gibt: `signed` (mit Vorzeichen) und `unsigned` (ohne Vorzeichen). `unsigned`-Werte sind immer positiv. Variablen, denen nicht das Schlüsselwort `unsigned` vorangestellt wird, arbeiten mit Vorzeichen, d.h., sie können negative und positive Werte repräsentieren. Ein Grund für die Verwendung von `unsigned`-Werten ist, dass die Werte möglicherweise nicht in `signed`-Variablen passen (der vorzeichenlose, maximale Wert einer unsigned-Variable ist doppelt so hoch wie der einer signed-Variablen). Ein weiterer Grund, warum Programmierer `unsigned`-Typen verwenden, liegt darin, möglichen Lesern des Codes klar anzuzeigen, dass der Wert einer Variablen niemals negativ wird.

`Boolesche` Typen besitzen nur zwei mögliche Werte: wahr (`true`) oder falsch (`false`). Sie werden üblicherweise verwendet, um solche Dinge zu prüfen wie den Status eines Schalters (Wurde er gedrückt oder nicht?). An den Stellen, an denen es sinnvoll ist, können Sie auch `HIGH` und `LOW` als Äquivalent zu `true` und `false` nutzen. `digitalWrite(pin, HIGH)` ist für das Einschalten aussagekräftiger als `digitalWrite(pin, true)` oder `digitalWrite(pin,1)`, auch wenn alle Varianten, was die Ausführung des Sketches betrifft, identisch sind. Bei im Web gepostetem Code werden Ihnen sehr wahrscheinlich alle Varianten begegnen.

Siehe auch

Details zu den Datentypen finden Sie in der Arduino-Referenz an *http://www.arduino.cc/en/Reference/HomePage* .

2.3 Fließkommazahlen verwenden

Problem

Fließkommazahlen werden für Werte mit Nachkommastellen (also Brüche) verwendet. Sie möchten solche Werte in Ihren Sketches nutzen.

Lösung

Der folgende Code zeigt, wie man Fließkomma-Variablen deklariert, illustriert mögliche Probleme beim Vergleich von Fließkomma-Werten und zeigt, wie man sie vermeiden kann. Beachten Sie bitte, dass das Komma im Amerikanischen als Punkt (.) dargestellt wird:

```
/*
 * Fließkomma-Beispiel
 * Der Sketch initialisiert eine Variable mit
 * dem Fließkommawert 1,1 und verringert ihn
 * fortlaufend um 0,1, bis der Wert 0 erreicht ist.
 */


float value = 1.1;

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

void loop()
{
 value = value - 0.1;  // Reduziere value bei jedem Durchlauf um 0,1.
 if( value == 0)
   Serial.println("Der Wert ist genau 0");
 else if(almostEqual(value, 0))
 {
  Serial.print("Der Wert ");
  Serial.print(value,7); // 7 Dezimalstellen ausgeben
  Serial.println(" ist fast 0");
 }
 else
  Serial.println(value);

 delay(100);
}

// Gibt wahr zurück, wenn die Differenz zwischen a und b klein ist.
// Der Wert von DELTA gibt die max. Differenz an, die noch als "gleich" betrachtet wird.
boolean almostEqual(float a, float b)
```

```
{
 const float DELTA = .00001; // Max. Differenz, die noch "gleich" ist
 if (a == 0) return fabs(b) <= DELTA;
 if (b == 0) return fabs(a) <= DELTA;
 return fabs((a - b) / max(fabs(a), fabs(b))) <= DELTA ;
}
```

Diskussion

Fließkommazahlen sind nicht genau, und die zurückgelieferten Werte können kleine Rundungsfehler enthalten. Diese Fehler treten auf, weil Fließkommazahlen einen großen Wertebereich abdecken, weshalb die interne Repräsentation des Wertes nur eine Näherung sein kann. Aus diesem Grund dürfen Sie nicht auf exakte Übereinstimmung testen, sondern ob die Werte innerhalb eines Tolerlanzbereichs liegen.

Der Sketch gibt auf dem seriellen Monitor folgendes aus:

```
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
Der Wert -0.0000001 ist fast 0
-0.10
-0.20
```

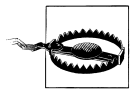
Der serielle Monitor setzt die Ausgabe negativer Zahlen fort.

Sie würden wohl erwarten, dass der Code `"Der Wert ist genau 0"` ausgibt, nachdem `value` `0,1` enthält und dann erneut 0,1 subtrahiert wird. Doch `value` ist nie genau Null. Der Wert ist nah dran, aber nie nah genug, um den Test `if (value == 0)` zu bestehen. Das liegt daran, dass der einzige speichereffiziente Weg zur Speicherung von Fließkommazahlen (die einen riesigen Wertebereich abdecken) darin besteht, eine Näherung der Zahl zu speichern.

Die Lösung besteht (wie in diesem Rezept zu sehen) darin, zu prüfen, ob die Variable nah genug am gewünschten Wert liegt.

Die Funktion `almostEqual` prüft, ob die Variable `value` im Bereich von 0,00001 des gewünschten Zielwertes liegt und gibt wahr zurück, wenn das so ist. Der akzeptable Wertebereich wird in der Konstanten `DELTA` festgelegt, damit Sie ganz nach Bedarf größere oder kleinere Werte einstellen können. Die Funktion `fabs` (eine Abkürzung für *Fließkomma-Absolutwert*) gibt den Absolutwert einer Fließkommazahl zurück, der dann genutzt wird, um die Differenz der angegebenen Parameter zu überprüfen.

Fließkommazahlen sind nur Näherungen, weil nur 32 Bit zur Verfügung stehen, um alle Werte eines riesigen Wertebereichs aufzunehmen. Acht Bit werden für den dezimalen Multiplikator (den Exponenten) verwendet, bleiben 24 Bit für das Vorzeichen und den eigentlichen Wert – gerade genug für sieben signifikate Nachkommastellen.



Zwar sind `float` und `double` bei Arduino identisch, doch bei vielen anderen Plattformen bietet `double` eine höhere Genauigkeit. Wenn Sie Code von anderen Plattfomen importieren, der `float` und `double` mischt, müssen Sie sicherstellen, dass die Genauigkeit für Ihre Anwendung ausreicht.

Siehe auch

Die Arduino-Referenz zu `float`: *http://www.arduino.cc/en/Reference/Float*.

2.4 Mit Gruppen von Werten arbeiten

Problem

Sie wollen eine Gruppe von Werten (ein sog. *Array*) anlegen und nutzen. Arrays können einfache Listen sein, aber auch zwei oder mehr Dimensionen aufweisen. Sie wollen wissen, wie man die Größe des Arrays bestimmt und auf die Elemente des Arrays zugreift.

Lösung

Dieser Sketch verwendet zwei Arrays. Ein Array mit Pins, an denen Schalter angeschlossen sind, und ein zweites Array mit Pins, die mit LEDs verbunden sind (siehe Abbildung 2-1):

```
/*
 array-Sketch
 Ein Array von Schaltern steuert ein Array von LEDs
 in Kapitel 5 erfahren Sie mehr über Schalter
 Informationen zu LEDs finden Sie in Kapitel 7
 */

int inputPins[] = {2,3,4,5}; // Array mit Pins für die Schalter anlegen (Eingänge)

int ledPins[] = {10,11,12,13}; // Array mit Pins für die LEDs anlegen (Ausgänge)

void setup()
{
 for(int index = 0; index < 4; index++)
 {
  pinMode(ledPins[index], OUTPUT);      // LED als Ausgang deklarieren
  pinMode(inputPins[index], INPUT);     // Schalter als Eingang deklarieren

  digitalWrite(inputPins[index],HIGH);   // Pullup-Widerstände aktivieren
  // (siehe Rezept 5.2)
 }
```

```
}

void loop(){
 for(int index = 0; index < 4; index++)
 {
  int val = digitalRead(inputPins[index]);  // Eingabe einlesen
  if (val == LOW)                 // Prüfen, ob Schalter auf Ein steht
  {
   digitalWrite(ledPins[index], HIGH); // LED einschalten, wenn Schalter auf Ein steht,
  }
  else
  {
   digitalWrite(ledPins[index], LOW);  // sonst LED ausschalten
  }
 }
}
```

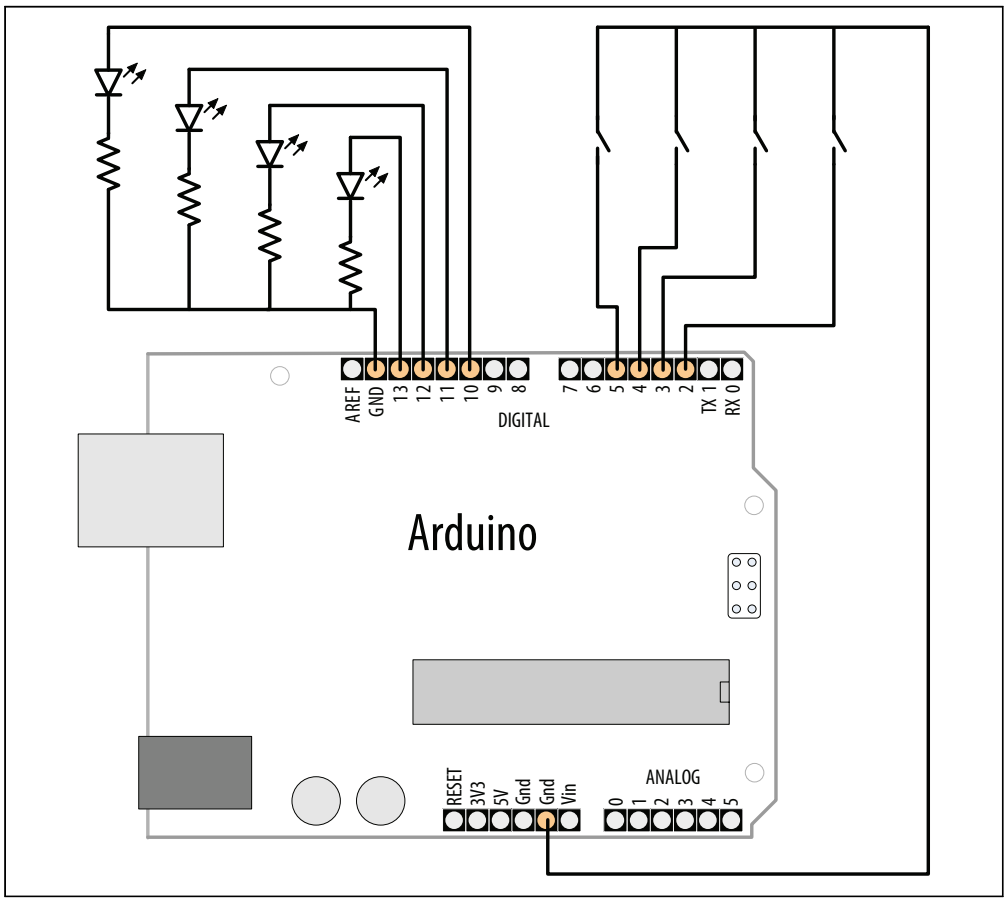


Abbildung 2-1: Verbindungen für LEDs und Schalter

Diskussion

Arrays sind Gruppen aufeinanderfolgender Variablen des gleichen Typs. Jede Variable innerhalb der Gruppe wird als *Element* bezeichnet. Die Anzahl der Elemente nennt man die *Größe* des Arrays.

Die Lösung zeigt ein typisches Einsatzgebiet von Arrays im Arduino-Code: die Speicherung einer Gruppe von Pins. Die Pins sind hier mit Schaltern und LEDs verbunden (ein Thema, dem wir uns in Kapitel 5 ausführlicher widmen). Die wichtigen Teile dieses Beispiels sind die Deklaration des Arrays und der Zugriff auf die Array-Elemente.

Die folgende Codezeile deklariert (erzeugt) ein Array aus vier Integer-Elementen und initialisiert jedes Element. Das erste Element wird auf 2 gesetzt, das zweite auf 3 und so weiter:

```
int inputPins[] = {2,3,4,5};
```

Wenn Sie die Werte bei der Deklaration des Arrays nicht initialisieren (z.B. weil sie erst im laufenden Sketch verfügbar sind), müssen Sie jedes Element einzeln ändern. Sie deklarieren das Array wie folgt:

```
int inputPins[4];
```

Das deklariert ein Array mit vier Elementen und setzt den Wert jedes Elements auf Null. Die Zahl innerhalb der eckigen Klammern ([]) ist die Größe, legt also die Anzahl der Elemente fest. Dieses Array hat die Größe 4 und kann daher höchstens vier Integerwerte aufnehmen. Sie können die Größe weglassen, wenn die Array-Deklaration die Werte initialisiert (wie im ersten Beispiel). Der Compiler weiß dann, wie groß das Array werden muss, indem er die Zahl der Initialisierungen mitzählt.

Das erste Element des Arrays ist `element[0]`:

```
int firstElement = inputPins[0]; // Das erste Element

inputPins[0] = 2; // Setzt den Wert dieses Elements auf 2
```

Das letzte Element ist 1 kleiner als die Größe des Arrays. Im obigen Beispiel mit der Größe 4 ist Element 3 das letzte Element:

```
int lastElement = inputPins[3]; // Das letzte Element
```

Sie wundern sich vielleicht, warum bei einem Array der Größe 4 das letzte Element über `array[3]` angesprochen wird, doch da `array[0]` das erste Element ist, heißen die vier Elemente:

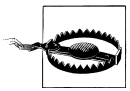
```
inputPins[0],inputPins[1],inputPins[2],inputPins[3]
```

Im obigen Sketch erfolgt der Zugriff auf die vier Elemente in einer `for`-Schleife:

```
for(int index = 0; index < 4; index++)
{
  //Pin durch Zugriff auf jedes Element des Pin-Arrays ermitteln
  pinMode(ledPins[index], OUTPUT);      // LED als Ausgang deklarieren
  pinMode(inputPins[index], INPUT);     // Schalter als Eingang deklarieren
}
```

Dieses Schleife durchläuft die Variable `index` von 0 bis 3. Ein typischer Fehler besteht darin, versehentlich auf ein Element zuzugreifen, das außerhalb der Größe des Arrays liegt. Solche Fehler können verschiedene Symptome aufweisen, und Sie müssen darauf achten, sie zu vermeiden. Eine Möglichkeit, solche Schleifen zu kontrollieren, besteht darin, die Größe des Arrays in einer Konstanten festzuhalten:

```
const int PIN_COUNT = 4; // Konstante für Anzahl der Elemente definieren
int inputPins[PIN_COUNT] = {2,3,4,5};

 for(int index = 0; index < PIN_COUNT; index++)
  pinMode(inputPins[index], INPUT);
```



Der Compiler meldet keinen Fehler, wenn Sie versehentlich einen Wert außerhalb der Grenzen des Arrays lesen oder schreiben. Sie müssen daher sorgfältig darauf achten, dass Sie nur auf Elemente innerhalb der von Ihnen gesetzten Grenzen zugreifen. Die Verwendung einer Konstanten für die Größe des Arrays und im Code hilft Ihnen dabei, sich innerhalb der Array-Grenzen zu bewegen.

Ein weiteres Einsatzgebiet für Arrays ist das Vorhalten einzelner Textzeichen. Im Arduino-Code werden sie als *Zeichenketten* (oder *Strings*) bezeichnet. Eine Zeichenkette besteht aus einem oder mehreren Zeichen, die mit einem Nullzeichen (dem Wert 0) abgeschlossen werden.

Die Null am Ende des Strings ist nicht mit dem Zeichen 0 identisch. Die Null hat den ASCII-Wert 0, während 0 den ASCII-Wert 48 hat.

Methoden zur Verwendung von Strings werden in 2.5 und 2.6 behandelt.

Siehe auch

Rezept 5.2; Rezept 7.1

2.5 Arduino-Stringfunktionen nutzen

Problem

Sie wollen Text bearbeiten. Sie wollen ihn kopieren, verketten oder die Anzahl der Zeichen bestimmen.

Lösung

Das vorige Kapitel hat kurz angedeutet, wie man Arrays von Zeichen nutzt, um Text zu speichern. Solche Zeichen-Arrays werden üblicherweise als Strings bezeichnet. Arduino besitzt eine `String`-Bibliothek, die eine Vielzahl von Funktionen zur Speicherung und Bearbeitung von Text zur Verfügung stellt.

Der Ausdruck *String* mit einem *S* als Großbuchstabe bezieht sich auf die Arduino-Text-Funktion, die von der Arduino-String-Bibliothek geliefert wird. Das Wort string mit dem Kleinbuchstaben s bezieht sich eher auf Zeichengruppen als auf die Arduino-String-Funktionalität.

Dieses Rezept zeigt, wie man mit Arduino-`String`s arbeitet.

Die `String`-Bibliothek wurde mit der Version 0019 alpha (älter als 1.0) von Arduino eingeführt. Wenn Sie eine ältere Version nutzen, können Sie die TextString-Bibliothek verwenden. Beachten Sie hierzu den Link am Ende des Rezepts.

Laden Sie den folgenden Sketch auf Ihr Board und öffnen Sie den seriellen Monitor, um sich die Ergebnisse anzusehen:

```
/*
 Basic_Strings-Sketch
 */

String text1 = "Dieser String";
String text2 = "hat mehr Text";
String text3; // Wird im Sketch zugewiesen

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

 Serial.print( text1);
 Serial.print(" ist ");
 Serial.print(text1.length());
 Serial.println(" Zeichen lang. ");

 Serial.print("text2 ist ");
 Serial.print(text2.length());
 Serial.println(" Zeichen lang. ");

 text1.concat(text2);
 Serial.println("text1 enthaelt nun: ");
 Serial.println(text1);
}

void loop()
{
}
```

Diskussion

Dieser Sketch erzeugt drei Variablen namens `text1`, `text2` und `text3` vom Typ `String`. Variablen vom Typ `String` besitzen Fähigkeiten zur Bearbeitung von Text. Die Anweisung `text1.length()` gibt die Länge (Anzahl der Zeichen) des Strings `text1` zurück.

`text1.concat(text2)`kombiniert die Inhalte von Strings. In unserem Beispiel wird der Inhalt von `text2` an das Ende von `text1` angehangen (`concat` ist die Abkürzung von *concatenate,* zu deutsch verketten).

Der serielle Monitor gibt Folgendes aus:

```
Dieser String ist 13 Zeichen lang.
text2 ist 14 Zeichen lang.
text1 enhält nun:
Dieser String hat mehr Text
```

Eine weitere Möglichkeit zur Verkettung von Strings bietet der Additionsoperator. Hängen Sie die folgenden beiden Zeilen an das Ende des `setup`-Codes an:

```
text3 = text1 + " und mehr";
Serial.println(text3);
```

Der neue Code gibt im seriellen Monitor zusätzlich noch die folgende Zeile aus:

```
Dieser String hat mehr Text und mehr
```

Sie können die Funktionen `indexOf` und `lastIndexOf` nutzen, um das Vorkommen eines bestimmten Zeichens in einem String zu finden.

Da die `String`-Klasse eine recht junge Arduino-Erweiterung ist, werden Sie viel Code sehen, der statt des `String`-Typs mit Zeichenketten arbeitet. Weitere Informationen zur Verwendung von Zeichenketten anstelle von Arduino-`String`s finden Sie in Rezept 2.6.

Wenn Sie eine Zeile wie die folgende sehen:

```
char oldString[] = "Dies ist eine Zeichenkette";
```

dann verwendet der Code Zeichenketten im C-Stil (siehe Rezept 2.6). Sieht die Deklaration hingegen so aus:

```
String newString = "Dies ist ein String-Objekt";
```

verwendet der Code Arduino-`String`s. Um eine Zeichenkette im C-Stil einem Arduino-`String` zuzuweisen, weisen Sie einfach den Inhalt des Arrays einem `String`-Objekt zu:

```
char oldString[] = "Diese Zeichenkette soll ein String-Objekt sein";
String newString = oldString;
```

Um die in der Tabelle 2-2 aufgeführten Funktionen verwenden zu können, müssen Sie sie auf ein existierendes String-Objekt anwenden, wie im folgenden Beispiel gezeigt wird:

```
int len = myString.length();
```

Tabelle 2-2: Kurze Übersicht der Arduino String-Funktionen

`charAt(n)`	Gibt das *n*-te Zeichen des `String`s zurück.
`compareTo(S2)`	Vergleicht den `String` mit dem angegebenen `String` S2
`concat(S2)`	Gibt einen neuen `String` zurück, bei dem der `String` und S2 verkettet sind.
`endsWith(S2)`	Gibt wahr zurück, wenn der `String` mit den Zeichen in S2 endet.

Tabelle 2-2: Kurze Übersicht der Arduino String-Funktionen (Fortsetzung)

`equals(S2)`	Gibt wahr zurück, wenn der `String` genau mit S2 übereinstimmt (Groß-/Kleinschreibung wird beachtet).
`equalsIgnoreCase(S2)`	Wie `equals`, ignoriert aber die Groß-/Kleinschreibung.
`getBytes(buffer,len)`	Kopiert `len` Zeichen in den angegebenen Bytepuffer.
`indexOf(S)`	Gibt den Index des angegebenen `String`s (oder Zeichens) zurück bzw. –1, wenn er nicht gefunden wird.
`lastIndexOf(S)`	Wie `indexOf`, beginnt aber mit dem Ende des `String`s.
`length()`	Gibt die Anzahl der Zeichen im `String` zurück.
`replace(A,B)`	Ersetzt alle Instanzen von `String` (oder Zeichen) A durch B.
`setCharAt(index,c)`	Speichert das Zeichen c am angegebenen Index im `String`.
`startsWith(S2)`	Gibt wahr zurück, wenn der `String` mit den Zeichen in S2 beginnt.
`substring(index)`	Gibt einen `String` mit den Zeichen beginnend bei Index bis zum Ende des `String`s zurück.
`substring(index,to)`	Wie oben, aber der Substring endet vor der Zeichenposition \9to\9.
`toCharArray(buffer,len)`	Kopiert bis zu `len` Zeichen des `String`s in den angegebenen Puffer.
`toInt()`	Gibt den Integerwert der im `String` stehenen Ziffern zurück.
`toLowerCase()`	Gibt einen `String` zurück, bei dem alle Zeichen in Kleinbuchstaben umgewandelt wurden.
`toUpperCase()`	Gibt einen `String` zurück, bei dem alle Zeichen in Großbuchstaben umgewandelt wurden.
`trim()`	Gibt einen `String` zurück, bei dem alle führenden und anhängenden Whitespaces entfernt wurden.

Weitere Hinweise zur Nutzung und den Varianten dieser Funktionen finden Sie auf den Arduino-Referenzseiten.

Zwischen Arduino-Strings und C-Zeichenketten wählen

Arduinos fest eingebauter Datentyp `String` ist einfacher zu nutzen als C-Zeichenketten, was aber durch komplexen Code in der String-Bibliothek erreicht wird, die größere Anforderungen an Ihren Arduino stellt und naturgemäß problematischer ist.

Der `String`-Datentyp ist so flexibel, weil er die *dynamische Speicherallozierung* nutzt. Wenn Sie also einen `String` anlegen oder modifizieren, fordert Arduino einen neuen Speicherbereich von der C-Bibliothek an. Benötigen Sie den `String` nicht länger, muss Arduino den Speicher wieder freigeben. Das läuft üblicherweise sauber, doch in der Praxis gibt es viele Stellen, an denen der Speicher ein »Leck« haben kann. Fehler in der String-Bibliothek können dazu führen, dass ein Teil oder der gesamte Speicher nicht freigegeben wird. Wenn das passiert, steht dem Arduino (bis zum Neustart) mit der Zeit immer weniger Speicher zur Verfügung. Und selbst wenn es kein Speicherleck gibt, ist es schwierig, Code zu schreiben, der prüft, ob ein String aufgrund unzureichenden Speichers nicht angefordert werden konnte (die `String`-Funktionen imitieren die von Processing, doch im Gegensatz zu dieser Plattform verfügt Arduino nicht über eine Ausnahmebehandlung von Laufzeitfehlern. Fehler mit dynamischem Speicher lassen sich nur schwer finden, da der Sketch Tage oder Wochen problemlos laufen kann, bevor es zu einem Fehlverhalten kommt.

Wenn Sie C-Zeichenketten nutzen, haben Sie die Kontrolle über die Speichernutzung: Sie allozieren eine feste (statische) Speichermenge während der Kompilierung, so dass es nicht zu einem Speicherleck kommen kann. Ihrem Arduino-Sketch steht bei jedem Lauf die gleiche Speichermenge zur Verfügung. Und wenn Sie versuchen mehr Speicher zu allozieren, als Ihnen zur Verfügung steht, ist der Fehler leichter zu ermitteln, da es Tools gibt, die Ihnen sagen, wie viel statischen Speicher Sie alloziert haben (siehe hierzu die Referenz zu `avr-objdump` in Rezept 17.1, als Download).

Allerdings kann es bei C-Zeichenketten leicht zu einem anderen Problem kommen: C hindert Sie nicht daran, Speicher zu ändern, der außerhalb der Grenzen des Arrays liegt. Wenn Sie also ein Array mit `myString[4]` allozieren und `myString[4] = 'A'` zuweisen (denken Sie daran, dass `myString[3]` das Ende des Arrays ist), wird Sie niemand daran hindern. Doch wer weiß, auf welche Speicherstelle `myString[4]` verweist? Und wer weiß schon, zu welchem Problem die Zuweisung von `'A'` an diese Speicherstelle führt? Sehr wahrscheinlich wird es ein Fehlverhalten des Sketches verursachen.

Bei Arduinos fest eingebauter String-Bibliothek laufen Sie also durch die Nutzung dynamischen Speichers Gefahr, den verfügbaren Speicher aufzufressen. Bei C-Zeichenketten müssen Sie darauf achten, die Grenzen des verwendeten Arrays nicht zu überschreiten. Nutzen Sie Arduinos fest eingebaute String-Bibliothek dann, wenn Sie die umfassende Möglichkeiten zur Textbearbeitung brauchen und `String`s nicht immer wieder neu erzeugen. Werden sie in einer Schleife wiederholt erzeugt und modifiziert, allozieren Sie besser ein großes C-Zeichen-Array und entwickeln den Code sorgfältig, damit Sie die Grenzen des Arrays nicht verlassen.

Ein weiterer Fall, bei dem man C-Zeichenketten gegenüber Arduino-`String`s vorzieht, sind große Sketches, die einen Großteil des verfügbaren Speichers oder Flashspeichers nutzen. Der Arduino `StringToInt`-Beispielcode benötigt fast 2 KB mehr Flash als das Äquivalent mit C-Zeichenketten und der `atoi`-Funktion zur Umwandlung in einen `int`-Wert. Die Arduino `String`-Version benötigt außerdem etwas mehr RAM, um zusätzlich zum String noch Allozierungsinformationen zu speichern.

Wenn Sie den Verdacht haben, dass die String-Bibliothek oder jede andere Bibliothek, die dynamisch Speicher alloziert, ein Speicherleck hat, können Sie jederzeit den freien Speicher ermitteln. Siehe Rezept 17.2. Prüfen Sie den freien Speicher beim Start des Sketches und überwachen Sie, ob er mit der Zeit abnimmt. Wenn Sie ein Problem mit der String-Bibliothek vermuten, suchen Sie in der Liste offener Bugs (*http://code.google.com/p/arduino/issues/list*) nach »String«.

Siehe auch

Die Arduino-Distribution enthält `String`-Beispielsketches (File→Examples→Strings).

Die `String`-Referenzseite finden Sie unter *http://arduino.cc/en/Reference/StringObject*.

Einführungen zur neuen String-Bibliothek finden Sie unter *http://arduino.cc/en/Tutorial/HomePage*. Eine Einführung in die Original-String-Bibliothek (die Sie nur benötigen,

wenn Sie mit einer Arduino-Version älter als 0019 alpha arbeiten) finden Sie unter *http://www.arduino.cc/en/Tutorial/TextString*.

2.6 C-Zeichenketten nutzen

Problem

Sie wollen verstehen, wie man mit Zeichenketten arbeitet. Sie wollen wissen, wie man einen solchen String erzeugt, seine Länge bestimmt, ihn vergleicht, kopiert und anhängt. Der Sprachkern von C unterstützt die `String`-Fähigkeiten von Arduino nicht, weshalb Sie den für andere Plattformen entwickelten Code verstehen wollen, der mit primitiven Zeichenketten arbeitet.

Lösung

Arrays von Zeichen, werden auch *Zeichenketten* (oder einfach *Strings*) genannt. Rezept 2.4 beschreibt Arduino-Arrays im Allgemeinen. Dieses Rezept beschreibt Funktionen, die mit Zeichenketten arbeiten.

Sie deklarieren Strings wie folgt:

```
char stringA[8]; // Deklariere einen String mit bis zu 7 Zeichen plus abschließender Null
char stringB[8] = "Arduino"; // Wie oben, init(inialisiert) den String aber gleich mit "Arduino"
char stringC[16] = "Arduino"; // Wie oben, aber der String kann wachsen
char stringD[ ] = "Arduino"; // Der Compiler initialisiert den String und errechnet seine Größe
```

Verwenden Sie `strlen` (eine Abkürzung für *string length*, also Stringlänge), um die Anzahl der Zeichen vor der abschließenden Null zu bestimmen:

```
int length = strlen(string); // Gibt die Anzahl der Zeichen im String zurück
```

`length` ist im obigen Beispiel 0 für `stringA` und 7 für die anderen Strings. Die Null, die das Ende des Strings festlegt, wird von `strlen` nicht mitgezählt.

Verwenden Sie `strcpy` (*string copy*), um einen String in einen anderen zu kopieren:

```
strcpy(destination, source);  // Kopiert den String von der Quelle (source) an das Ziel
(destination)
```

Verwenden Sie `strncpy`, um die Zahl der zu kopierenden Zeichen zu beschränken. (Nützlich, um nicht mehr Zeichen zu kopieren, als der Zielstring aufnehmen kann.) Sie können sie in Rezept 2.7 im Einsatz sehen:

```
// Kopiere bis zu 6 Zeichen von der Quelle (source) zum Ziel(destination)
strncpy(destination, source, 6);
```

Verwenden Sie `strcat` (*string concatenate*), um einen String an das Ende eines anderen anzuhängen:

```
// Quellstring an das Ende des Zielstrings anhängen
strcat(destination, source);
```

Achten Sie beim Kopieren oder Verketten immer darauf, dass das Ziel ausreichend Platz hat. Denken Sie auch an den Platz für die abschließende Null.

Verwenden Sie `strcmp` (*string compare*), um zwei Strings zu vergleichen. Ein Anwendungsbeispiel finden Sie in Rezept 2.7:

```
if(strcmp(str "Arduino") == 0)
  // Mach etwas, wenn die Variable str den String "Arduino" enthält
```

Diskussion

Text wird in der Arduino-Umgebung durch ein Array von Zeichen, sog. Strings, repräsentiert. Ein String besteht aus einer Folge von Zeichen, die mit einer Null (dem Wert 0) abgeschlossen wird. Die Null wird nicht ausgegeben, wird aber benötigt, um der Software das Ende des Strings anzuzeigen.

Siehe auch

Eine der vielen Online verfügbaren C/C++-Referenzseiten, etwa *http://www.cplusplus.com/reference/clibrary/cstring/* und *http://www.cppreference.com/wiki/string/c/start*.

2.7 Durch Komma getrennten Text in Gruppen aufteilen

Problem

Ein String enthält zwei oder mehr Datenelemente, die durch Kommata (oder ein anderes Trennzeichen) voneinander getrennt sind. Sie wollen den String so zerlegen, dass Sie die einzelnen Elemente nutzen können.

Lösung

Dieser Sketch gibt den Text zwischen den Kommata aus:

```
/*
 * SplitSplit-Sketch
 * Kommaseparierten String zerlegen
 */

String text = "Peter,Paul,Mary"; // Beispiel-String
String message = text; // Enthält noch nicht zerlegten Text
int   commaPosition; // Position des nächsten Kommas im String

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

 Serial.println(message);      // Quellstring ausgeben,
 do
```

```
 {
   commaPosition = message.indexOf(',');
   if(commaPosition != -1)
   {
     Serial.println( message.substring(0,commaPosition));
     message = message.substring(commaPosition+1, message.length());
   }
   else
   { // nachdem das letzte Komma gefunden wurde
     if(message.length() > 0)
      Serial.println(message); // Wenn Text auf das letzte Komma folgt,
                    // ausgeben
   }
  }
  while(commaPosition >=0);
}

void loop()
{
}
```

Im seriellen Monitor wird Folgendes ausgegeben:

```
Peter,Paul,Mary
Peter
Paul
Mary
```

Diskussion

Dieser Sketch nutzt `String`-Funktionen, um den Text zwischen den Kommata zu extrahieren. Der Code:

```
commaPosition = message.indexOf(',');
```

legt in der Variablen `commaPosition` die Position des ersten Kommas im `String` namens `message` ab (der Wert ist –1, wenn kein Komma gefunden wurde). Gibt es ein Komma, wird die Funktion `substring` genutzt, um den Text vom Anfang des Strings bis zum Komma auszugeben. Der ausgegebene Text samt Komma wird dann in der folgenden Zeile aus `message` entfernt:

```
message = message.substring(commaPosition+1, message.length());
```

`substring` liefert einen String zurück, der bei `commaPosition+1` (der Position gleich hinter dem Komma) beginnt und sich bis zum Ende des Strings erstreckt. Message enthält dann nur noch den Text, der auf das erste Komma folgt. Das wird so lange wiederholt, bis kein Komma mehr gefunden wird (`commaPosition` enthält dann –1).

Als erfahrener Programmierer können Sie auch die Low-Level-Funktionen nutzen, die Teil der Standard-C-Bibliothek sind. Der folgende Sketch bietet die gleiche Funktionalität wie oben mit Arduino-Strings:

```
/*
 * SplitSplit Sketch
 * Komma separieren, String zerlegen
```

```
 */

const int  MAX_STRING_LEN = 20; // Längsten String festlegen,
                  // den Sie verarbeiten wollen

char stringList[] = "Peter,Paul,Mary"; // Beispiel-String

char stringBuffer[MAX_STRING_LEN+1]; // Statischer Puffer für Berechnung und Ausgabe

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

void loop()
{
 char *str;
 char *p;
 strncpy(stringBuffer, stringList, MAX_STRING_LEN); // Quellstring kopieren
 Serial.println(stringBuffer);            // Quellstring ausgeben

 for( str = strtok_r(stringBuffer, ",", &p);     // An Komma zerlegen,
    str;                       // solange str nicht Null ist,
    str = strtok_r(NULL, ",", &p)          // nächstes Token ermitteln
   )
 {
  Serial.println(str);
 }
  delay(5000);
}
```

Die Kernfunktionalität liefert eine Funktion namens `strtok_r` (der Name der `strtok`-Version, die der Arduino-Compiler bereitstellt). Beim ersten Aufruf von `strtok_r` übergeben Sie den String, den Sie in *Token* (einzelne Werte) zerlegen wollen. Doch `strtok_r` überschreibt die Zeichen in diesem String jedes Mal, wenn es ein neues Token findet, weshalb man (wie in diesem Beispiel gezeigt) am besten eine Kopie des Strings übergibt. Alle nachfolgenden Aufrufe übergeben eine `NULL`, um die Funktion anzuweisen, sich zum nächsten Token zu bewegen. Im obigen Beispiel wird jedes Token über den seriellen Port ausgegeben.

Wenn Ihre Token nur aus Zahlen bestehen, sehen Sie sich Rezept 4.5 an. Es zeigt, wie man numerische Werte, die durch Kommata voneinander getrennt sind, aus einem Stream serieller Zeichen extrahiert.

Siehe auch

Unter *http://www.nongnu.org/avr-libc/user-manual/group__avr__string.html* erfahren Sie mehr über C-Stringfunktionen wie `strtok_r` und `strcmp`.

Rezept 2.5; Online- Referenzen zu den C/C++-Funktionen `strtok_r` und `strcmp`.

2.8 Eine Zahl in einen String umwandeln

Problem

Sie müssen eine Zahl in einen String umwandeln, etwa um sie auf einem LCD- oder einem anderen Display auszugeben.

Lösung

Die `String`-Variable wandelt Zahlen automatisch in Strings um. Sie können Literale oder den Inhalt einer Variablen verwenden. So funktioniert der folgende Code:

```
String myNumber = 1234;
```

Ebenso wie dieser:

```
int value = 127;
String myReadout = "Der Messwert ist ";
myReadout.concat(value);
```

Und auch dieser:

```
int value = 127;
String myReadout = "Der Messwert ist ";
myReadout += value;
```

Diskussion

Wenn Sie eine Zahl in Text umwandeln wollen, um sie auf einem LC-Display oder über eine serielle Schnittstelle auszugeben, besteht die einfachste Lösung darin, die Konvertierungsfähigkeiten zu nutzen, die in die LCD- oder Serial-Bibliotheken integriert sind (siehe Rezept 4.2). Doch vielleicht arbeiten Sie mit einem Gerät, in das die entsprechende Unterstützung nicht integriert ist (siehe Kapitel 13), oder Sie wollen die Zahl in Form eines Strings in Ihrem Sketch weiterverarbeiten.

Die Arduino-Klasse `String` wandelt numerische Werte automatisch um, wenn sie einer `String`-Variablen zugewiesen werden. Sie können numerische Werte verketten, indem Sie die Funktion `concat`, oder den Stringoperator + nutzen.

Der Operator + wird sowohl bei Zahlen als auch bei Strings verwendet, verhält sich aber leicht unterschiedlich.

Im folgenden Code erhält `number` den Wert `13`:

```
int number = 12;
number += 1;
```

Bei einem `String`

```
String textNumber = "12";
textNumber += 1;
```

enthält `textNumber` den Textstring `"121"`.

Vor der Einführung der `String`-Klasse hat Arduino-Code üblichereise die Funktionen `itoa` oder `ltoa` verwendet. Die Namen stehen für »integer to ASCII« (`itoa`) und »long to ASCII« (`ltoa`). Die vorhin beschriebene `String`-Version ist einfacher zu nutzen, doch wenn Sie lieber mit C-Zeichenketten arbeiten (siehe Rezept 2.6), können Sie den folgenden Code verwenden.

`itoa` und `ltoa` verlangen drei Parameter: den umzuwandelnden Wert, einen Puffer, der den Ausgabestring aufnimmt und die Basis (10 für Dezimal-, 16 für Hexadezimal und 2 für Binärzahlen).

Der folgende Sketch zeigt, wie man numerische Werte mit `ltoa` konvertiert:

```
/*
 * NumberToString
 * Erzeugt einen String aus einer Zahl
 */

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

char buffer[12]; // Datentyp long hat 11 Zeichen (inklusive
          // Minuszeichen) und ein abschließendes Null-Zeichen
void loop()
{
 long value = 12345;
 ltoa(value, buffer, 10);
 Serial.print( value);
 Serial.print(" hat ");
 Serial.print(strlen(buffer));
 Serial.println(" Ziffern");
 value = 123456789;
 ltoa(value, buffer, 10);
 Serial.print( value);
 Serial.print(" hat ");
 Serial.print(strlen(buffer));
 Serial.println(" Ziffern");
 delay(1000);
}
```

Der Puffer muss so groß sein, dass er die maximale Anzahl von Zeichen im String aufnehmen kann. Bei 16-Bit-Integerwerten zur Basis 10 (dezimal) sind das sieben Zeichen (fünf Ziffern, ein mögliches Minuszeichen und die abschließende 0, die immer das Ende des Strings anzeigt); 32-Bit-Integerwerte benötigen 12 Zeichen (10 Ziffern, Minuszeichen und die abschließende 0). Sie erhalten keine Warnung, wenn die Puffergröße überschritten wird, doch dieser Fehler kann zu seltsamen Symptomen führen, weil der Überlauf einen anderen Teil des Speichers überschreibt, der von Ihrem Programm genutzt werden kann. Die einfachste Möglichkeit, das zu vermeiden, besteht darin, immer einen 12-Zeichen-Puffer zu verwenden und immer mit `ltoa` zu arbeiten, da es mit 16-Bit- und 32-Bit-Werten umgehen kann.

2.9 Einen String in eine Zahl umwandeln

Problem

Sie wollen einen String in eine Zahl umwandeln. Zum Beispiel könnten Sie einen Wert als String über einen Kommunikationslink empfangen haben und müssen ihn nun in einen Integer- oder einen Fließkommawert umwandeln.

Lösung

Das kann auf unterschiedliche Weise gelöst werden. Kommt der String in Form serieller Daten an, kann er während des Empfangs umgewandelt werden. Wie man das bei der seriellen Schnittstelle macht, zeigt Rezept 4.3.

Eine weitere Möglichkeit zur Umwandlung von Textstrings in Zahlen bieten die C-Konvertierungsfunktionen `atoi` (für `int`-Variablen) und `atol` (für `long`-Variablen).

Das folgende Code-Fragment terminiert die eingehenden Ziffern, sobald ein Zeichen keine Ziffer ist (oder wenn der Puffer voll ist). Damit das funktionieren kann, müssen Sie aber die Newline-Option im seriellen Monitor aktivieren oder ein anderes terminierendes Zeichen eingeben:

```
/*
 * StringToNumber
 * Erzeugt eine Zahl aus einem String
 */

const int ledPin = 13; // Mit Pin 13 verbundene LED

int  blinkDelay;   // Blinkrate wird durch diese Variable bestimmt
char strValue[6];   // Muss groß genug sein, um alle Ziffern und die den
            // String abschließende 0 aufzunehmen
int index = 0;     // Index auf das die empfangenen Ziffern speichernde Array

void setup()
{
 Serial.begin(9600);
 pinMode(ledPin,OUTPUT); // LED-Pin als Ausgang aktivieren
}

void loop()
{
 if( Serial.available())
 {
  char ch = Serial.read();
  if(index < 5 && isDigit(ch) ){
   strValue[index++] = ch; // ASCII-Zeichen an String anhängen;
  }
  else
  {
   // Bei vollem Puffer oder erster Nicht-Ziffer
   strValue[index] = 0;      // String mit einer 0 abschließen
   blinkDelay = atoi(strValue);  // String mit atoi in einen int-Wert umwandeln
```

```
    index = 0;
   }
  }
  blink();
}

void blink()
{
  digitalWrite(ledPin, HIGH);
  delay(blinkDelay/2); // Hälfte der Blinkperiode warten
  digitalWrite(ledPin, LOW);
  delay(blinkDelay/2); // Die andere Hälfte warten
}
```

Diskussion

Die etwas seltsam benannten Funktionen `atoi` (für ASCII nach `int`) und `atol` (für ASCII nach `long`) wandeln einen String in Integer- oder Long-Integer-Werte um. Um Sie verwenden zu können, müssen Sie den gesamten String zuerst in einem Zeichen-Array ablegen, bevor Sie die Konvertierungsfunktion aufrufen können. Der Code erzeugt ein Zeichen-Array namens `strValue`, das bis zu fünf Ziffern aufnehmen kann (die Deklaration mit `char strValue[6]` berücksichtigt noch die abschließende Null). Er füllt das Array über `Serial.read`, bis das erste Zeichen empfangen wird, das keine Ziffer ist. Das Array wird mit einer Null abgeschlossen und dann wird die `atoi`-Funktion aufgerufen, um das Zeichen-Array umzuwandeln. Das Ergebnis der Umwandlung wird in `blinkRate` gespeichert.

Eine Funktion namens `blink` wird aufgerufen, die den in `blinkDelay` gespeicherten Wert nutzt.

Wie in der Warnung in Rezept 2.4 erwähnt, müssen Sie darauf achten, innerhalb der Grenzen des Arrays zu bleiben. Falls Sie nicht wissen, wie Sie das anstellen sollen, sehen Sie sich die Diskussion dieses Rezepts an.

Die Arduino-Release 22 hat die Methode `toInt` eingeführt, die einen String in einen Integerwert umwandelt:

```
String aNumber = "1234";
int value = aNumber.toInt();
```

Arduino 1.0 verfügt über die Methode `parseInt`, mit deren Hilfe Sie Integerwerte über serielle Ports oder Ethernet (oder jedem aus der `Stream`-Klasse abgeleiteten Objekt) einlesen können. Das folgende Code-Fragment wandelt Ziffernfolgen in Zahlen um. Es ähnelt unserer Lösung, benötigt aber keinen Puffer (und ist nicht auf Zahlen mit fünf Ziffern beschränkt):

```
int  blinkDelay;   // Blinkrate wird durch diese Variable bestimmt
void loop()
{
 if( Serial.available())
 {
  blinkRate = Serial.parseInt();
 }
 blink();
 }
```

Stream-Parsing-Methoden wie `parseInt` nutzen ein Timeout, um die Kontrolle wieder an den Sketch zurückzugeben, falls innerhalb der gewünschten Zeitspanne keine Daten eintreffen. Das Standard-Timeout beträgt eine Sekunde, kann aber über die Methode `setTimeout` geändert werden:

```
Serial.setTimeout(1000 * 60); // Warte bis zu einer Minute
```

`parseInt` (und alle anderen Stream-Methoden) geben den Wert zurück, der bis zum Timeout eingelesen werden konnte (wenn kein Trennzeichen empfangen wurde). Der Rückgabewert besteht aus dem eingesammelten Wert. Wurden keine Ziffern empfangen, wird Null zurückgegeben. Arduino 1.0 hat keine Möglichkeit zu erkennen, ob es in der Parse-Methode zu einem Timeout kam, aber die Möglichkeit ist für eine zukünftige Release geplant.

Siehe auch

Dokumentation zu `atoi` finden Sie unter: *http://www.nongnu.org/avr-libc/user-manual/group__avr__stdlib.html*.

Viele Online-C/C++-Referenzseiten behandeln diese Low-Level-Funktionen, z.B. *http://www.cplusplus.com/reference/clibrary/cstdlib/atoi/* und *http://www.cppreference.com/wiki/string/c/atoi*.

In Rezept 4.3 und Rezept 4.5 erfahren Sie mehr über den Einsatz von `parseInt` mit Serial.

2.10 Ihren Code in Funktionsblöcken strukturieren

Problem

Sie möchten wissen, wie man einen Sketch um Funktionen erweitert und welche Funktionalität in eine Funktion gehört. Sie wollen außerdem verstehen, wie man die Gesamtstruktur des Sketches plant.

Lösung

Funktionen werden genutzt, um die von Ihrem Sketch durchgeführten Aktionen in funktionale Blöcke zu packen. Funktionen fassen Funktionalität zu wohldefinierten *Eingaben* (an eine Funktion übergebene Informationen) und *Ausgaben* (von der Funktion gelieferte Informationen) zusammen. Das erleichtert die Strukturierung, Pflege und Wiederverwendung Ihres Codes. Sie kennen bereits zwei Funktionen, die Teil jedes Arduino-Sketches sind: `setup` und `loop`. Sie legen eine Funktion an, indem Sie ihren *Rückgabetyp* (also die von ihr bereitgestellte Information) deklarieren, sowie ihren Namen und optionale Parameter (Werte), die die Funktion beim Aufruf verarbeitet.

Die Begriffe *Funktion* und *Methode* werden für wohldefinierte Code-Blöcke verwendet, die von anderen Teilen eines Programms als einzelne Entität aufgerufen werden können. Die Sprache C bezeichnet sie als Funktionen. Objektorientierte Sprachen wie C++, die Funktionalitäten über Klassen bereitstellen, neigen eher zum Begriff Methode. Arduino verwendet einen Stilmix (die Beispiel-Sketches verwenden eher einen C-Stil, während Bibliotheken eher so geschrieben werden, dass Sie C++-Methoden zur Verfügung stellen). In diesem Buch verwenden wir üblicherweise den Begriff Funktion, solange der Code nicht durch eine Klasse bereitgestellt wird. Doch keine Sorge, wenn Ihnen die Unterscheidung nicht klar ist, können Sie beide Begriffe gleichsetzen.

Hier eine einfache Funktion, die nur eine LED blinken lässt. Es gibt keine Parameter und sie gibt auch nichts zurück (was durch das vor der Funktion stehende `void` festgelegt wird):

```
// LED einmal blinken lassen
void blink1()
{
  digitalWrite(13,HIGH);  // LED einschalten
  delay(500);        // 500 Millisekunden warten
  digitalWrite(13,LOW);   // LED ausschalten
  delay(500);        // 500 Millisekunden warten
}
```

Die folgende Version verwendet einen Parameter (ein Integer namens `count`), der bestimmt, wie oft die LED blinken soll:

```
// LED count mal blinken lassen
void blink2(int count)
{
 while(count > 0 )  // Solange Zähler größer 0
 {
  digitalWrite(13,HIGH);
  delay(500);
  digitalWrite(13,LOW);
  delay(500);
  count = count -1; // Zähler dekrementieren
 }
}
```

Erfahrene Programmierer werden bemerken, dass beide Funktionen blink heißen können, da der Compiler sie anhand der als Parameter verwendeten Werte unterscheiden kann. Dieses Verhalten wird als *Überladen von Funktionen* bezeichnet. Das Arduino-`print`, das in Rezept 4.2 diskutiert wird, ist hierfür ein typisches Beispiel. Ein anderes Beispiel für das Überladen finden Sie in der Diskussion von Rezept 4.6.

Diese Version überprüft, ob der Wert von `count` `0` ist. Ist das nicht der Fall, lässt sie die LED einmal blinken und reduziert den Wert von `count` um 1. Das wird solange wiederholt, bis `count` nicht mehr größer als `0` ist.

Ein *Parameter* wird manchmal auch als *Argument* bezeichnet. Aus praktischen Erwägungen betrachten wir beide Begriffe als gleichwertig.

Hier ein Beispiel-Sketch mit einer Funktion, die einen Parameter erwartet und einen Wert zurückgibt. Der Parameter legt fest, wie lange die LED an- und ausgeschaltet bleibt (in Millisekunden). Die Funktion lässt die LED blinken, bis eine Taste gedrückt wird und gibt zurück, wie oft die LED geblinkt hat:

```
/*
 blink3 Sketch
 Demonstriert den Aufruf einer Funktion mit einem Parameter und die Rückgabe eines Wertes.
 Verwendet die gleiche Verschaltung wie im Sketch aus
 Rezept 5.2

 Die LED blinkt, sobald das Programm startet und hört auf zu blinken, sobald der mit dem digitalen
Pin 2 verbundene Taster gedrückt wird.
 Das Programm gibt dann aus, wie oft die LED geblinkt hat.
 */

const int ledPin = 13;      // Ausgangspin für LED
const int inputPin = 2;     // Eingangspin für Taster

void setup() {
 pinMode(ledPin, OUTPUT);
 pinMode(inputPin, INPUT);
 digitalWrite(inputPin,HIGH); // Internen Pullup nutzen (Rezept 5.2)
 Serial.begin(9600);
}

void loop(){
 Serial.println("Taster druecken und halten, damit die LED nicht mehr blinkt");
 int count = blink3(250); // LED 250ms ein und 250ms ausschalten
 Serial.print("Die LED hat ");
 Serial.println(count);
 Serial.println(" mal geblinkt");
}

// LED mit der übergebenen Zeitspanne blinken lassen
// Gibt zurück, wie oft die LED geblinkt hat
int blink3(int period)
{
 int result = 0;
 int switchVal = HIGH; //Mit Pullups high, wenn der Schalter nicht gedrückt ist

 while(switchVal == HIGH)  // Wiederholen, bis Taster gedrückt wurde
              // (Wert wird dann LOW)
 {
  digitalWrite(13,HIGH);
  delay(period);
  digitalWrite(13,LOW);
  delay(period);
  result = result + 1; // Zähler erhöhen
  switchVal = digitalRead(inputPin);  // Tasterwert einlesen
 }
```

```
 // switchVal ist nicht länger HIGH, weil der Taster gedrückt wurde
 return result; // Dieser Wert wird zurückgegeben
}
```

Diskussion

Der Code in der Lösung dieses Rezepts illustriert die drei Arten von Funktionsaufrufen, denen Sie begegnen werden. `blink1` hat keine Parameter und keinen Rückgabewert. Die Form ist:

```
void blink1()
{
  // Implementierung steht hier...
}
```

`blink2` erwartet einen einzelnen Parameter, gibt aber keinen Wert zurück:

```
void blink2(int count)
{
  // Implementierung steht hier...
}
```

`blink3` erwartet einen einzelnen Parameter und liefert einen Wert zurück:

```
int blink3(int period)
{
  // Implementierung steht hier...
}
```

Der vor dem Funktionsnamen stehende Datentyp legt den Typ des Rückgabewerts fest (bzw. keinen Rückgabewert bei `void`). Wenn Sie *die Funktion deklarieren* (den Code schreiben, der die Funktion und ihre Aktionen definiert), hängen Sie kein Semikolon an die schließende geschweifte Klammer an. Wenn Sie die Funktion *nutzen* (aufrufen), müssen Sie hinter dem Funktionsaufruf ein Semikolon anhängen.

Die meisten Funktionen, die Ihnen unterkommen werden, sind Variationen dieser Formen. Hier zum Beispiel eine Funktion, die einen Parameter erwartet und einen Wert zurückgibt:

```
int sensorPercent(int pin)
{
int percent;

 int val = analogRead(pin); // Sensorwert einlesen (im Bereich von 0 bis 1023)
 percent = map(val,0,1023,0,100); // percent liegt im Bereich von 0 bis 100.
 return percent;
}
```

Der Name der Funktion ist `sensorPercent`. Sie übergeben ihr die Nummer eines Analogpins und erhalten einen Wert in Prozent zurück (in Rezept 5.7 erfahren Sie mehr über `analogRead` und `map`). Das `int` vor der Deklaration teilt dem Compiler (und dem Programmierer) mit, dass die Funktion einen Integerwert zurückgibt. Bei der Entwicklung von Funktionen müssen Sie den für die Funktion geeigneten Rückgabetyp wählen. Diese Funktion gibt einen Integerwert zwischen 0 und 100 zurück, weshalb der Rückgabetyp `int` geeignet ist.

Es ist empfehlenswert, den Funktionen sinnvolle Namen zu geben. Es ist gängige Praxis, dafür Worte miteinander zu verbinden und dabei den ersten Buchstaben jedes Wortes großzuschreiben (mit Ausnahme des ersten Wortes). Sie können den von Ihnen bevorzugte Stil nutzen, es hilft aber anderen, wenn Sie eine konsistente Namensgebung nutzen.

`sensorPercent` nutzt einen Parameter namens `pin` (beim Aufruf der Funktion wird `pin` der Wert zugewiesen, der an die Funktion übergeben wurde).

Der Funktionsrumpf (der Code zwischen den geschweiften Klammern) führt die gewünschten Aktionen durch – im obigen Beispiel wird ein Wert von einem analogen Eingangspin eingelesen und auf einem Prozentwert abgebildet. Dieser Prozentwert wird temporär in einer Variablen namens `percent` festgehalten. Die nachfolgende Anweisung sorgt dafür, dass der in der Variablen `percent` gespeicherte Wert an die aufrufende Anwendung zurückgegeben wird:

```
return percent;
```

Sie können den gleichen Effekt aber auch ohne die `percent`-Variable erreichen:

```
int sensorPercent(int pin)
{
 int val = analogRead(pin);  // Sensorwert einlesen (im Bereich von 0 bis 1023)
 return map(val,0,1023,0,100); // Prozentwert liegt zwischen 0 und 100.
}
```

Aufgerufen wird die Funktion wie folgt:

```
// Prozentwerte von 6 Analogpins ausgeben
for(int sensorPin = 0; sensorPin < 6; sensorPin++)
{
 Serial.print("Sensor an Pin ");
 Serial.print(sensorPin);
 Serial.print(" steht bei ");
 int val = sensorPercent(sensorPin);
 Serial.print(val);
 Serial.print(" Prozent.");
}
```

Siehe auch

Die Arduino-Funktionsreferenz unter: *http://www.arduino.cc/en/Reference/FunctionDeclaration*

2.11 Mehr als einen Wert in einer Funktion zurückliefern

Problem

Sie möchten zwei oder mehr Werte in einer Funktion zurückgeben. Rezept 2.10 enthält Beispiele für die gängigste Form einer Funktion, die nur einen (oder keinen) Wert zurückgibt. Doch manchmal muss man mehr als einen Wert modifizieren oder zurückgeben.

Lösung

Für dieses Problem gibt es verschiedene Lösungen. Die einfachste besteht darin, die Funktion einige globale Variablen verändern zu lassen und gar nichts zurückzugeben:

```
/*
 swap Sketch
 Änderung zweier Werte über globale Variablen
 */

int x; // x und y sind globale Variablen
int y;

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

void loop(){
 x = random(10); // Zufällige Zahlen wählen
 y = random(10);

 Serial.print("Die Werte von x und y vor dem Tausch: ");
 Serial.print(x); Serial.print(","); Serial.println(y);
 swap();

 Serial.print("Die Werte von x und y nach dem Tausch: ");
 Serial.print(x); Serial.print(","); Serial.println(y);Serial.println();


 delay(1000);
}

// Zwei globale Variablen vertauschen
void swap()
{
 int temp;
 temp = x;
 x = y;
 y = temp;
}
```

Die `swap`-Funktion vertauscht die Werte in zwei globalen Variablen. Globale Variablen sind leicht zu verstehen (sie sind überall im Programm zugänglich und können von jedem geändert werden), werden von erfahrenen Programmierern aber gemieden, da es sehr leicht ist, sie versehentlich zu überschreiben. Darüber hinaus können Funktionen plötzlich nicht mehr laufen, weil Sie den Namen oder den Typ einer globalen Variablen an einer anderen Stelle im Sketch geändert haben.

Eine sichere und elegante Lösung besteht darin, Referenzen an die zu ändernden Werte zu übergeben. In der Funktion werden dann die Referenzen genutzt, um die Werte zu modifizieren. Das geht wie folgt:

```
/*
 functionReferences Sketch
 Rückgabe mehrerer Werte mittels Referenz
*/


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

void loop(){
 int x = random(10); // Zufällige Zahlen wählen
 int y = random(10);

 Serial.print("Die Werte von x und y vor dem Tausch: ");
 Serial.print(x); Serial.print(","); Serial.println(y);
 swap(x,y);

 Serial.print("Die Werte von x und y nach dem Tausch: ");
 Serial.print(x); Serial.print(","); Serial.println(y);Serial.println();

 delay(1000);
}

// Zwei Werte vertauschen
void swap(int &value1, int &value2)
{
 int temp;
 temp = value1;
 value1 = value2;
 value2 = temp;
}
```

Diskussion

Die swap-Funktion ähnelt den in Rezept 2.10 beschriebenen Funktionen mit Parametern, doch das kaufmännische UND (Ampersand, &) zeigt an, dass die Parameter *Referenzen* sind. Das bedeutet, dass Werteänderungen innerhalb der Funktion auch die Werte der Variablen ändern, die beim Aufruf der Funktion übergeben wurden. Sie können verfolgen, wie das funktioniert, indem Sie zuerst den Lösungscode ausführen und sich vergewissern, dass die Werte vertauscht wurden. Dann entfernen Sie die beiden &-Zeichen aus der Funktionsdefinition:

```
void swap(int value1, int value2)
```

Wenn Sie diesen Code ausführen, sehen Sie, dass die Werte nicht vertauscht wurden – Änderungen innerhalb der Funktion erfolgen nur lokal zur Funktion und gehen verloren, wenn die Funktion beendet wird.

Wenn Sie die Arduino-Release 21 (oder älter) nutzen, müssen Sie die Funktion zuerst deklarieren, um den Compiler darüber zu informieren, dass die Funktion Referenzen nutzt. Der Sketch zu diesem Rezept im

Downloadbereich zur ersten Auflage des Buches zeigt, wie man eine Funktion deklariert:

```
// Funktionen mit Referenzen müssen vor der Verwendung deklariert werden
// Die Deklaration steht am Anfang vor dem setup- und loop-Code
// Beachten Sie das Semikolon am Ende der Deklaration
 void swap(int &value1, int &value2);
```

Eine Funktionsdeklaration ist ein *Prototyp* – sie legt den Namen, die Typen der an die Funktion übergebenen Werte und den Rückgabetyp fest. Der Arduino-Build-Prozess erzeugt die Deklarationen üblicherweise hinter den Kulissen für Sie. Doch wenn Sie eine vom Standard abweichende Syntax verwenden (zumindest was Arduino 21 und älter betrifft), dann erzeugt der Build-Prozess die Deklaration nicht. Sie müssen die Zeile dann vor `setup` selbst einfügen.

Eine Funktionsdefinition besteht aus einem Funktionskopf und einem Funktionsrumpf. Der Funktionskopf ähnelt der Deklaration, endet aber nicht mit einem Semikolon. Der Funktionsrumpf ist der Code innerhalb der geschweiften Klammern, der ausgeführt wird, wenn Sie die Funktion aufrufen.

2.12 Aktionen basierend auf Bedingungen ausführen

Problem

Sie möchten einen Code-Block nur dann ausführen, wenn eine bestimmte Bedingung erfüllt ist. Zum Beispiel könnten Sie eine LED nur dann einschalten wollen, wenn ein Taster gedrückt wird oder wenn ein Analogwert einen bestimmten Schwellwert überschritten hat.

Lösung

Der folgende Code verwendet die Verschaltung aus Rezept 5.1:

```
/*
  Pushbutton Sketch
  Ein mit dem digitalen Pin 2 verbundender Taster schaltet die LED an Pin 13
*/

const int ledPin = 13;       // Pin für die LED
const int inputPin = 2;       // Pin für den Taster

void setup() {
 pinMode(ledPin, OUTPUT);     // LED-Pin als Ausgang deklarieren
 pinMode(inputPin, INPUT);     // Taster-Pin als Eingang delarieren
}

void loop(){
 int val = digitalRead(inputPin);  // Eingangswert einlesen
 if (val == HIGH)           // Prüfen, ob Eingang HIGH ist
 {
```

```
  digitalWrite(ledPin, HIGH);   // LED einschalten, wenn Taster gedrückt ist
 }
}
```

Diskussion

Die `if`-Anweisung wird verwendet, um den Wert von `digitalRead` zu testen. Eine `if`-Anweisung muss innerhalb der Klammern einen Test durchführen, dessen Ergebnis nur wahr oder falsch sein kann. Im obigen Beispiel ist das `val == HIGH`, und der Code-Block, der auf die `if`-Anweisung folgt, wird nur ausgeführt, wenn der Ausdruck wahr ist. Ein Code-Block besteht aus dem gesamten Code innerhalb der geschweiften Klammern. (Wenn Sie keine Klammern verwenden, ist der Block einfach die nächste ausführbare Anweisung, die mit einem Semikolon abgeschlossen ist).

Soll eine Aktion ausgeführt werden, wenn die Bedingung erfüllt ist, und eine andere, wenn nicht, können Sie die `if...else`-Anweisung verwenden:

```
/*
  Pushbutton Sketch
  Ein mit dem digitalen Pin 2 verbundender Taster schaltet die LED an Pin 13
*/

const int ledPin = 13;       // Pin für die LED
const int inputPin = 2;       // Pin für den Taster

void setup() {
 pinMode(ledPin, OUTPUT);      // LED-Pin als Ausgang deklarieren
 pinMode(inputPin, INPUT);     // Taster als Eingang deklarieren
}

void loop(){
 int val = digitalRead(inputPin);  // Eingangswert einlesen
 if (val == HIGH)           // Prüfen, ob Eingang HIGH ist
 {
  // Dieser Teil wird ausgeführt, wenn val HIGH ist
  digitalWrite(ledPin, HIGH);   // LED einschalten, wenn Taster gedrückt ist
 }
 else
 {
  // Dieser Teil wird ausgeführt, wenn val nicht HIGH ist
  digitalWrite(ledPin, LOW);    // LED ausschalten
 }
}
```

Siehe auch

Die Diskussion zu Booleschen Typen in Rezept 2.2.

2.13 Eine Folge von Anweisungen wiederholt ausführen

Problem

Sie wollen einen Block mit Anweisungen ausführen, solange ein Ausdruck wahr ist.

Lösung

Eine while-Schleife führt eine oder mehrere Anweisungen aus, solange ein Ausdruck wahr ist:

```
/*
 * Repeat
 * Blinken, solange eine Bedingung wahr ist
 */

const int ledPin   = 13; // Digitaler Pin, mit dem die LED verbunden ist
const int sensorPin = 0; // Analoger Eingang 0

void setup()
{
 Serial.begin(9600);
 pinMode(ledPin,OUTPUT); // LED-Pin als Ausgang aktivieren
}

void loop()
{
 while(analogRead(sensorPin) > 100)
 {
  blink();   // Funktion aufrufen, die die LED ein- und ausschaltet
  Serial.print(".");
 }
 Serial.println(analogRead(sensorPin)); // Wird erst ausgeführt,
                      // wenn die while-Schleife beendet wurde!!!
}

void blink()
{
  digitalWrite(ledPin, HIGH);
  delay(100);
  digitalWrite(ledPin, LOW);
  delay(100);
}
```

Dieser Code führt die Anweisungen im Block zwischen den geschweiften Klammern {} aus, solange der Wert von analogRead größer als 100 ist. Damit könnten Sie eine LED als Alarm blinken lassen, solange ein bestimmter Schwellwert überschritten ist. Die LED ist aus, wenn der Sensorwert bei 100 (oder kleiner) liegt. Sie blinkt kontinuierlich, wenn der Wert größer als 100 ist.

Diskussion

Geschweifte Klammern legen den Code-Block fest, der innerhalb einer Schleife ausgeführt wird. Ohne geschweifte Klammern wird nur eine Codezeile in der Schleife wiederholt:

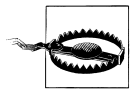
```
while(analogRead(sensorPin) > 100)
  blink();  // Die dem Schleifenausdruck unmittelbar folgende Zeile wird wiederholt ausgeführt
  Serial.print("."); // Wird erst ausgeführt, wenn die Schleife beendet wurde!!!
```

Schleifen ohne geschweifte Klammern können unerwartete Ergebnisse liefern, wenn es mehr als eine Codezeile gibt.

Das `do...while` ähnelt der `while`-Schleife, doch die Anweisungen innerhalb des Code-Blocks werden ausgeführt, bevor die Bedingung überprüft wird. Verwenden Sie diese Form, wenn der Code mindestens einmal ausgeführt werden muss, auch wenn der Ausdruck falsch ist:

```
do
{
  blink(); // Funktion zum Ein- und Ausschalten der LED aufrufen
}
while (analogRead(sensorPin) > 100);
```

Der obige Code lässt die LED mindestens einmal blinken, und dann so lange, wie der Sensorwert über 100 liegt. Ist der Wert kleiner als 100, blinkt die LED nur einmal auf. Dieser Code könnte bei batteriebetriebenen Schaltungen genutzt werden. Ein einzelnes Blinken zeigt, dass die Schaltung aktiv ist, während ein kontinuierliches Blinken anzeigt, dass die Batterie geladen wird.



Nur der Code innerhalb der `while`- oder `do-Schleife` wird ausgeführt, bis die Schleife beendet wird. Muss ein Sketch als Reaktion auf eine andere Bedingung (z.B. Timeout, Zustand eines Sensors oder eine andere Eingabe) die Schleife beenden, können Sie `break` nutzen:

```
while(analogRead(sensorPin) > 100)
{
  blink();
  if(Serial.available())
    break; // Jede serielle Eingabe beendet die while-Schleife
}
```

Siehe auch

Kapitel 4 and Kapitel 5

2.14 Anweisungen über einen Zähler wiederholen

Problem

Sie wollen eine oder mehrere Anweisungen n-mal wiederholen. Die `for`-Schleife ähnelt der `while`-Schleife, bietet aber eine genauere Kontrolle der Start- und Endbedingungen.

Lösung

Dieser Sketch zählt von 0 bis 3 und gibt den Wert der Variablen `i` in einer `for`-Schleife aus:

```
/*
  ForLoop Sketch
  Demonstration der for-Schleife
*/

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

void loop(){
 Serial.println("for(int i=0; i < 4; i++)");
 for(int i=0; i < 4; i++)
 {
  Serial.println(i);
 }
}
```

Die Ausgabe am seriellen Monitor sieht wie folgt aus und wird fortlaufend wiederholt.

```
for(int i=0; i < 4; i++)
0
1
2
3
```

Diskussion

Eine `for`-Schleife besteht aus drei Teilen: Initialisierung, Bedingung und Iteration (eine Anweisung, die am Ende jedes Schleifendurchlaufs ausgeführt wird). Die Teile werden durch ein Semikolon getrennt. Im obigen Code initialisiert `int i=0;` die Variable `i` mit `0`; `i < 4;` prüft, ob die Variable kleiner 4 ist und `i++` inkrementiert `i`.

Eine `for`-Schleife kann eine existierende Variable verwenden oder eine Variable erzeugen, die nur innerhalb der Schleife genutzt wird. Die nachfolgende Version verwendet den Wert der Variablen `j`, die an anderer Stelle des Sketches angelegt wurde:

```
 int j;

 Serial.println("for(j=0; j < 4; j++ )");
 for(j=0; j < 4; j++ )
 {
  Serial.println(j);
 }
```

Das ist nahezu identisch mit dem ersten Beispiel, lässt aber das Schlüsselwort `int` bei der Initialisierung weg, da die Variable `j` bereits definiert ist. Die Ausgabe dieser Version ist mit der aus der erste Version identisch:

```
for(j=0; i < 4; i++)
0
1
2
3
```

Sie können die Initialisierung auch ganz weglassen, wenn die Schleife den Wert einer bereits früher definierten Variablen verwenden soll. Der folgende Code beginnt die Schleife mit `j = 1`:

```
int j = 1;

Serial.println("for(  ; j < 4; j++ )");
for( ; j < 4; j++ )
{
 Serial.println(j);
}
```

und liefert die folgende Ausgabe:

```
for( ; j < 4; j++)
1
2
3
```

Sie kontrollieren im Bedingungsteil, wann die Schleife beendet wird. Im obigen Beispiel wird überprüft, ob die Schleifenvariable kleiner als 4 ist und die Schleife endet, sobald diese Bedingung nicht länger erfüllt ist.

Wenn Ihre Schleifenvariable bei 0 beginnt und viermal durchlaufen werden soll, muss die Bedingung auf einen Wert kleiner 4 testen. Die Schleife wird durchlaufen, solange die Bedingung erfüllt ist, und wenn die Schleife bei 0 beginnt, gibt es vier Werte, die kleiner als 4 sind.

Der folgende Code prüft, ob der Wert der Schleifenvariablen kleiner oder gleich 4 ist. Die Schleife gibt also die Ziffern 0 bis 4 aus:

```
Serial.println("for(int i=0; i <= 4; i++)");
for(int i=0; i <= 4; i++)
{
 Serial.println(i);
}
```

Der dritte Teil einer `for`-Schleife ist die Iterator-Anweisung, die am Ende jedes Schleifendurchgangs ausgeführt wird. Sie kann jede gültige C/C++-Anweisung enthalten. Das folgende Beispiel erhöht den Wert von `i` bei jedem Durchgang um 2:

```
Serial.println("for(int i=0; i < 4; i+= 2)");
for(int i=0; i < 4; i+=2)
{
 Serial.println(i);
}
```

Der Sketch gibt nur die Werte 0 und 2 aus.

Der *Iterator*-Ausdruck kann auch herunterzählen, im folgenden Beispiel von 3 bis 0:

```
Serial.println("for(int i=3; i > = 0 ; i--)");
for(int i=3; i >= 0 ; i--)
{
 Serial.println(i);
}
```

Wie alle anderen Teil einer `for`-Schleife kann auch der Iterator-Ausdruck leer bleiben. (Die Teile müssen aber immer durch Semikolon getrennt werden, auch wenn sie leer sind.)

Die folgende Version inkrementiert `i` nur dann, wenn ein Eingangspin aktiv ist. Die `for`-Schleife ändert den Wert von `i` nicht. Er wird nur in der `if`-Anweisung hinter `Serial.print` geändert – Sie müssen `inPin` definieren und mit `pinMode()`als INPUT festlegen:

```
Serial.println("for(int i=0; i < 4; )");
for(int i=0; i < 4; )
{
 Serial.println(i);
 if(digitalRead(inPin) == HIGH) {
  i++; // Wert wird nur inkrementiert, wenn Eingang HIGH ist
 }
}
```

Siehe auch

Arduino-Referenz zur `for`- Anweisung: *http://www.arduino.cc/en/Reference/For*

2.15 Aus Schleifen ausbrechen

Problem

Sie wollen eine Schleife basierend auf einer anderen Bedingung vorzeitig beenden.

Lösung

Nutzen Sie den folgenden Code:

```
while(analogRead(sensorPin) > 100)
{
  if(digitalRead(switchPin) == HIGH)
  {
    break;  //Schleife beenden, wenn Taster gedrückt
  }
  flashLED();       // LED ein- und ausschalten
}
```

Diskussion

Der Code ähnelt den anderen `while`-Beispielen, nutzt aber die `break`-Anweisung, um die Schleife zu verlassen, wenn ein digitaler Pin HIGH ist. Ist beispielsweise wie in Rezept 5.1 ein Taster mit dem Pin verbunden, wird die Schleife beendet (und die LED hört auf zu blinken), selbst wenn die Bedingung der `while`-Schleife wahr ist.

Siehe auch

Arduino-Referenz zur `break`-Anweisung: *http://www.arduino.cc/en/Reference/Break*

2.16 Basierend auf einem Variablenwert verschiedene Aktionen durchführen

Problem

Sie müssen in Abhängigkeit von einem Wert unterschiedliche Aktionen ausführen. Sie könnten dazu mehrere `if`- und `else if`-Anweisungen verwenden, aber dadurch wird der Code schnell kompliziert, unverständlich und schwer zu modifizieren. Darüber hinaus könnten Sie auf einen Wertebereich hin prüfen wollen.

Lösung

Die `switch`-Anweisung ermöglicht die Wahl zwischen einer Reihe von Alternativen. Die Funktionalität ähnelt der mehrerer `if`/`else if`-Anweisungen, ist aber kompakter:

```
/*
 * SwitchCase Sketch
 * Beispiel für switch-Anweisung über Zeichen am seriellen Port
 *
 * Das Zeichen 1 lässt die LED einmal blinken, 2 lässt sie zweimal blinken.
 * + schaltet die LED ein, - schaltet sie aus
 * jedes andere Zeichen gibt eine Nachricht über den seriellen Monitor aus
 */
const int ledPin = 13; // Mit Pin 13 verbundene LED

void setup()
{
 Serial.begin(9600); // Initialisiert seriellen Port zum Senden
            // und Empfangen mit 9600 Baud
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 if ( Serial.available()) // Prüfe, ob ein Zeichen
              // vorhanden ist
 {
  char ch = Serial.read();
```

```
  switch(ch)
  {
  case '1':
   blink();
   break;
  case '2':
   blink();
   blink();
   break;
  case '+':
   digitalWrite(ledPin,HIGH);
   break;
  case '-':
   digitalWrite(ledPin,LOW);
   break;
  default :
   Serial.print(ch);
   Serial.println(" wurde empfangen, hat aber keine Funktion");
   break;
  }
 }
}

void blink()
{
 digitalWrite(ledPin,HIGH);
 delay(500);
 digitalWrite(ledPin,LOW);
 delay(500);
}
```

Diskussion

Die `switch`-Anweisung evaluiert den Wert der Variablen `ch`, der über die serielle Schnittstelle empfangen wurde, und verzweigt bei dem zum Wert passenden Label. Die Label müssen numerische Konstanten sein (Sie dürfen keine Strings in einer `case`-Anweisung verwenden), und ein Wert darf nur einmal vorkommen. Wenn auf die Anweisungen keine `break`-Anweisung folgt, *rutscht* die Ausführung zum nächsten Fall durch:

```
  case '1':
   blink();  // Keine break-Anweisung vor dem nächsten Label
  case '2':
   blink();  // Fall '1' wird hier fortgesetzt
   blink();
   break;   // break-Anweisung beendet den switch-Ausdruck
```

Lässt man die `break`-Anweisung am Ende von `case '1':` weg (wie im obigen Code zu sehen), dann wird die `blink`-Funktion dreimal aufgerufen, wenn `ch` eine `1` enthält. Das `break` zu vergessen, ist ein typischer Fehler. Das `break` bewusst wegzulassen, ist manchmal aber auch ganz praktisch. Da das andere Leser des Codes verwirren kann, verdeutlicht man seine Absichten mit einem Kommentar.

Wenn sich eine switch-Anweisung nicht verhält wie gewünscht, sollten Sie sicherstellen, dass Sie keine break-Anweisungen vergessen haben.

Das Label default: wird genutzt, um Werte abzufangen, die von den case-Labeln nicht abgedeckt werden. Gibt es kein default-Label, macht der switch-Ausdruck nichts weiter, wenn es keinen Treffer gibt.

Siehe auch

Arduino-Referenz zu den switch- und case-Anweisungen: *http://www.arduino.cc/en/Reference/SwitchCase*

2.17 Zeichen und Zahlen vergleichen

Problem

Sie wollen die Beziehung zwischen Werten ermitteln.

Lösung

Vergleichen Sie Integerwerte mit den relationalen Operatoren in Tabelle 2-3.

Tabelle 2-3: Relationale Operatoren und Gleichheitsoperatoren

Operator	Test auf	Beispiel
==	Gleich	2 == 3 // Evaluiert zu falsch
!=	Ungleich	2 != 3 // Evaluiert zu wahr
>	Größer als	2 > 3 // Evaluiert zu falsch
<	Kleiner als	2 < 3 // Evaluiert zu wahr
>=	Größer oder gleich	2 >= 3 // Evaluiert zu falsch
<=	Kleiner oder gleich	2 <= 3 // Evaluiert zu wahr

Der folgende Sketch demonstriert die Ergebnisse der Vergleichsoperatoren:

```
/*
 * RelationalExpressions Sketch
 * demonstriert Wertevergleiche
 */

int i = 1; // Unsere Ausgangswerte
int j = 2;

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

void loop(){
```

```
  Serial.print("i = ");
  Serial.print(i);
  Serial.print(" und j = ");
  Serial.println(j);

  if(i < j)
   Serial.println(" i ist kleiner als j");
  if(i <= j)
   Serial.println(" i ist kleiner oder gleich j");
  if(i != j)
   Serial.println(" i ist ungleich j");
  if(i == j)
   Serial.println(" i ist gleich j");
  if(i >= j)
   Serial.println(" i ist groesser der gleich j");
  if(i > j)
   Serial.println(" i ist groesser als j");

  Serial.println();
  i = i + 1;
  if(i > j + 1)
   delay(10000); // Lange Verzögerung, wenn i nicht mehr nah an j liegt
}
```

Hier die Ausgabe:

```
i = 1 und j = 2
 i ist kleiner als j
 i ist kleiner oder gleich j
 i ist ungleich j

i = 2 und j = 2
 i ist kleiner oder gleich j
 i ist gleich j
 i ist groesser oder gleich j

i = 3 und j = 2
 i ist ungleich j
 i ist groesser oder gleich j
 i ist groesser als j
```

Diskussion

Beachten Sie, dass der Gleichheitsoperator aus zwei Gleichheitszeichen (==) besteht. Einer der häufigsten Programmierfehler besteht darin, ihn mit dem Zuweisungsoperator zu verwechseln, der aus nur einem Gleichheitszeichen besteht.

Der folgende Ausdruck vergleicht i mit 3. Der Programmierer wollte Folgendes ausdrücken:

```
if(i == 3) // Prüfe, ob i gleich 3 ist
```

hat im Sketch aber Folgendes geschrieben:

```
if(i = 3) // Versehentlich nur ein Gleichheitszeichen verwendet!!!!
```

Das gibt immer `true` zurück, weil `i` auf `3` gesetzt wird und der Vergleich somit immer zutrifft.

Eine Möglichkeit, diesen Fehler beim Vergleich mit Konstanten (festen Werten) zu vermeiden, besteht darin, die Konstante auf die linke Seite des Ausdrucks zu stellen:

```
if(3 = i) // Versehentlich nur ein Gleichheitszeichen verwendet!!!!
```

Der Compiler erkennt das als Fehler, weil er weiß, dass man einer Konstanten keinen Wert zuweisen kann.

Die Fehlermeldung klingt allerdings etwas unfreundlich: »value required as left operand of assignment“. Wenn Sie diese Meldung sehen, versuchen Sie, etwas einen Wert zuzuweisen, das nicht geändert werden kann.

Siehe auch

Arduino-Referenz zu Bedingungs- und Vergleichsoperatoren: *http://www.arduino.cc/en/Reference/If*

2.18 Strings vergleichen

Problem

Sie wollen wissen, ob zwei Strings gleich sind.

Lösung

Es gibt eine Funktion zum Stringvergleich namens `strcmp` (*string compare*). Hier ein Fragment, das seine Nutzung verdeutlicht:

```
char string1[ ] = "links";
char string2[ ] = "rechts";

if(strcmp(string1, string2) == 0)
  Serial.print("Strings sind gleich")
```

Diskussion

`strcmp` gibt `0` zurück, wenn die Strings gleich sind, und einen Wert größer 0, wenn das erste nicht übereinstimmende Zeichen im ersten String größer ist als im zweiten String. Ein Wert kleiner 0 wird zurückgegeben, wenn das erste nicht übereinstimmende Zeichen im ersten String kleiner ist als im zweite String. Üblicherweise möchte man nur wissen, ob die Strings gleich sind, und auch wenn der Test auf 0 nicht gerade intuitiv ist, gewöhnt man sich doch schnell daran.

Beachten Sie, dass unterschiedlich lange Strings nicht gleich sind, auch wenn der kürzere String im längeren enthalten ist:

```
strcmp("links", "linksmitte") == 0) // Evaluiert zu falsch
```

Sie können mit der strncmp-Funktion eine bestimmte Zahl von Zeichen vergleichen. Sie übergeben strncmp die maximale Anzahl zu vergleichender Zeichen, und die Funktion bricht den Vergleich nach dieser Anzahl von Zeichen ab:

```
strncmp("links", "linksmitte", 4) == 0) // Evaluiert zu wahr
```

Im Gegensatz zur Zeichenkette können Arduino-Strings direkt verglichen werden:

```
String stringOne = String("dies");
if (stringOne == "dies")
 Serial.println("dies ist wahr");
if (stringOne == "das")
 Serial.println("dies ist falsch");
```

Eine Einführung zum Arduino String-Vergleich finden Sie unter *http://arduino.cc/en/Tutorial/StringComparisonOperators*.

Siehe auch

Weitere Informationen zu strcmp finde Sie unter *http://www.cplusplus.com/reference/clibrary/cstring/strcmp/*.

In Rezept 2.5 finden Sie eine Einführung in Arduino-Strings.

2.19 Logische Vergleiche durchführen

Problem

Sie wollen logische Beziehungen zwischen zwei oder mehr Ausdrücken vergleichen. Zum Beispiel wollen Sie basierend auf den Bedingungen einer if-Anweisung unterschiedliche Aktionen durchführen.

Lösung

Nutzen Sie die logischen Operatoren aus Tabelle 2-4.

Tabelle 2-4: Logische Operatoren

Symbol	Funktion	Kommentar
&&	Logisches UND	Evaluiert zu true, wenn die Bedingungen auf beide Seiten des &&-Operators wahr sind
||	Logisches ODER	Evaluiert zu true, wenn die Bedingung auf zumindest einer Seite des ||-Operators wahr ist
!	NICHT	Evaluiert zu true, wenn der Ausdruck falsch ist und zu false, wenn der Ausdruck wahr ist

Diskussion

Logische Operatoren liefern Wahr/Falsch-Werte basierend auf logischen Beziehungen zurück. Die nachfolgenden Beispiele setzen Sensoren an den digitalen Pins 2 und 3 voraus, wie in Kapitel 5 beschrieben.

Der logische UND-Operator && gibt `true` zurück, wenn beide Operanden wahr sind, anderenfalls `false`:

```
if( digitalRead(2) && digitalRead(3) )
 blink(); // Blinken, wenn beide Pins HIGH sind
```

Der logische ODER-Operator || gibt `true` zurück, wenn einer der beiden Operanden wahr ist, und `false`, wenn beide Operanden falsch sind:

```
if( digitalRead(2) || digitalRead(3) )
 blink(); // Blinken, wenn einer der beiden Pins HIGH ist
```

Der NICHT-Operator ! besitzt nur einen Operanden, dessen Wert invertiert wird – er liefert also `false` zurück, wenn der Operand wahr ist, und `true`, wenn er falsch ist:

```
if( !digitalRead(2) )
 blink(); // Blinken, wenn der Pin nicht HIGH ist
```

2.20 Bitweise Operationen durchführen

Problem

Sie möchten bestimmte Bits in einem Wert setzen oder löschen.

Lösung

Verwenden Sie die Bit-Operatoren aus Tabelle 2-5.

Tabelle 2-5: Bit-Operatoren

Symbol	Funktion	Ergebnis	Beispiel
&	Bitweises UND	Setzt die Bits an den jeweiligen Stellen auf 1, wenn beide Bits 1 sind. Anderenfalls werden die Bits auf 0 gesetzt.	`3 & 1` ergibt `1`(`11 & 01` ergibt `01`)
|	Bitweises ODER	Setzt die Bits an den jeweiligen Stellen auf 1, wenn eines der Bits 1 ist.	`3 | 1` ergibt `3`(`11 | 01` ergibt `11`)
^	Bitweises EXKLUSIV-ODER	Setzt die Bits an den jeweiligen Stellen auf 1, wenn nur eines der beiden Bits 1 ist.	`3 ^ 1` ergibt `2`(`11 ^ 01` ergibt `10`)
~	Bitweise Negation	Invertiert den Wert jedes Bits. Das Ergebnis ist von der Anzahl der Bits und dem Datentyp abhängig.	`~1` ergibt `254`(`~00000001` ergibt `11111110`)

Der folgende Sketch geht die Beispiele aus Tabelle 2-5 noch mal durch:

```
/*
 * bits Sketch
 * Bitweise Operatoren
 */

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

void loop(){
 Serial.print("3 & 1 ergibt dezimal "); // Bitweises UND von 3 und 1
 Serial.print(3 & 1);      // Ergebnis ausgeben
 Serial.print(" und binaer ");
 Serial.println(3 & 1 , BIN);  // Ergebnis binär ausgeben

 Serial.print("3 | 1 ergibt dezimal "); // Bitweises ODER von 3 und 1
 Serial.print(3 | 1 );
 Serial.print(" und binaer ");
 Serial.println(3 | 1 , BIN);  // Ergebnis binär ausgeben

 Serial.print("3 ^ 1 ergibt dezimal "); // Bitweises EXKLUSIV-ODER von 3 und 1
 Serial.print(3 ^ 1); Serial.print(" und binaer ");
 Serial.println(3 ^ 1 , BIN);  // Ergebnis binär ausgeben


 byte byteVal = 1;
 int intVal = 1;

 byteVal = ~byteVal; // Bitweise Negation
 intVal = ~intVal;

 Serial.print("~byteVal (1) ergibt "); // 8-Bit-Wert bitweise negieren
 Serial.println(byteVal, BIN); // Ergebnis binär ausgeben
 Serial.print("~intVal (1) ergibt "); // 16-Bit-Wert negieren
 Serial.println(intVal, BIN);  // Ergebnis binär augeben

 delay(10000);
}
```

Hier die Ausgabe über den seriellen Monitor:

```
3 & 1 ergibt dezimal 1 und binaer 1
3 | 1 ergibt dezimal 3 und binaer 11
3 ^ 1 ergibt dezimal 2 und binaer 10
~byteVal (1) ergibt 11111110
~intVal (1) ergibt 11111111111111111111111111111110
```

Diskussion

Bitweise Operatoren werden genutzt, um Bits zu setzen oder zu testen. Wenn Sie zwei Werte über »UND« oder »ODER« verknüpfen, arbeiten die Operatoren mit den einzelnen Bits. Wie das funktioniert, ist einfacher zu erkennen, wenn man sich die Binärdarstellung der Werte ansieht.

Die dezimale 3 entspricht der binären 00000011 und die dezimale 1 ist binär 00000001. Bitweise Operatoren arbeiten mit jedem Bit. Die ganz rechts stehenden Bits sind beide 1, und die UND-Verknüpfung dieser beiden ist ebenfalls 1. Bewegt man sich nach links, sind die nächsten Bits 1 und 0, und die UND-Verknüpfung ergibt eine 0. Die restlichen Bits sind alle 0, und damit ist auch das Ergebnis für alle Bits 0. Mit anderen Worten ergibt jede Bitposition, an der beide Bits 1 sind, ebenfalls eine 1, anderenfalls eine 0. Also ergibt `11 & 01` eine 1.

Die Tabellen 2-6, 2-7 und 2-8 verdeutlichen diese Verknüpfungen.

Tabelle 2-6: Bitweises UND

Bit 1	Bit 2	Bit 1 and Bit 2
0	0	0
0	1	0
1	0	0
1	1	1

Tabelle 2-7: Bitweises ODER

Bit 1	Bit 2	Bit 1 or Bit 2
0	0	0
0	1	1
1	0	1
1	1	1

Tabelle 2-8: Bitweises EXKLUSIV-ODER

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
0	1	1
1	0	1
1	1	0

Alle bitweisen Ausdrücke arbeiten mit zwei Werten. Die Ausnahme bildet der Negationsoperator, der einfach jedes Bit umkehrt, d.h., aus einer 0 wird eine 1, und aus der 1 wird eine 0. Im obigen Beispiel wird der `byte`-Wert (8 Bit) 00000001 zu 11111110. Der `int`-Wert hat 16 Bits und seine Invertierung ergibt 15 Einsen und eine Null.

Siehe auch

Arduino-Referenz zu den bitweisen UND-, ODER und EXKLUSIV-ODER-Operatoren: *http://www.arduino.cc/en/Reference/Bitwise*

2.21 Operationen und Zuweisungen kombinieren

Problem

Sie wollen die zusammengesetzten Operatoren (compound operators) verstehen und nutzen. Es ist nicht ungewöhnlich, in veröffentlichtem Code Ausdrücke zu finden, die mit einer Anweisung mehr als eine Aufgabe erledigen. Sie wollen `a += b`, `a >>= b` und `a &= b` verstehen.

Lösung

Tabelle 2-9 zeigt die zusammengesetzten Zuweisungsoperatoren und die dazugehörigen vollständigen Ausdrücke.

Tabelle 2-9: Zusammengesetzte Operatoren

Operator	Beispiel	Vollständiger Ausdruck
`+=`	`value += 5;`	`value = value + 5; // Addiert 5 zu value hinzu`
`-=`	`value -= 4;`	`value = value - 4; // Subtrahiert 4 von value`
`*=`	`value *= 3;`	`value = value * 3; // Multipliziert value mit 3`
`/=`	`value /= 2;`	`value = value / 2; // Dividiert value durch 2`
`>>=`	`value >>= 2;`	`value = value >> 2; // Schiebt value um zwei Stellen (Bits) nach rechts`
`<<=`	`value <<= 2;`	`value = value << 2; // Schiebt value um zwei Stellen (Bits) nach links`
`&=`	`mask &= 2;`	`mask = mask & 2; // Binäre UND-Maske mit 2`
`|=`	`mask |= 2;`	`mask = mask | 2; // Binäre ODER-Maske mit 2`

Diskussion

Diese zusammengesetzten Anweisungen sind zur Laufzeit nicht effektiver als die vollständigen Ausdrücke, und für Programmier-Neulinge sind die vollständigen Ausdrücke verständlicher. Erfahrene Programmierer nutzen aber häufig diese Kurzformen, weshalb es hilfreich ist, diese Ausdrücke zu kennen.

Siehe auch

Einen Index mit Referenzseiten zu den zusammengesetzten Operatoren finden Sie unter *http://www.arduino.cc/en/Reference/HomePage*.

KAPITEL 3

Mathematische Operatoren nutzen

3.0 Einführung

Nahezu jeder Sketch nutzt mathematische Operatoren, um Variablenwerte zu verarbeiten. Dieses Kapitel enthält eine kurze Übersicht der wichtigsten mathematischen Operatoren. Wie das vorige Kapitel richtet sich auch dieses an Nichtprogrammierer bzw. Programmierer, die mit C oder C++ nicht vertraut sind. Weitere Details finden Sie in einem der C-Referenzwerke, die im Vorwort erwähnt werden.

3.1 Addieren, subtrahieren, multiplizieren und dividieren

Problem

Sie wollen mit den Werten Ihrer Sketches einfache mathematische Operationen durchführen. Sie wollen die Reihenfolge kontrollieren, in der die Operationen ausgeführt werden, und müssen möglicherweise auch mit unterschiedlichen Variablentypen arbeiten.

Lösung

Verwenden Sie den folgenden Code:

```
int myValue;
myValue = 1 + 2; // Addition
myValue = 3 - 2; // Subtraktion
myValue = 3 * 2; // Multiplikation
myValue = 3 / 2; // Division (das Ergebnis ist 1)
```

Diskussion

Addition, Subtraktion und Multiplikation von Integerwerten funktionieren, wie Sie es erwarten.

Stellen Sie sicher, dass die maximale Größe der Zielvariablen nicht überschritten wird. Siehe Rezept 2.2.

Bei der Integer-Division wird ein möglicher Rest einfach ignoriert. Im obigen Beispiel hat `myValue` nach der Division den Wert 1 (siehe Rezept 2.3, wenn Ihre Anwendung Fließkommazahlen verarbeiten muss):

```
int value =  1 + 2 * 3 + 4;
```

Zusammengesetzte Anweisungen wie oben erscheinen einem vielleicht nicht ganz eindeutig, doch der *Vorrang* (die Reihenfolge) jedes Operators ist genau definiert. Multiplikation und Division haben einen höheren Vorrang als Addition und Subtraktion. Das Ergebnis ist daher 11. Es ist durchaus ratsam, Klammern zu verwenden, um den Vorrang der Berechnung deutlich zu machen. `int value = 1 + (2 * 3) + 4;` führt zum gleichen Ergebnis und ist einfacher zu lesen.

Nutzen Sie Klammern auch, um den Vorrang zu ändern, wie im folgenden Beispiel:

```
int value =  ((1 + 2) * 3) + 4;
```

Das Ergebnis ist nun 13. Der Ausdruck in den inneren Klammern wird zuerst berechnet, d.h., 1 und 2 werden addiert und dann mit 3 multipliziert. Zum Schluss wird die 4 hinzuaddiert.

Siehe auch

Rezept 2.2; Rezept 2.3

3.2 Werte inkrementieren und dekrementieren

Problem

Sie möchten den Wert einer Variablen erhöhen oder verkleinern.

Lösung

Verwenden Sie den folgenden Code:

```
int myValue = 0;

myValue = myvalue + 1;  // Inkrementiert myValue um 1
myValue += 1;       // dito

myValue = myvalue - 1;  // Dekrementiert myValue um 1
myValue -= 1;       // dito

myValue = myvalue + 5;  // Addiert 5 zu myValue hinzu
myValue += 5;       // dito
```

Diskussion

Das Erhöhen und Verkleinern von Variablenwerten ist eine der gängigsten Programmieraufgaben, und das Arduino-Board kennt Operatoren, die das einfach machen. Einen Wert um 1 erhöhen, nennt man *inkrementieren*, ihn um 1 verkleinern, *dekrementieren*. Die lange Schreibweise sieht wie folgt aus:

```
myValue = myValue + 1; // Addiert eine 1 zu myValue
```

Doch Sie können Inkrement- und Dekrement-Operatoren auch mit einem Zuweisungsoperator kombinieren:

```
myValue += 1;       // Wie oben
```

Siehe auch

Rezept 3.1

3.3 Den Rest einer Division bestimmen

Problem

Sie wollen den Rest der Division zweier Werte ermitteln.

Lösung

Verwenden Sie das Symbol % (den Modulo-Operator), um den Rest zu ermitteln:

```
int myValue0 = 20 % 10; // Liefert den Rest von 20 durch 10
int myValue1 = 21 % 10; // Liefert den Rest von 21 durch 10
```

`myValue0` hat den Wert `0` (20 durch 10 ergibt den Rest 0). `myValue1` hat den Wert `1` (21 durch 10 ergibt einen Rest von 1).

Diskussion

Der Modulo-Operator ist überraschend nützlich, insbesondere wenn man prüfen will, ob ein Wert ein Vielfaches einer Zahl ist. Zum Beispiel kann der Code aus diesem Rezept so angepasst werden, dass er erkennt, ob der Wert ein Vielfaches von 10 ist:

```
int myValue;
//... Der myValue setzende Code steht hier
if (myValue % 10 == 0)
{
 Serial.println("Der Wert ist ein Vielfaches von 10");
}
```

Der obige Code berechnet den Rest der `myValue`-Variablen und vergleicht das Ergebnis mit 0 (siehe Rezept 2.17). Ist das Ergebnis 0, wird eine Meldung ausgegeben, die besagt, dass der Wert ein Vielfaches von 10 ist.

Hier ein weiteres Beispiel, bei dem aber eine 2 für den Modulo-Operator verwendet wird. Wir nutzen das Ergebnis, um zu prüfen, ob ein Wert gerade oder ungerade ist:

```
int myValue;
//... den Wert von myValue setzender Code steht hier
if (myValue % 2 == 0)
{
 Serial.println("Der Wert ist gerade");
}
else
{
 Serial.println("Der Wert ist ungerade");
}
```

Das folgende Beispiel berechnet aus einer Stundenangabe den Stundenwert für eine 24-Stunden-Uhr:

```
void printOffsetHour( int hourNow, int offsetHours)
{
  Serial.println((hourNow + offsetHours) % 24);
}
```

Siehe auch

Arduino-Referenz zum %- (Modulo-) Operator: *http://www.arduino.cc/en/Reference/Modulo*

3.4 Den Absolutwert ermitteln

Problem

Sie wollen den Absolutwert einer Zahl bestimmen.

Lösung

`abs(x)` berechnet den Absolutwert von `x`. Das folgende Beispiel ermittelt den Absolutwert der Differenz zweier analoger Eingänge (in Kapitel 5 erfahren Sie mehr über `analogRead()`):

```
int x = analogRead(0);
int y = analogRead(1);

if (abs(x-y) > 10)
{
 Serial.println("Die Analogwerte unterscheiden sich um mehr als 10");
}
```

Diskussion

`abs(x-y);` gibt den Absolutwert der Differenz zwischen `x` und `y` zurück. Sie wird für Integerwerte (und `long`) verwendet. Wie man den Absolutwert von Fließkommazahlen bestimmt, erläutert Rezept 2.3.

Siehe auch

Arduino-Referenz zu `abs`: *http://www.arduino.cc/en/Reference/Abs*

3.5 Zahlen auf einen Wertebereich beschränken

Problem

Sie wollen sicherstellen, dass sich ein Wert immer innerhalb einer unteren und oberen Grenze bewegt.

Lösung

`constrain(x, min, max)` liefert einen Wert zurück, der innerhalb der Grenzen von `min` und `max` liegt:

```
myConstrainedValue = constrain(myValue, 100, 200);
```

Diskussion

`myConstrainedValue` wird auf einen Wert gesetzt, der immer größer oder gleich 100 und kleiner oder gleich 200 ist. Ist `myValue` kleiner 100, ist das Ergebnis 100; liegt er über 200, wird er auf 200 gesetzt.

Tabelle 3-1 zeigt beispielhaft einige Ausgabewerte für ein `min` von 100 und ein `max` von 200.

Tabelle 3-1: constrain-Ausgabe mit min = 100 und max = 200

`myValue` **(Eingangswert)**	`constrain(myValue, 100, 200)`
99	100
100	100
150	150
200	200
201	200

Siehe auch

Rezept 3.6

3.6 Das Minimum oder Maximum bestimmen

Problem

Sie wollen das Minimum oder Maximum bei zwei oder mehr Werten bestimmen.

Lösung

`min(x,y)` gibt die kleinere und `max(x,y)` gibt die größere der beiden Zahlen zurück:

```
myValue = analogRead(0);
myMinValue = min(myValue, 200); // myMinValue enthält den kleineren Wert
                  // von myValue oder 200

myMaxValue = max(myValue, 100); // myMaxValue enthält den größeren Wert
                  // von myValue oder 100
```

Diskussion

Tabelle 3-2 zeigt einige Beispielergebnisse für ein `min` von 200. Die Tabelle zeigt, dass das Ergebnis der Eingabe (`myValue`) entspricht, bis der Wert die 200 übersteigt.

Tabelle 3-2: Ergebnisse für min(myValue, 200)

`myValue` **(Eingangswert)**	`min(myValue, 200)`
99	99
100	100
150	150
200	200
201	200

Tabelle 3-3 zeigt das Ergebnis für ein `max` von 100. Die Tabelle zeigt, dass das Ergebnis der Eingabe (`myValue`) entspricht, solange der Wert größer oder gleich 100 ist.

Tabelle 3-3: Ergebnisse für max(myValue, 100)

`myValue` **(Eingangswert)**	`max(myValue, 100)`
99	100
100	100
150	150
200	200
201	201

Nutzen Sie `min`, um eine Obergrenze festzulegen. Das klingt nicht gerade intuitiv, aber da sich `min` beim Eingangswert und dem Minimum immer für den kleineren entscheidet, ist das Ergebnis nie höher als das Minimum (200 in unserem Beispiel).

In gleicher Weise können Sie max nutzen, um eine untere Grenze festzulegen. Das Ergebnis von max ist niemals kleiner als der Maximalwert (100 in unserem Beispiel).

Wenn Sie die min- oder max-Werte für mehr als zwei Werte bestimmen müssen, können Sie die Funktionsaufrufe wie folgt kaskadieren:

```
// myMinValue enthält den kleinsten der drei Analogwerte:
int myMinValue = min(analogRead(0), min(analogRead(1), analogRead(2)) );
```

Bei diesem Beispiel wird zuerst der Minimalwert für die Analogports 1 und 2 bestimmt. Das Ergebnis wird dann wiederum für die Berechnung des Minimums mit Port 0 genutzt. Das lässt sich auf so viele Werte wie nötig ausweiten, allerdings müssen Sie dabei auf die korrekte Klammerung achten. Das folgende Beispiel ermittelt das Minimum von vier Werten:

```
int myMaxValue = max(analogRead(0), max(analogRead(1), max(analogRead(2),
                                    analogRead(3))));
```

Siehe auch

Rezept 3.5

3.7 Eine Zahl potenzieren

Problem

Sie wollen eine Zahl potenzieren.

Lösung

pow(x, y) gibt den Wert von x hoch y zurück:

```
myValue = pow(3,2);
```

Der obige Code berechnet 3^2, myValue ist also 9.

Diskussion

Die pow-Funktion kann mit Integer- und Fließkommawerten arbeiten und liefert das Ergebnis als Fließkommazahl zurück:

```
Serial.print(pow(3,2)); // gibt 9.00 aus
int z = pow(3,2);
Serial.println(z);    // gibt 9 aus
```

Im ersten Fall wird 9.00 ausgegeben und im zweiten 9. Die Werte sind nicht gleich, weil das erste print eine Fließkommazahl ausgibt, während die Zahl bei der zweiten Ausgabe als Integerwert betrachtet wird, weshalb keine Nachkommastellen ausgegeben werden. Wenn Sie die pow-Funktion nutzen, sollten Sie auch Rezept 2.3 lesen, um den Unterschied zwischen Fließkomma- und Integerwerten zu verstehen.

Hier ein Beispiel für eine Bruchpotenz:

```
float s = pow(2, 1.0 / 12); // Die zwölfte Wurzel von 2
```

Die zwölfte Wurzel von zwei entspricht 2 hoch 0,083333. Der resultierende Wert für s ist 1,05946 (das entspricht der Frequenz, die den Unterschied zwischen zwei nebeneinanderliegenden Töne auf dem Klavier ausmacht).

3.8 Die Quadratwurzel berechnen

Problem

Sie wollen die Quadratwurzel einer Zahl berechnen.

Lösung

Die Funktion `sqrt(x)` gibt die Quadratwurzel von `x` zurück:

```
Serial.print( sqrt(9) ); // gibt 3.00 aus
```

Diskussion

Die `sqrt`-Funktion gibt eine Fließkommazahl zurück (siehe auch die Diskussion der `pow`-Funktion in Rezept 3.7).

3.9 Fließkommazahlen auf- und abrunden

Problem

Sie wollen eine Fließkommazahl auf den nächsten kleineren oder größeren Integerwert ab- bzw. aufrunden (`floor` oder `ceil`).

Lösung

`floor(x)` gibt den größten ganzzahligen Wert zurück, der nicht größer als `x` ist. `ceil(x)` gibt den kleinsten ganzzahligen Wert zurück, der nicht kleiner als `x` ist.

Diskussion

Diese Funktionen werden zum Runden von Fließkommazahlen genutzt. Nutzen Sie `floor(x)`, um den größten Integerwert zu bestimmen, der nicht größer als `x` ist. Verwenden Sie `ceil`, um den kleinsten Integerwert zu ermitteln, der größer ist als `x`.

Hier einige Beispiele für `floor`:

```
Serial.println( floor(1) );   // Gibt 1.00 aus
Serial.println( floor(1.1) ); // Gibt 1.00 aus
Serial.println( floor(0) );   // Gibt 0.00 aus
```

```
Serial.println( floor(.1) );  // Gibt 0.00 aus
Serial.println( floor(-1) );  // Gibt -1.00 aus
Serial.println( floor(-1.1) ); // Gibt -2.00 aus
```

Hier einige Beispiele für `ceil`:

```
Serial.println( ceil(1) );   // Gibt 1.00 aus
Serial.println( ceil(1.1) );  // Gibt 2.00 aus
Serial.println( ceil(0) );   // Gibt 0.00 aus
Serial.println( ceil(.1) );   // Gibt 1.00 aus
Serial.println( ceil(-1) );   // Gibt -1.00 aus
Serial.println( ceil(-1.1) ); // Gibt -1.00 aus
```

Auf den nächstgelegenen Integerwert können Sie wie folgt runden:

```
if (floatValue > 0.0)
 result = floor(floatValue + 0.5);
else
 result = ceil(floatValue - 0.5);
```

Sie können Nachkommastellen auch »abschneiden«, indem Sie ein *Casting* (keine Konvertierung) nach `int` vornehmen, aber dabei wird nicht korrekt gerundet. Negative Zahlen wie –1,9 sollten auf –2 abgerundet werden, doch beim `int`-Casting wird auf –1 aufgerundet. Das gleiche Problem haben Sie auch bei positiven Zahlen: 1,9 sollte auf 2 aufgerundet werden, wird aber auf 1 abgerundet. Für korrekte Ergebnisse müssen Sie `floor` und `ceil` verwenden.

3.10 Trigonometrische Funktionen nutzen

Problem

Sie wollen den den Sinus, Kosinus oder Tangens für einen in Grad oder Bogenmaß angegebenen Winkel bestimmen.

Lösung

`sin(x)` gibt den Sinus, `cos(x)` den Kosinus und `tan(x)` den Tangens des Winkels `x` zurück.

Diskussion

Winkel werden im Bogenmaß angegeben, und das Ergebnis ist eine Fließkommazahl (siehe Rezept 2.3). Das folgende Beispiel veranschaulicht den Einsatz der trigonometrischen Funktionen:

```
float deg = 30;          // Winkel in Grad
float rad  = deg * PI / 180;   // in Bogenmaß umwandeln
Serial.println(rad);       // Bogenmaß ausgeben
Serial.println (sin(rad));    // Sinus ausgeben
Serial.println (cos(rad));    // Kosinus ausgeben
```

Das obige Beispiel wandelt den Winkel in Bogenmaß um und gibt den Sinus und Kosinus aus. Hier die Ausgabe (mit zusätzlichen Anmerkungen versehen):

```
0.52  30 Grad entspricht dem Bogenmaß 0,5235988 ausgegeben werden nur die ersten beiden
Nachkommastellen
0.50  Der Sinus von 30 ist 0,5000000, hier auf 2 Nachkommastellen genau
0.87  Der Kosinus ist 0,8660254, was auf 0,87 aufgerundet wird
```

Zwar berechnet der Sketch die Werte mit der Genauigkeit von Fließkommazahlen, doch die Routine `Serial.print` gibt hier nur die ersten beiden Nachkommastellen aus.

Die Umwandlung von Bogenmaß in Grad und wieder zurück ist Trigonometrie aus dem Lehrbuch. `PI` ist die vertraute Konstante π (3,14159265...). `PI` und `180` sind beides Konstanten, und Arduino stellt einige vorberechnete Konstanten zur Verfügung, mit denen Sie Umwandlungen nach Grad und Bogenmaß durchführen können:

```
rad = deg * DEG_TO_RAD;  // Grad in Bogenmaß
deg = rad * RAD_TO_DEG;  // Bogenmaß in Grad
```

Die Nutzung von `deg * DEG_TO_RAD` mag effizienter aussehen als `deg * PI / 180`, ist sie aber nicht, weil der Arduino-Compiler clever genug ist, um zu erkennen, dass `PI / 180` eine Konstante ist (deren Wert sich nie ändert). Er fügt daher das Ergebnis der Division von `PI` durch 180 ein, was genau dem Wert der Konstanten `DEG_TO_RAD` (0,017453292519...) entspricht. Sie können also den von Ihnen bevorzugten Ansatz verwenden.

Siehe auch

Arduino-Referenzen zu `sin` (*http://www.arduino.cc/en/Reference/Sin*), `cos` (*http://arduino.cc/en/Reference/Cos*) und `tan` (*http://arduino.cc/en/Reference/Tan*)

3.11 Zufallszahlen erzeugen

Problem

Sie benötigen Zufallszahlen zwischen Null und einem festgelegten Maximum, oder zwischen einem vorgegebenen Minimum und Maximum.

Lösung

Verwenden Sie die Funktion `random`. Beim Aufruf von `random` mit einem einzelnen Parameter wird die Obergrenze festgelegt. Die zurückgelieferten Werte liegen zwischen Null und 1 unter dieser Obergrenze:

```
random(max);    // Gibt eine Zufallszahl zwischen 0 und max-1 zurück
```

Beim Aufruf von `random` mit zwei Parametern wird die Unter- und Obergrenze festgelegt. Die zurückgelieferten Werte reichen von der Untergrenze (einschließlich) bis zu eins unter der Obergrenze:

```
random(min, max); // Gibt eine Zufallszahl zwischen min und max-1 zurück
```

Diskussion

Auch wenn man bei den zurückgelieferten Zahlen kein offensichtliches Muster erkennen kann, sind die Werte nicht zufällig. Bei jedem Start des Sketches wird die gleiche Folge erzeugt. Bei vielen Anwendungen spielt das aber keine Rolle. Wenn Sie bei jedem Start des Sketches unterschiedliche Folgen benötigen, verwenden Sie die Funktion `randomSeed(seed)` mit einem anderen seed-Wert. (Wenn Sie den gleichen seed-Wert verwenden, erhalten Sie auch die gleiche Folge). Diese Funktion initialisiert den Zufallszahlengenerator mit einem Startwert, der auf dem übergebenen seed-Parameter basiert :

```
randomSeed(1234); // Startfolge der Zufallszahlen ändern.
```

Hier ein Beispiel, das die unterschiedlichen Formen der Zufallszahlengenerierung nutzt, die bei Arduino zur Verfügung stehen:

```
// Random
// Generierung von Zufallszahlen

int randNumber;

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

 // Zufallszahlen ohne seed-Wert ausgeben
 Serial.println("20 Zufallszahlen zwischen 0 und 9");
 for(int i=0; i < 20; i++)
 {
  randNumber = random(10);
  Serial.print(randNumber);
  Serial.print(" ");
 }
 Serial.println();
 Serial.println("20 Zufallszahlen zwischen 0 und 9");
 for(int i=0; i < 20; i++)
 {
  randNumber = random(2,10);
  Serial.print(randNumber);
  Serial.print(" ");
 }

 // Zufallszahl mit immer gleichem seed-Wert ausgeben
 randomSeed(1234);
 Serial.println();
 Serial.println("20 Zufallszahlen zwischen 0 und 9 mit konstantem seed");
 for(int i=0; i < 20; i++)
 {
  randNumber = random(10);
  Serial.print(randNumber);
  Serial.print(" ");
 }

 // Zufallszahlen mit unterschiedlichem seed-Wert ausgeben
 randomSeed(analogRead(0)); // Nicht angeschlossenen Analogport auslesen
 Serial.println();
```

```
 Serial.println("20 Zufallszahlen zwischen 0 und 9 mit unterschiedlichem seed");
 for(int i=0; i < 20; i++)
 {
  randNumber = random(10);
  Serial.print(randNumber);
  Serial.print(" ");
 }
 Serial.println();
 Serial.println();
}

void loop()
{
}
```

Hier die Ausgabe des Codes:

```
20 Zufallszahlen zwischen 0 und 9
7 9 3 8 0 2 4 8 3 9 0 5 2 2 7 3 7 9 0
20 Zufallszahlen zwischen 0 und 9
9 3 7 7 2 7 5 8 2 9 3 4 2 5 4 3 5 7 5 7
20 Zufallszahlen zwischen 0 und 9 mit konstantem seed
8 2 8 7 1 8 0 3 6 5 9 0 3 4 3 1 2 3 9 4
20 Zufallszahlen zwischen 0 und 9 mit unterschiedlichem seed
0 9 7 4 4 7 7 4 4 9 1 6 0 2 3 1 5 9 1 1
```

Wenn Sie den Reset-Button des Arduino drücken, um den Sketch neu zu starten, bleiben die Zufallszahlen der ersten drei Zeilen unverändert. Nur die letzte Zeile ändert sich bei jedem Start des Sketches, weil der seed-Wert auf einen neuen Wert gesetzt wird, indem ein ungenutzter Analogport ausgelesen und als Parameter an `randomSeed` übergeben wird. Wenn Sie den Analogport 0 für etwas anderes nutzen, müssen Sie einen ungenutzten Analogport als Argument an `analogRead` übergeben.

Siehe auch

Arduino-Referenz für `random` (*http://www.arduino.cc/en/Reference/Random*) und `random Seed` (*http://arduino.cc/en/Reference/RandomSeed*)

3.12 Bits setzen und lesen

Problem

Sie möchten ein bestimmtes Bit in einer numerischen Variablen auslesen oder setzen.

Lösung

Nutzen Sie die folgenden Funktionen:

`bitSet(x, bitPosition)`
: Setzt (schreibt eine 1 an) die gegebene `bitPosition` der Variablen `x`

`bitClear(x, bitPosition)`
: Löscht (schreibt eine 0 an) die gegebene `bitPosition` der Variablen `x`

`bitRead(x, bitPosition)`
: Gibt den Wert (0 oder 1) des Bits an der gegebenen `bitPosition` der Variablen `x` zurück

`bitWrite(x, bitPosition, value)`
: Setzt den angegebenen Wert (0 oder 1) des Bits an `bitPosition` der Variablen `x`

`bit(bitPosition)`
: Gibt den Wert der gegebenen Bitposition zurück: `bit(0)` ist 1, `bit(1)` ist 2, `bit(2)` ist 4 und so weiter

Bei dieser Funktion ist `bitPosition` `0` das niederwertigste (ganz rechts stehende) Bit.

Der folgende Sketch nutzt diese Funktionen, um die Bits einer 8-Bit-Variablen namens `flags` zu bearbeiten:

```
// bitFunctions
// Verwendung der Bitfunktionen


byte flags = 0; // Diese Beispiele setzen, löschen oder lesen die Bits in der Variablen flags.

// bitSet-Beispiel
void setFlag( int flagNumber)
{
 bitSet(flags, flagNumber);
}

// bitClear-Beispiel
void clearFlag( int flagNumber)
{
 bitClear(flags, flagNumber);
}

// bitPosition-Beispiel

int getFlag( int flagNumber)
{
 return bitRead(flags, flagNumber);
}

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

void loop()
{
  showFlags();
  setFlag(2); // Ein paar Flags setzen;
  setFlag(5);
  showFlags();
  clearFlag(2);
```

```
  showFlags();

  delay(10000); // sehr lange warten
}

// Gibt Status des Flags aus
void showFlags()
{
  for(int flag=0; flag < 8; flag++)
  {
   if (getFlag(flag) == true)
     Serial.print("* Bit gesetzt fuer Flag ");    else
     Serial.print("Bit geloescht fuer Flag ");

   Serial.println(flag);
  }
  Serial.println();
}
```

Der Code gibt Folgendes aus:

```
Bit geloescht fuer Flag 0
Bit geloescht fuer Flag 1
Bit geloescht fuer Flag 2
Bit geloescht fuer Flag 3
Bit geloescht fuer Flag 4
Bit geloescht fuer Flag 5
Bit geloescht fuer Flag 6
Bit geloescht fuer Flag 7

Bit geloescht fuer Flag 0
Bit geloescht fuer Flag 1
* Bit gesetzt fuer Flag 2
Bit geloescht fuer Flag 3
Bit geloescht fuer Flag 4
* Bit gesetzt fuer Flag 5
Bit geloescht fuer Flag 6
Bit geloescht fuer Flag 7

Bit geloescht fuer Flag 0
Bit geloescht fuer Flag 1
Bit geloescht fuer Flag 2
Bit geloescht fuer Flag 3
Bit geloescht fuer Flag 4
* Bit gesetzt fuer Flag 5
Bit geloescht fuer Flag 6
Bit geloescht fuer Flag 7
```

Diskussion

Das Auslesen und Setzen von Bits ist eine typische Aufgabe, und viele Arduino-Bibliotheken nutzen diese Funktionalität. Ein typisches Einsatzgebiet für Bitoperationen ist das effiziente Speichern und Lesen von Binärwerten (Ein/Aus, Wahr/Falsch, 1/0, high/low, etc.).

Arduino definiert die Konstanten `true` und `HIGH` als 1 und `false` und `LOW` als 0.

Den Zustand von acht Schaltern können Sie in einem einzigen 8-Bit-Wert speichern, statt acht Byte oder Integerwerte zu benötigen. Das Beispiel in diesem Rezept zeigt, wie man acht Werte in einem Byte einzeln setzen und löschen kann.

Der Begriff *Flag* wird in der Programmierung für Werte verwendet, die den Status eines bestimmten Aspekts des Programms festhalten. Im obigen Sketch werden die Flag-Bits mit `bitRead` ausgelesen und mit `bitSet` oder `bitClear` gesetzt bzw. gelöscht. Die Funktionen benötigen zwei Parameter: Der erste ist der zu lesende oder zu schreibende Wert (in diesem Beispiel also `flags`), und der zweite gibt die Bitposition an, die gelesen oder geschrieben werden soll. Die Bitposition 0 ist das niederwertigste (ganz rechts stehende) Bit, Position 1 die zweite Position von rechts und so weiter.

```
bitRead(2, 1); // Ergibt 1; 2 ist binär 10 und das Bit an Position 1 ist 1
bitRead(4, 1); // Ergibt 0; 4 ist binär 100 und das Bit an Position 1 ist 0
```

Es gibt auch eine Funktion namens `bit`, die die Wertigkeit jeder Bitposition zurückgibt:

```
bit(0) ist 1;
bit(1) ist 2;
bit(2) ist 4;
...
bit(7) ist 128
```

Siehe auch

Arduino-Referenz zu den Bit- und Byte-Funktionen:

`lowByte`
: *http://www.arduino.cc/en/Reference/LowByte*

`highByte`
: *http://arduino.cc/en/Reference/HighByte*

`bitRead`
: *http://www.arduino.cc/en/Reference/BitRead*

`bitWrite`
: *http://arduino.cc/en/Reference/BitWrite*

`bitSet`
: *http://arduino.cc/en/Reference/BitSet*

`bitClear`
: *http://arduino.cc/en/Reference/BitClear*

`bit`
: *http://arduino.cc/en/Reference/Bit*

3.13 Bits verschieben (Shifting)

Problem

Sie müssen Bitoperationen durchführen, die die Bits in einem `byte`, `int` oder `long` nach links oder rechts verschieben.

Lösung

Nutzen Sie die Operatoren << (Bitshift links) und >> (Bitshift rechts), um die Bits in einem Wert zu verschieben.

Diskussion

Das folgende Code-Fragment setzt die Variable `x` auf 6, verschiebt die Bits dann um eine Stelle nach links und gibt anschließend den neuen aus (`12`). Dieser Wert wird dann um zwei Stellen nach rechts verschoben (und das Ergebnis ist `3`):

```
int x = 6;
int result = x << 1;  // 6 um 1 nach links verschoben ergibt 12
Serial.println(result);
int result = x >> 2;  // 12 um 2 nach rechts verschoben ergibt 3;
Serial.println(result);
```

Das funktioniert wie folgt: Schiebt man 6 um eine Stelle nach links, dann ergibt das 12, weil die Dezimalzahl 6 als Binärzahl 0110 ist, und wenn man die Ziffern um eine Stelle nach links verschiebt, erhält man 1100 (dezimal 12). Verschiebt man 1100 um zwei Stellen nach rechts, erhält man 0011 (dezimal 3). Sie werden bemerken, dass die Verschiebung um *n* Stellen nach links einer Multiplikation des Wertes mal 2 hoch *n* entspricht. Die Verschiebung um *n* nach rechts entspricht hingegen der Division durch 2 hoch *n*. Mit anderen Worten sind die folgenden Ausdrücke gleich:

`x << 1` ist gleich `x * 2`.
`x << 2` ist gleich `x * 4`.
`x << 3` ist gleich `x * 8`.
`x >> 1` ist gleich `x / 2`.
`x >> 2` ist gleich `x / 4`.
`x >> 3` ist gleich `x / 8`.

Der Arduino-Controllerchip kann Bits effizienter verschieben als multiplizieren und dividieren, und gelegenlich werden Sie auch Code sehen, der Bits verschiebt, um zu multiplizieren oder zu dividieren:

```
int c = (a << 1) + (b >> 2); //entspricht (a * 2) + ( b / 4)
```

Der Ausdruck `(a << 1) + (b >> 2);` hat keine große Ähnlichkeit mit `(a * 2) + (b / 4);`, doch beide Ausdrücke machen genau das gleiche. Allerdings ist der Arduino-Compiler clever genug, die Multiplikation eines Integerwertes mit einer Konstanten, die ein Vielfaches einer Zweierpotenz ist, zu erkennen und daraus Maschinencode zu erzeugen, der mit

Shiftoperationen arbeitet. Der mit arithmetischen Operatoren arbeitende Quellcode ist für Menschen einfacher zu lesen, weshalb man diese Variante bevorzugt, wenn man multiplizieren und dividieren will.

Siehe auch

Arduino-Referenz zu Bit- und Bytefunktionen: `lowByte`, `highByte`, `bitRead`, `bitWrite`, `bitSet`, `bitClear` und `bit` (siehe Rezept 3.12)

3.14 Höher- und niederwertige Bytes aus int oder long extrahieren

Problem

Sie wollen das höher- oder niederwertige Byte aus einem Integerwert extrahieren, z.B. wenn Sie Integerwerte als Bytes über einen seriellen Port oder einen anderen Kommunikationskanal senden wollen.

Lösung

Verwenden Sie `lowByte(i)`, um das niederwertige Byte aus einem Integerwert zu extrahieren. Verwenden Sie `highByte(i)`, um das höherwertige Byte des Integerwerts zu bestimmen.

Der folgende Sketch wandelt einen Integerwert in sein höher- und niederwertiges Byte um:

```
//ByteOperators

int intValue = 258; // 258 ist hexadezimal 0x102


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

void loop()
{
 int loWord,hiWord;
 byte loByte, hiByte;


 hiByte = highByte(intValue);
 loByte = lowByte(intValue);

 Serial.println(intValue,DEC);
 Serial.println(intValue,HEX);
 Serial.println(loByte,DEC);
 Serial.println(hiByte,DEC);

 delay(10000); // Sehr lange warten
}
```

Diskussion

Der Beispiel-Sketch gibt `intValue` aus, gefolgt von dessen nieder- und höherwertigem Byte:

```
258   // der umzuwandelnde Integerwert
102   // in hexadezimaler Notation
2    // das niederwertige Byte
1    // das höherwertige Byte
```

Um die Bytes eines `long`-Werts zu extrahieren, muss der 32-Bit-long-Wert zuerst in zwei 16-Bit-*Wörter* zerlegt werden, die dann wie im obigen Code konvertiert werden können. Während diese Zeilen geschrieben werden, kennt die Standard-Arduino-Bibliothek diese Operation für `long`s nicht, aber Sie können dazu die folgenden Zeilen in Ihre Sketches aufnehmen:

```
#define highWord(w) ((w) >> 16)
#define lowWord(w) ((w) & 0xffff)
```

Das sind sog. *Makros*: `hiWord` führt eine 16-Bit-Shiftoperation durch, um einen 16-Bit-Wert zu erzeugen und `lowWord` maskiert die niederwertigen 16 Bits mit Hilfe des bitorientierten UND-Operators (siehe Rezept 2.20).

Die Anzahl der Bits in einem `int` variiert je nach Plattfom. Bei Arduino sind es 16 Bit, bei anderen Umgebungen hingegen 32 Bit. Der Begriff *Word* wird hier für einen 16-Bit-Wert verwendet.

Der folgende Code wandelt den 32-Bit-Hexwert 0x1020304 in seine beiden höher- und niederwertigen 16-Bit-Werte um:

```
loword = lowWord(longValue);
hiword = highWord(longValue);
Serial.println(loword,DEC);
Serial.println(hiword,DEC);
```

Ausgegeben werden die folgenden Werte:

```
772  // 772 entspricht 0x0304 hexadezimal
258  // 258 entspricht 0x0102 hexadezimal
```

772 dezimal ist 0x0304 hexadezimal, was dem niederwertigen Wort (16 Bit) des `long-Value-Werts` `0x102`**`0304`** entspricht. Vielleicht erkennen Sie 258 aus dem ersten Teil des Rezepts wieder. Er ist die Kombination aus einer 1 im höher- und einer 2 im niederwertige Byte (hexadezimal 0x0102).

Siehe auch

Arduino-Referenz zu den Bit- und Bytefunktionen: `lowByte`, `highByte`, `bitRead`, `bitWrite`, `bitSet`, `bitClear` und `bit` (siehe Rezept 3.12).

3.15 int- oder long-Werte aus höher- und niederwertigen Bytes bilden

Problem

Sie wollen einen16-Bit- (`int`) oder 32-Bit- (`long`) Integerwert aus einzelnen Bytes bilden, z.B. wenn Sie Integerwerte in einzelnen Bytes über eine serielle Kommunikationsleitung empfangen. Das ist die Umkehroperation zu Rezept 3.14.

Lösung

Nutzen Sie die Funktion `word(h,l)`, um zwei Bytes zu einem Arduino-Integerwert zusammenzufassen. Nachfolgend wird der Code aus Rezept 3.14 dahingehend erweitert, dass er die jeweiligen höher- und niederwertigen Bytes wieder zu einem Integerwert zusammensetzt:

```
//ByteOperators

int intValue = 0x102; // 258


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

void loop()
{
 int loWord,hiWord;
 byte loByte, hiByte;


 hiByte = highByte(intValue);
 loByte = lowByte(intValue);

 Serial.println(intValue,DEC);
 Serial.println(loByte,DEC);
 Serial.println(hiByte,DEC);

 loWord = word(hiByte, loByte);  // Bytes wieder zu einem Wort zusammenfassen
 Serial.println(loWord,DEC);
 delay(10000); // Sehr lange warten
}
```

Diskussion

Der Ausdruck `word(high,low)` fasst ein höher- und ein niederwertiges Byte zu einem 16-Bit-Wert zusammen. Der Code nimmt die in Rezept 3.14 erzeugten höher- und niederwertigen Bytes und setzt sie wieder zu einem Wort zusammen. Die Ausgabe zeigt den Integerwert, das niederwertige und das höherwertige Byte und dann den wieder zusammengesetzten Integerwert:

```
258
2
1
258
```

Arduino besitzt (während dies geschrieben wird) keine Funktion, die einen 32-Bit-long-Wert aus zwei 16-Bit-Worten erzeugt, aber Sie können dazu ein eigenes `makeLong()`-Makro nutzen. Fügen Sie einfach die folgende Zeile in Ihren Sketch ein:

```
#define makeLong(hi, low) ((hi) << 16 & (low))
```

Das definiert ein Makro, das den höherwertigen Teil um 16 Bits nach links verschiebt und dann den niederwertigen Teil hinzufügt:

```
#define makeLong(hi, low) (((long) hi) << 16 | (low))
#define highWord(w) ((w) >> 16)
#define lowWord(w) ((w) & 0xffff)

// Testwert deklarieren
long longValue = 0x1020304; // Dezimal 16909060
                // Binär 00000001 00000010 00000011 00000100

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

void loop()
{
 int loWord,hiWord;

 Serial.println(longValue,DEC); // Gibt 16909060 aus
 loWord = lowWord(longValue);   // Wandelt long in zwei Wörter um
 hiWord = highWord(longValue);
 Serial.println(loWord,DEC);    // Gibt 772 aus
 Serial.println(hiWord,DEC);    // Gibt 258 aus
 longValue = makeLong( hiWord, loWord); // Wandelt die Wörter wieder in long um
 Serial.println(longValue,DEC); // Gibt wieder 16909060 aus

 delay(10000); // Sehr lange warten
}
```

Hier die Ausgabe:

```
16909060
772
258
16909060
```

Siehe auch

Arduino-Referenz zu Bit- und Bytefunktionen: `lowByte`, `highByte`, `bitRead`, `bitWrite`, `bitSet`, `bitClear` und `bit` (siehe Rezept 3.12)

KAPITEL 4

Serielle Kommunikation

4.0 Einführung

Die serielle Kommunikation bietet eine einfache und flexible Möglichkeit, Ihr Arduino-Board mit Ihrem Computer und anderen Geräten interagieren zu lassen. Dieses Kapitel erläutert, wie man auf diese Weise Informationen senden und empfangen kann.

In Kapitel 1 wurde beschrieben, wie man den seriellen Port des Arduino mit dem Computer verbindet, um Sketches hochzuladen. Der Upload-Prozess sendet Daten von Ihrem Computer an den Arduino, und der Arduino sendet Statusmeldungen zurück an den Computer, um zu bestätigen, dass der Transfer funktioniert. Die hier vorgestellten Rezepte zeigen, wie Sie diesen Kommunikationslink nutzen können, um beliebige Informationen zwischen dem Arduino und Ihrem Computer (oder einem anderen seriellen Gerät) zu senden und zu empfangen.

Die serielle Kommunikation ist auch ein praktisches Tool zur Fehlersuche (Debugging). Sie senden Debugging-Nachrichten vom Arduino an den Computer und geben sie auf dem Bildschirm oder einem externen LC-Display aus.

Die Arduino-IDE (beschrieben in Rezept 1.3) stellt einen seriellen Monitor zur Verfügung (siehe Abbildung 4-1), der vom Arduino gesendete serielle Daten ausgibt.

Sie können Daten über den seriellen Monitor an den Arduino senden, indem Sie Text in das Textfeld links neben dem Send-Button eingeben. Die Baudrate (die Geschwindigkeit, mit der die Daten übertragen werden, gemessen in Bits pro Sekunde) wird über eine Dropdown-Box am unteren rechten Rand ausgewählt. Sie können die Dropdown-Box namens »No line ending« nutzen, um automatisch ein Carriage Return (Wagenrücklauf) oder eine Kombination aus Carriage Return und Linefeed (Zeilenvorschub) an das Ende jeder Nachricht anzuhängen, sobald der Send-Button angeklickt wird. Ändern Sie dazu einfach »No line ending« in die gewünschte Option.

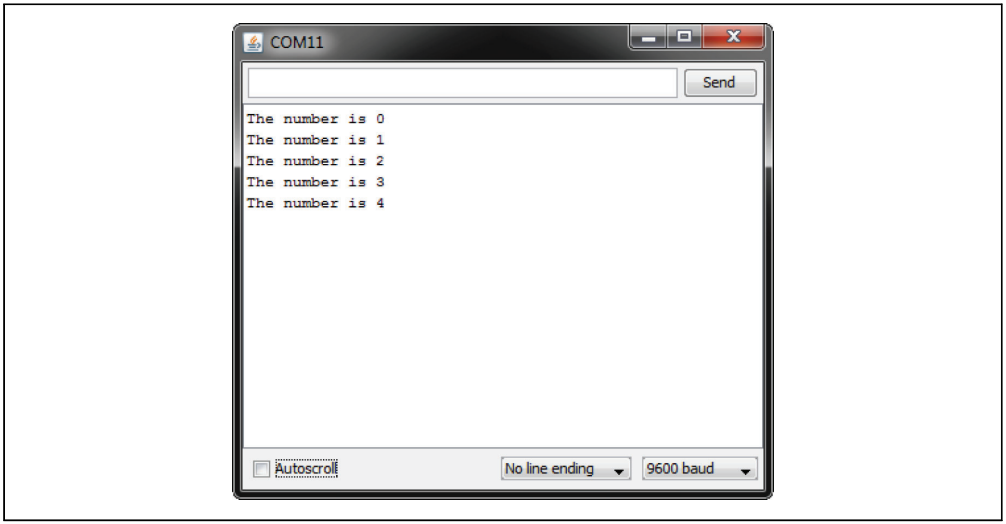


Abbildung 4-1: Serieller Monitor des Arduino

Ihr Arduino-Sketch kann den seriellen Port nutzen, um indirekt (üblicherweise über ein Proxy-Programm in einer Sprache wie Processing) auf alle Ressourcen (Speicher, Bildschirm, Tastatur, Maus, Netzwerk etc.) Ihres Computers zugreifen zu können. Ihr Computer kann wiederum die serielle Schnittstelle nutzen, um mit Sensoren oder anderen, mit dem Arduino verbundenen Geräten zu interagieren.

Die Implementierung einer seriellen Kommunikation verlangt Hard- und Software. Die Hardware sorgt für die elektrischen Signale zwischen dem Arduino und dem Gerät, mit dem er sich unterhält. Die Software nutzt die Hardware, um Bytes oder Bits zu senden, die von der angeschlossenen Hardware verstanden werden. Arduinos serielle Bibliotheken verstecken einen Großteil der Hardware-Komplexität vor Ihnen, es ist aber hilfreich, die Grundlagen zu verstehen, besonders wenn Sie bei Ihren Projekten Probleme mit der seriellen Kommunikation untersuchen müssen.

Serielle Hardware

Die serielle Hardware sendet und empfängt Daten in Form elektrischer Impulse, die eine sequentielle Folge von Bits darstellen. Die Nullen und Einsen, die die Informationen enthalten, aus denen ein Byte besteht, können auf verschiedene Art repräsentiert werden. Das von Arduino verwendete Schema ist 0 Volt für den Bitwert 0 und 5 (oder 3,3) Volt für den Bitwert 1.

Die Verwendung von 0 Volt (für 0) und 5 Volt (für 1) ist weit verbreitet. Man spricht hier vom *TTL-Level* (Pegel), weil Signale in einer der ersten Implementierungen digitaler Logik, der sog. Transistor-Transistor Logik (TTL), in dieser Form repräsentiert wurden.

Boards wie das Uno, Duemilanove, Diecimila, Nano und Mega besitzen einen Chip, der den seriellen Hardware-Port des Arduino-Chips in Universal Serial Bus (USB) umwandelt, um die Verbindung mit dem seriellen Port herzustellen. Andere Boards wie das Mini, Pro, Pro Mini, Boarduino, Sanguino und Modern Device Bare Bones Board unterstützen USB nicht und benötigen für die Verbindung zum Computer einen Adapter, der TTL in USB umwandelt. Weitere Details zu diesen Boards finden Sie unter *http://www.arduino.cc/en/Main/Hardware*.

Einige beliebte USB-Adapter sind:

- Mini USB Adapter (*http://arduino.cc/en/Main/MiniUSB*)
- USB Serial Light Adapter (*http://arduino.cc/en/Main/USBSerial*)
- FTDI USB TTL Adapter (*http://www.ftdichip.com/Products/FT232R.htm*)
- Modern Device USB BUB-Board (*http://shop.moderndevice.com/products/usb-bub*)
- Seeedstudio UartSBee (*http://www.seeedstudio.com/depot/uartsbee-v31-p-688.html*)

Einige serielle Geräte verwenden den RS-232-Standard für die serielle Verbindung. Sie haben üblicherweise einen Neun-Pin-Stecker, und ein Adapter wird benötigt, um sie mit dem Arduino verwenden zu können. RS-232 ist ein altehrwürdiges Kommunikationsprotokoll, dessen Spannungspegel mit den Digitalpins des Arduino nicht kompatibel sind.

Sie können Arduino-Boards kaufen, die für die RS-232-Signalpegel gebaut sind, etwa das Freeduino Serial v2.0 (*http://www.nkcelectronics.com/freeduino-serial-v20-board-kit-arduino-diecimila-compatib20.html*).

Hier einige RS-232-Adapter, die RS-232-Signale mit den 5 (oder 3,3) Volt der Arduino-Pins verbinden:

- RS-232 nach TTL 3V–5.5V Adapter (*http://www.nkcelectronics.com/rs232-to-ttl-converter-board-33v232335.html*)
- P4 RS232 nach TTL Serial Adapter Kits (*http://shop.moderndevice.com/products/p4*)
- RS232 Shifter SMD (*http://www.sparkfun.com/commerce/product_info.php?products_id=449*)

Ein Standard-Arduino verfügt über einen einzigen seriellen Hardware-Port, doch die serielle Kommunikation ist auch über Software-Bibliotheken möglich, die zusätzliche Ports (Kommunikationskanäle) emulieren, um mehr als ein Gerät anschließen zu können. Serielle Software-Ports benötigen sehr viel Hilfe vom Arduino-Controller, um Daten senden und empfangen zu können, weshalb sie nicht so schnell und effizient sind wie serielle Hardware-Ports.

Das Arduino Mega besitzt vier serielle Hardware-Ports, die mit bis zu vier verschiedenen seriellen Geräten kommunizieren können. Nur bei einem ist ein USB-Adapter integriert (alle anderen seriellen Ports können mit einem USB/TTL-Adapter verbunden werden). Tabelle 4-1 zeigt die Portnamen und -Pins aller seriellen Ports des Mega.

Tabelle 4-1: Serielle Ports des Arduino Mega

Portname	Sendepin	Empfangspin
Serial	1 (auch USB)	0 (auch USB)
Serial1	18	19
Serial2	16	17
Serial3	14	15

Serielle Software

Sie werden üblicherweise die in Arduino integrierte Serial-Bibliothek verwenden, um mit den seriellen Hardware-Ports zu kommunizieren. Serielle Bibliotheken vereinfachen die Verwendung serieller Ports, indem sie die Komplexität der Hardware vor Ihnen verbergen.

Manchmal benötigen Sie mehr serielle Ports, als Hardware-Ports zur Verfügung stehen. In diesem Fall können Sie eine zusätzliche Bibliothek nutzen, die serielle Hardware in Software emuliert. Die Rezepte 4.13 und 4.14 zeigen, wie man eine serielle Bibliothek nutzt, um mit mehreren Geräten zu kommunizieren.

Serielles Protokoll

Die Hardware- und Software-Bibliotheken übernehmen das Senden und Empfangen von Informationen. Diese Informationen bestehen häufig aus Gruppen von Variablen, die zusammen gesendet werden müssen. Damit diese Informationen korrekt interpretiert werden können, muss die Empfangsseite erkennen, wo eine Nachricht beginnt und endet. Eine sinnvolle serielle Kommunikation bzw. jede Art der Maschine/Maschine-Kommunikation kann nur erreicht werden, wenn die sendende und die empfangende Seite genau darin übereinstimmen, wie die Informationen in den Nachrichten organisiert sind. Die formale Organisation einer Nachricht und die Menge korrekter Antworten auf Anfragen wird *Kommunikationsprotokoll* genannt.

Nachrichten können ein oder mehr spezielle Zeichen enthalten, die den Anfang einer Nachricht markieren – das bezeichnet man als *Header* (Kopf). Ein oder mehr Zeichen können auch genutzt werden, um das Ende der Nachricht zu kennzeichnen – das bezeichnet man als *Footer* (Fuß). Die Rezepte dieses Kapitels zeigen beispielhafte Nachrichten, bei denen die Werte des Rumpfs (Body, also die eigentlichen Nutzdaten) im Text- oder Binärformat gesendet werden.

Das Senden und Empfangen von Nachrichten im Textformat verlangt das Senden von Befehlen und numerischen Werten in Form von für Menschen lesbaren Buchstaben und Wörtern. Zahlen werden als Strings von Ziffern gesendet, die den Wert repräsentieren. Ist der Wert beispielsweise 1234, dann werden die Zeichen 1, 2, 3 und 4 als einzelne Zeichen gesendet.

Binäre Nachrichten bestehen aus den Bytes, die der Computer zur Repräsentation der Werte verwendet. Binärdaten sind effizienter (weil weniger Bytes gesendet werden müs-

sen), doch die Daten sind für uns Menschen nicht so einfach zu lesen, was die Fehlersuche erschwert. Arduino stellt die Zahl 1234 beispielsweise mit den Bytes 4 und 210 (4 * 256 + 210 = 1234) dar. Wenn das verbundene Gerät nur Binärdaten sendet oder empfängt, bleibt Ihnen keine andere Wahl, als mit diesem Format zu arbeiten, doch wenn Sie die Wahl haben, sind Textnachrichten einfacher zu implementieren und zu debuggen.

Es gibt viele Möglichkeiten, Softwareprobleme anzugehen, und einige Rezepte dieses Kapitels bieten zwei oder drei unterschiedliche Lösungen für das gleiche Ergebnis an. Die Unterschiede (z.B. das Senden von Text anstelle reiner Binärdaten) liegen im Verhältnis von Einfachheit und Effizienz. Wo eine Auswahl angeboten wird, sollten Sie die Lösung wählen, die Sie am besten verstehen und adaptieren können (üblicherweise die erste Lösung). Die Alternativen sind möglicherweise etwas effizienter, oder für ein bestimmtes Protokoll besser geeignet, aber die »richtige Lösung« ist diejenige, die in Ihrem Projekt am einfachsten eingesetzt werden kann.

Die Processing-Entwicklungsumgebung

Einige Beispiele in diesem Kapitel verwenden die Sprache Processing, um serielle Meldungen auf einem Computer zu senden und zu empfangen.

Processing ist ein freies Open-Source-Tool, das eine ähnliche Entwicklungsumgebung nutzt wie Arduino. Statt aber Sketches auf dem Mikrocontroller auszuführen, laufen Processing-Sketches auf Ihrem Computer. Alle Informationen zu Processing und zum Download finden Sie auf der Processing-Website (*http://processing.org/*).

Processing basiert auf Java, doch die Processing-Codebeispiele in diesem Buch sollten sich recht einfach in anderen Umgebungen nutzen lassen, die die serielle Kommunikation unterstützen. Processing wird mit einigen Beispiel-Sketches ausgeliefert, die die Kommunikation zwischen Arduino und Processing illustrieren. SimpleRead ist ein Arduino-Code enthaltendes Processing-Beispiel. In Processing wählen Sie File→Examples→Libraries→Serial→SimpleRead. Das Beispiel liest Daten über den seriellen Port ein und ändert die Farbe eines Rechtecks, wenn ein am Arduino angeschlossener Taster gedrückt oder losgelassen wird.

Neues in Arduino 1.0

Arduino 1.0 führt eine Reihe von Verbesserungen und Änderungen bei der Serial-Bibliothek ein:

- `Serial.flush` wartet nun, bis alle ausgehenden Daten gesendet wurden, statt empfangene Daten einfach auszusortieren. Mit der folgenden Anweisung können Sie alle Daten aus dem Empfangspuffer löschen: `while(Serial.read() >= 0) ; // Empfangspuffer leeren`
- `Serial.write` und `Serial.print` »blockieren« nicht. Der alte Code hat gewartet, bis alle Zeichen gesendet waren, bevor er zurückkehrte. Seit 1.0 werden von `Serial.write` gesendete Daten im Hintergrund übertragen (über einen Interrupthandler), d.h., der

Sketch kann seine Arbeit direkt wieder aufnehmen. Üblicherweise ist das eine gute Sache (der Sketch reagiert schneller), doch manchmal muss man warten, bis alle Zeichen gesendet wurden. Sie erreichen das, indem Sie `Serial.flush()` gleich nach `Serial.write()` aufrufen.

- Die print-Funktionen von Serial geben die Anzahl der ausgegebenen Zeichen zurück. Das ist nützlich, wenn die Textausgaben ausgerichtet werden müssen, oder wenn die übertragenen Daten die Gesamtzahl der gesendeten Zeichen enthalten.
- Ein Parsing ist für Streams wie Serial fest integriert, um Zahlen extrahieren und Text aufspüren zu können. Mehr zu diesen Möglichkeiten bei Serial zeigt Rezept 4.5.
- Die bei Arduino mitgelieferte SoftwareSerial-Bibliothek wurde stark verbessert. Siehe 4.13 und 4.14.
- Die Funktion `Serial.peek` wurde hinzugefügt, mit der Sie sich das nächste Zeichen im Empfangspuffer ansehen können. Im Gegensatz zu `Serial.read` wird das Zeichen mit `Serial.peek` nicht aus dem Puffer entfernt.

Siehe auch

Eine Arduino-Einführung zu RS-232 finden Sie unter *http://www.arduino.cc/en/Tutorial/ArduinoSoftwareRS232*. Sehr viele Informationen und Links sind auch auf der Serial Port Central-Website *http://www.lvr.com/serport.htm* zu finden.

Darüber hinaus gibt eine Reihe von Büchern zu Processing:

- *Processing* (ISBN 978-3-89721-997-7) von Erik Bartmann, erschienen bei O'Reilly.
- *Getting Started with Processing: A Quick, Hands-on Tutorial* von Casey Reas und Ben Fry (Make).
- *Processing: A Programming Handbook for Visual Designers and Artists* von Casey Reas und Ben Fry (MIT Press).
- *Visualizing Data* von Ben Fry (O'Reilly; suchen Sie bei oreilly.de) danach.
- *Processing: Creative Coding and Computational Art* von Ira Greenberg (Apress).
- *Making Things Talk* (ISBN 978-3-86899-162-8) von Tom Igoe (Make). Dieses Buch behandelt Processing und Arduino und enthält viele Beispiele für Kommunkationscode. Bei O'Reilly erschienen.

4.1 Debugging-Informationen vom Arduino an Ihren Computer senden

Problem

Sie wollen Texte und Daten senden, die auf Ihrem PC oder Mac in der Arduino-IDE oder einem Terminalprogramm Ihrer Wahl ausgegeben werden sollen.

Lösung

Dieser Sketch gibt eine Folge von Zahlen über den seriellen Monitor aus:

```
/*
 * SerialOutput Sketch
 * Gibt Zahlen am seriellen Port aus
*/
void setup()
{
 Serial.begin(9600); // Senden und Empfangen mit 9600 Baud
}

int number = 0;

void loop()
{
 Serial.print("Die Zahl ist ");
 Serial.println(number);   // Zahl ausgeben

 delay(500); // Halbe Sekunde warten
 number++; // Nächste Zahl
}
```

Verbinden Sie den Arduino wie in Kapitel 1 beschrieben mit dem Computer und laden Sie den Sketch hoch. Klicken Sie das Icon für den seriellen Monitor in der IDE an, und die folgende Ausgabe sollte erscheinen:

```
Die Zahl ist 0
Die Zahl ist 1
Die Zahl ist 2
```

Diskussion

Um Texte oder Zahlen von Ihrem Sketch auf einem PC oder Mac über den seriellen Link auszugeben, fügen Sie die Anweisung `Serial.begin(9600)` in `setup()` ein und verwenden dann `Serial.print()`-Anweisungen, um die gewünschten Texte oder Werte auszugeben.

Der serielle Monitor kann vom Arduino gesendete serielle Daten ausgeben. Um den seriellen Monitor zu starten, klicken Sie das Icon in der Werkzeugleiste an (siehe Abbildung 4-2). Ein neues Fenster wird geöffnet, das die Ausgaben des Arduino enthält.

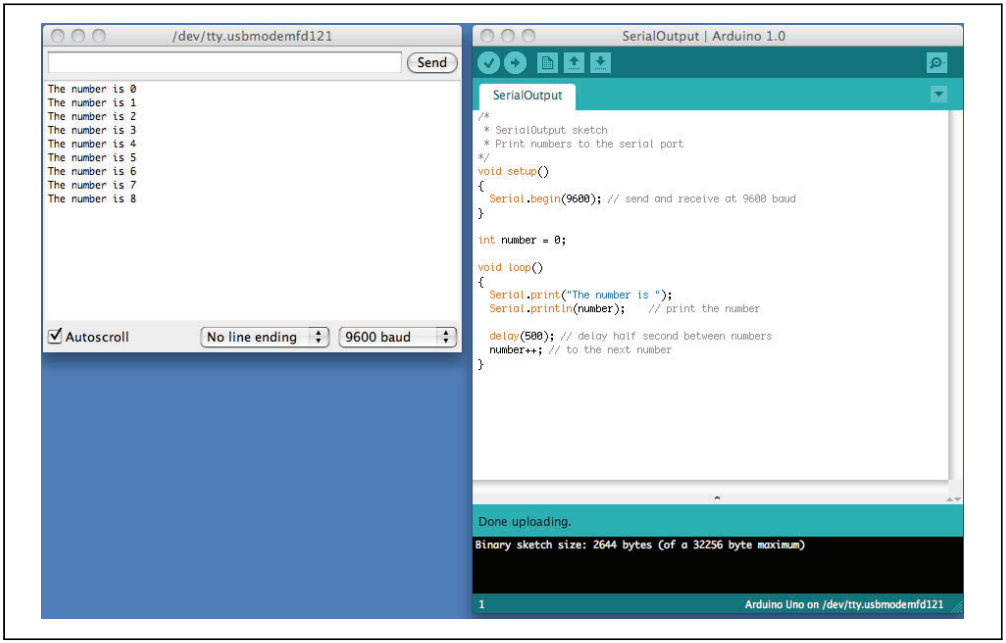


Abbildung 4-2: Serieller Monitor des Arduino

Ihr Sketch muss `Serial.begin()` aufrufen, bevor er die serielle Ein- und Ausgabe nutzen kann. Die Funktion verlangt einen einzelnen Parameter: die gewünschte Kommunikationsgeschwindigkeit. Sie müssen auf Sende- und Empfangsseite die gleiche Geschwindigkeit einstellen, sonst erscheint auf dem Bildschirm nur Zeichensalat (oder gar nichts). Diese Beispiele (und die meisten anderen in diesem Buch) verwenden eine Geschwindigkeit von 9600 Baud (*Baud* ist das Maß für die Zahl der pro Sekunde übertragenen Bits). Eine Baudrate von 9600 entspricht ungefähr 1000 Zeichen pro Sekunde. Sie können kleinere und höhere Geschwindigkeiten (von 300 bis 115200) einstellen, müssen aber sicherstellen, das auf beiden Seiten die gleiche Geschwindigkeit verwendet wird. Der serielle Monitor legt die Geschwindigkeit über die Baudraten-Dropdown-Box (am unteren rechten Rand des Seriellen-Monitor-Fensters in Abbildung 4-2) fest. Wenn Ihre Ausgabe eher so aussieht:

```
`3??f<ÌxÌ□□□ü`³??f<
```

sollten Sie überprüfen, ob die im seriellen Monitor gewählte Baudrate der Baudrate entspricht, die Sie im Sketch bei `Serial.begin()` angegeben haben.

Wenn Sende- und Empfangsgeschwindigkeit übereinstimmen und trotzdem unleserlicher Text erscheint, überprüfen Sie, ob das korrekte Board im Menü Tools→Board ausgewählt wurde. Bei einigen Boards gibt es Unterschiede bei den Chip-Geschwindigkeiten, und wenn Sie den falschen gewählt haben, müssen Sie das korrigieren und das Programm noch einmal hochladen.

Sie können Text mit der Funktion `Serial.print()`ausgeben. Strings (zwischen Anführungszeichen stehender Text) wird unverändert (aber ohne die Anführungszeichen) ausgegeben. Der folgende Code:

```
Serial.print("Die Zahl ist ");
```

gibt also Folgendes aus:

```
Die Zahl ist
```

Die ausgegebenen Werte (Zahlen) hängen vom Variablentyp ab. Mehr zu diesem Thema finden Sie in Rezept 4.2. Für ein Integer wird zum Beispiel der numerische Wert ausgegeben. Ist die Variable `number` auf `1` gesetzt, dann gibt der Code:

```
Serial.println(number);
```

Folgendes aus:

```
1
```

Im Beispiel-Sketch wird beim Start der Schleife zuerst der Wert `0` ausgegeben und dann bei jedem Schleifendurchlauf erhöht. Das `ln` am Ende von `println` sorgt dafür, dass die nächste Ausgabe in der nächsten Zeile beginnt.

Sie sind nun soweit, Texte und Integerwerte ausgeben zu können. Details zu Formatoptionen finden Sie in Rezept 4.2.

Sie könnten auch mit einem Terminalprogramm von einem Drittanbieter liebäugeln, das über mehr Features verfügt als der serielle Monitor. Die Darstellung von Daten im Text- oder Binärformat (oder beides), Darstellung von Steuerzeichen und das Logging in eine Datei sind nur einige zusätzliche Fähigkeiten vieler Terminalprogramme. Hier einige Programme, die von Arduino-Benutzern empfohlen wurden:

CoolTerm (http://freeware.the-meiers.org/)
: Ein einfach zu nutzendes Freeware-Terminal-Programm für Windows, Mac und Linux

CuteCom (http://cutecom.sourceforge.net/)
: Ein Open-Source-Terminal-Programm für Linux

Bray Terminal (https://sites.google.com/site/terminalbpp/)
: Ein freies Programm für den PC

GNU screen (http://www.gnu.org/software/screen/)
: Ein Open-Source-Programm zur Verwaltung virtueller Bildschirme, das die serielle Kommunikation unterstützt. Bei Linux und Mac OS X enthalten

moserial (http://live.gnome.org/moserial)
: Ein weiteres Open-Source-Terminal-Programm für Linux

PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/)
: Ein Open-Source- SSH-Programm für Windows und Linux, das die serielle Kommunikation unterstützt

RealTerm (http://realterm.sourceforge.net/)
: Ein Open- Source-Terminal-Programm für den PC

ZTerm (http://homepage.mac.com/dalverson/zterm/)
: Ein Shareware- Programm für den Mac

Ein Artikel im Arduino-Wiki erläutert außerdem, wie man Linux konfiguriert, um mit dem Arduino per TTY zu kommunizieren (siehe *http://www.arduino.cc/playground/Interfacing/LinuxTTY*).

Sie können ein LC-Display für die serielle Ausgabe verwenden, auch wenn die Funktionalität stark eingeschränkt ist. Schauen Sie in der Dokumentation nach, wie das Display Carriage-Returns handhabt, da einige Displays bei `println`-Anweisungen nicht automatisch zur nächsten Zeile springen.

Siehe auch

Die Arduino-Bibliothek LiquidCrystal für Text-LCDs besitzt eine print-Funktionalität, die derjenigen der Serial-Bibliothek ähnelt. Viele der in diesem Kapitel gegebenen Vorschläge können auch mit dieser Bibliothek umgesetzt werden (siehe Kapitel 11).

4.2 Formatierten Text und numerische Daten vom Arduino senden

Problem

Sie wollen vom Arduino serielle Daten senden, die als Text, als Dezimalwert, als Hexadezimalwert oder als Binärwert ausgegeben werden sollen.

Lösung

Sie können Daten in vielen verschiedenen Formaten über den seriellen Port ausgeben. Hier ein Sketch, der alle Formatoptionen vorstellt:

```
/*
 * SerialFormatting
 * Gibt Werte in verschiedenen Formaten über den seriellen Port aus
 */
char chrValue = 65; // Startwert für die Ausgabe
byte byteValue = 65;
int intValue  = 65;
float floatValue = 65.0;

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

void loop()
```

```
{
 Serial.println("chrValue: ");
 Serial.println(chrValue);
 Serial.write(chrValue);
 Serial.println();
 Serial.println(chrValue,DEC);

 Serial.println("byteValue: ");
 Serial.println(byteValue);
 Serial.write(byteValue);
 Serial.println();
 Serial.println(byteValue,DEC);

 Serial.println("intValue: ");
 Serial.println(intValue);
 Serial.println(intValue,DEC);
 Serial.println(intValue,HEX);
 Serial.println(intValue,OCT);
 Serial.println(intValue,BIN);

 Serial.println("floatValue: ");
 Serial.println(floatValue);

 delay(1000); // Eine Sekunde Warten
 chrValue++; // Nächster Wert
 byteValue++;
 intValue++;
 floatValue +=1;
}
```

Die Ausgabe (hier auf wenige Zeilen gekürzt) sieht wie folgt aus:

```
chrValue:  A A 65
byteValue: 65 A 65
intValue:  65 65 41 101 1000001
floatValue: 65.00
chrValue:  B B 66
byteValue: 66 B 66
intValue:  66 66 42 102 1000010
floatValue: 66.00
```

Diskussion

Die Ausgabe eines Textstrings ist einfach: `Serial.print("Hallo, Welt");` sendet den Textstring »Hallo, Welt« an das Gerät am Ende des seriellen Ports. Soll nach jeder Zeile ein Zeilenvorschub (Newline) ausgegeben werden, verwenden Sie `Serial.println()`statt `Serial.print()`.

Die Ausgabe numerischer Werte kann etwas schwieriger sein. Wie Byte- und Integerwerte ausgegeben werden, hängt vom Variablentyp und einem optionalen Formatparameter ab. Die Arduino-Sprache ist recht locker, wenn es um die Übergabe von Werten an unterschiedliche Datentypen geht (mehr zu Datentypen finden Sie in Rezept 2.2). Doch diese Flexibilität kann verwirrend sein, weil die numerischen Werte, selbst wenn sie gleich sind, vom Compiler als verschiedene Typen mit unterschiedlichen Charakteristika betrachtet

werden. Zum Beispiel liefert die Ausgabe eines char, byte und int mit dem gleichen Wert nicht unbedingt die gleiche Ausgabe.

Hier einige Beispiele, die alle Variablen mit gleichen Werten erzeugen:

```
char asciiValue = 'A';  // Das ASCII 'A' hat den Wert 65
char chrValue   = 65;   // 8-Bit-Zeichen mit Vorzeichen, ebenfalls das ASCII 'A'
byte byteValue  = 65;   // 8-Bit-Zeichen ohne Vorzeichen, ebenfalls das ASCII 'A'
int intValue    = 65;   // 16-Bit-Integer mit Vorzeichen mit dem Wert 65
float floatValue = 65.0; // Fließkommazahl mit dem Wert 65
```

Tabelle 4-2 zeigt das Ergebnis der Variablenausgabe mit Arduino-Routinen.

Tabelle 4-2: Ausgabeformate bei Serial.print

Datentyp	print (val)	print (val,DEC)	write (val)	print (val,HEX)	print (val,OCT)	print (val,BIN)
char	A	65	A	41	101	1000001
byte	65	65	A	41	101	1000001
int	65	65	A	41	101	1000001
long	long-Format entspricht dem int-Format					
float	65.00	Wird für Fließkommazahlen nicht unterstützt				
double	65.00	double ist mit float identisch				

Der Ausdruck Serial.print(val,BYTE); wird bei Arduino 1.0 nicht länger unterstützt.

Wenn Ihr Code erwartet, dass sich Byte-Variablen wie char-Variablen verhalten (d.h., dass sie als ASCII ausgegeben werden), müssen Sie Serial.write(val); verwenden.

Der Sketch des Rezepts verwendet im Quelltext eine separate Zeile für jede print-Anweisung. Das macht komplexe print-Anweisungen etwas sperrig. Um beispielsweise die Zeile

```
Bei 5 Sekunden: Geschwindigkeit = 17, Strecke = 120
```

auszugeben, würden Sie typischerweise den folgenden Code verwenden:

```
Serial.print("Bei ");
Serial.print(t);
Serial.print(" Sekunden: Geschwindigkeit= ");
Serial.print(s);
Serial.print(", Strecke= ");
Serial.println(d);
```

Viel Code für eine einzige Ausgabezeile. Sie könnten ihn wie folgt zusammenfassen:

```
Serial.print("Bei "); Serial.print(t); Serial.print(" Sekunden, Geschwindigkeit= ");
Serial.print(s); Serial.print(", Strecke= ");Serial.println(d);
```

Oder Sie können die *insertion-style-Fähigkeit* des Arduino-Compilers nutzen, um Ihre print-Anweisungen zu formatieren. Sie können die Vorteile einiger fortgeschrittener C++-Fähigkeiten (streaming insertion-Syntax und Templates) nutzen, wenn Sie ein

Streaming-Template in Ihrem Sketch verwenden. Sie erreichen das am einfachsten, indem Sie die Streaming-Bibliothek einbinden, die von Mikal Hart entwickelt wurde. Auf Mikals website (*http://arduiniana.org/libraries/streaming/*) erfahren Sie mehr über diese Bibliothek und zum Download.

Wenn Sie die Streaming-Bibliothek nutzen, liefert die folgende Zeile das gleiche Ergebnis wie der obige Code:

```
Serial << "Bei " << t << " Sekunden, Geschwindigkeit= " << s << ", Strecke = " << d << endl;
```

Siehe auch

Kapitel 2 enthält Informationen zu den von Arduino verwendeten Datentypen. Die Arduino-Web-Referenz unter *http://arduino.cc/en/Reference/HomePage* behandelt die seriellen Befehle und die Arduino-Web-Referenz unter *http://www.arduino.cc/playground/Main/StreamingOutput* behandelt das Streaming (insertion-Style).

4.3 Serielle Daten mit Arduino empfangen

Problem

Sie wollen mit dem Arduino serielle Daten von einem Computer oder einem anderen seriellen Gerät empfangen, damit er z.B. auf Befehle oder Daten reagiert, die von Ihrem Computer gesendet werden.

Lösung

Der Empfang von 8-Bit-Werten (Zeichen und Bytes) ist einfach, weil die `Serial`-Funktionen mit 8-Bit-Werten arbeiten. Der folgende Sketch empfängt eine Ziffer (ein einzelnes Zeichen zwischen 0 und 9) und lässt die LED an Pin 13 mit einer Rate proportional zur empfangenen Ziffer blinken:

```
/*
 * SerialReceive Sketch
 * LED mit einer Rate proportional zur empfangenen Ziffer blinken lassen
*/
const int ledPin = 13; // Mit Pin 13 verbundene LED
int  blinkRate=0;   // Blinkrate steht in dieser Variable

void setup()
{
 Serial.begin(9600); // Serieller Port sendet und empfängt mit 9600 Baud
 pinMode(ledPin, OUTPUT); // Diesen Pin als Ausgang verwenden
}

void loop()
{
 if ( Serial.available()) // Prüfen, ob mindestens ein Zeichen vorhanden ist
 {
  char ch = Serial.read();
```

```
  if( isDigit(ch) ) // ASCII-Zeichen zwischen 0 und 9?
  {
    blinkRate = (ch - '0');    // ASCII-Wert in numerischen Wert umwandeln
    blinkRate = blinkRate * 100; // Rate ist 100ms mal empfangene Ziffer
  }
 }
 blink();
}

// LED mit ermittelter blinkRate ein- und ausschalten
void blink()
{
 digitalWrite(ledPin,HIGH);
 delay(blinkRate); // Wartezeit abhängig von blinkRate-Wert
 digitalWrite(ledPin,LOW);
 delay(blinkRate);
}
```

Laden Sie den Sketch hoch und senden Sie Nachrichten über den seriellen Monitor. Öffnen Sie den seriellen Monitor durch Anklicken des Monitor-Icons (siehe Rezept 4.1) und geben Sie eine Ziffer im Textfeld des seriellen Monitors ein. Sobald Sie den Send-Button anklicken, wird das im Textfeld eingegebene Zeichen gesendet und Sie sehen, wie sich die Blinkgeschwindigkeit ändert.

Diskussion

Die Umwandlung der empfangenen ASCII-Zeichen in numerische Werte ist nicht gleich ersichtlich, wenn man nicht damit vertraut ist, wie Zeichen bei ASCII repräsentiert werden. Die folgende Zeile wandelt das Zeichen `ch` in seinen numerischen Wert um:

```
blinkRate = (ch - '0');  // ASCII-Wert in numerischen Wert umwandeln
```

Den ASCII-Zeichen ’0’ bis ’9’ sind die Werte 48 bis 57 zugeordnet (siehe Anhang G – steht als Download bereit). Die Umwandlung der ’1’ in den numerischen Wert erfolgt durch Subtraktion von ’0’, weil ’1’ den ASCII-Wert 49 hat, d.h., 48 (ASCII ’0’) muss abgezogen werden, um diese Ziffer in die entsprechende Zahl umzuwandeln. Wenn `ch` das Zeichen '1' enthält, ist der ASCII-Wert 49. Der Ausdruck `49- '0'` entspricht `49-48`. Das ergibt wiederum 1, was dem numerischen Wert des Zeichens '1' entspricht.

Mit anderen Worten, der Ausdruck (`ch - '0'`) ist mit dem Ausdruck (`ch - 48`) identisch und wandelt des ASCII-Wert der Variablen `ch` in den entsprechenden numerischen Wert um.

Der Empfang von Zahlen mit mehr als einer Ziffer verlangt die Akkumulation der Zeichen, bis ein Zeichen erkannt wird, das keine Ziffer ist. Der folgende Code verwendet die gleichen `setup()`- and `blink()`-Funktionen wie oben, liest aber Ziffern ein, bis ein Newline-Zeichen empfangen wird. Es verwendet den akkumulierten Wert, um die Blinkgeschwindigkeit festzulegen.

Das Newline-Zeichen (ASCII-Wert 10) kann bei jedem Klick auf Send automatisch angehangen werden. Der serielle Monitor besitzt am unteren Rand des Fensters eine entsprechende Dropdown-Box (siehe Abbildung 4-1). Ändern Sie die Option von »No line ending« in »Newline«.

Ändern Sie den Code wie folgt:

```
int value;

void loop()
{
 if( Serial.available())
 {
  char ch = Serial.read();
  if( isDigit(ch) )// ASCII-Zeichen zwischen 0 bis 9?
  {
    value = (value * 10) + (ch - '0'); // Ja, Wert akkumulieren
  }
  else if (ch == 10)  // Newline-Zeichen?
  {
    blinkRate = value;  // blinkRate auf akkumulierten Wert setzen
    Serial.println(blinkRate);
    value = 0; // Wert für die nächste Ziffernfolge auf 0 zurücksetzen
  }
 }
 blink();
}
```

Geben Sie einen Wert wie 123 in das Monitor-Textfeld ein und klicken Sie auf Send. Die Blinkgeschwindigkeit wird auf 123 Millisekunden gesetzt. Jede Ziffer wird von ihrem ASCII-Wert in ihren numerischen Wert umgewandelt. Da es sich bei den Zahlen um Dezimalzahlen handelt (Basis 10), wird der akkumulierte Wert mit 10 multipliziert. Zum Beispiel setzt sich der Wert der Zahl 234 aus 2 * 100 + 3 * 10 + 4 zusammen. Das wird mit dem folgenden Code erreicht:

```
  if( isDigit(ch) ) // ASCII-Zeichen zwischen 0 und 9?
  {
    value = (value * 10) + (ch - '0'); // Ja, Wert akkumulieren
  }
```

Wenn Sie negative Zahlen verarbeiten wollen, muss Ihr Code ein führendes Minuszeichen (`'-'`) erkennen können. Im folgenden Beispiel muss jeder numerische Wert durch ein Zeichen getrennt werden, das keine Ziffer und kein Minuszeichen ist:

```
int value = 0;
int sign = 1;

void loop()
{
 if( Serial.available())
 {
  char ch = Serial.read();
  if( isDigit(ch) ) // ASCII-Zeichen zwischen 0 und 9?
    value = (value * 10) + (ch - '0'); // Ja, Wert akkumulieren
```

```
  else if( ch == '-')
    sign = -1;
  else // Wert komplett, wenn keine Ziffer und kein Minuszeichen
  {
    value = value * sign ;  // Vorzeichen berücksichtigen
    Serial.println(value);
    value = 0; // Wert für die nächste Ziffernfolge auf 0 zurücksetzen
    sign = 1;
  }
 }
}
```

Eine weitere Möglichkeit zur Umwandlung von Strings in Zahlen bieten die C-Konvertierungsfunktionen `atoi` (für `int`-Variablen) oder `atol` (für `long`-Variablen). Diese seltsam klingenden Funktionen wandeln einen String in Integer- oder long-Integerwerte um. Um sie verwenden zu können, müssen Sie zuerst den gesamten String empfangen und in einem Zeichen-Array abspeichern, bevor Sie die Konvertierungsfunktion aufrufen dürfen.

Das folgende Code-Fragment beendet das Einlesen der Ziffern bei jedem Zeichen, das keine Ziffer ist (oder bei vollem Puffer):

```
const int MaxChars = 5; // Ein int-String besteht aus bis zu 5 Ziffern und wird
             // mit einer 0 abgeschlossen, die das Ende des Strings anzeigt
char strValue[MaxChars+1]; // Muss groß genug für die Ziffern und die abschließende Null sein
int index = 0;      // Array-Index zum Speichern der empfangenen Ziffern

void loop()
{
 if( Serial.available())
 {
  char ch = Serial.read();
  if( index < MaxChars && isDigit(ch) ){
   strValue[index++] = ch; // ASCII-Zeichen zum String hinzufügen;
  }
  else
  {
   // Puffer voll oder erste Nicht-Ziffer
   strValue[index] = 0;      // String mit einer 0 abschließen
   blinkRate = atoi(strValue);  // String mit atoi in int-Wert umwandeln
   index = 0;
  }
 }
 blink();
 }
```

`strValue` enthält den numerischen String, der aus den über den seriellen Port empfangenen Zeichen besteht.

Weitere Informationen zu Zeichenketten finden Sie in Rezept 2.6.

`atoi` (eine Abkürzung für »ASCII-nach-Integer«) ist eine Funktion, die eine Zeichenkette in einen Integerwert umwandelt (`atol` wandelt in long-Integer um).

Mit Arduino 1.0 wurde die Funktion `serialEvent` eingeführt, die Sie zur Verarbeitung eingehender serieller Zeichen nutzen können. Wenn es eine `serialEvent`-Funktion in Ihrem Sketch gibt, wird sie bei jedem Durchlauf innerhalb der `loop`-Funktion einmal aufgerufen. Der folgende Sketch bietet die gleiche Funktionalität wie der erste Sketch, nutzt aber `serialEvent` zur Verarbeitung eingehender Zeichen:

```
/*
 * SerialReceive Sketch
 * LED mit einer Rate proportional zur empfangenen Ziffer blinken lassen
 */
const int ledPin = 13; // Mit Pin 13 verbundene LED
int  blinkRate=0;    // Blinkrate steht in dieser Variable

void setup()
{
 Serial.begin(9600); // Serieller Port sendet und empfängt mit 9600 Baud
 pinMode(ledPin, OUTPUT); // Diesen Pin als Ausgang verwenden
}

void loop()
{
 blink();
}

void serialEvent()
{
 while(Serial.available())
 {
  char ch = Serial.read();
  Serial.write(ch);
  if( isDigit(ch) ) // ASCII-Zeichen zwischen 0 und 9?
  {
   blinkRate = (ch - '0');    // ASCII-Wert in numerischen Wert umwandeln
   blinkRate = blinkRate * 100; // Rate ist 100mS mal empfangener Ziffer
  }
 }
}

// LED mit ermittelter blinkRate ein- und ausschalten
void blink()
{
 digitalWrite(ledPin,HIGH);
 delay(blinkRate); // Wartezeit abhängig von blinkRate-Wert value
 digitalWrite(ledPin,LOW);
 delay(blinkRate);
}
```

Mit Arduino 1.0 wurden außerdem die Methoden `parseInt` und `parseFloat` eingeführt, die das Extrahieren von Zahlenwerten aus Serial vereinfachen. (Das funktioniert auch bei Ethernet und anderen Objekten, die aus der `Stream`-Klasse abgeleitet wurden. Weitere Informationen zum Stream-Parsing mit Netzwerkobjekten finden Sie in der Einführung zu Kapitel 15).

`Serial.parseInt()` und `Serial.parseFloat()` lesen Zeichen über Serial ein und liefern deren numerische Werte zurück. Nicht-numerische Zeichen vor der Zahl werden ignoriert und die Konvertierung endet mit dem ersten nicht-numerischen Zeichen (oder '.' bei `parseFloat`.)

In der Diskussion zu Rezept 4.5 finden Sie ein Beispiel dafür, wie `parseInt` zum Aufspüren und Extrahieren von Zahlen aus seriellen Daten genutzt wird.

Siehe auch

Eine Websuche nach »atoi« oder »atol« liefert viele Referenzen für diese Funktionen zurück. Beachten Sie auch den Wikipedia-Eintrag unter *http://en.wikipedia.org/wiki/Atoi*.

4.4 Mehrere Textfelder vom Arduino in einer einzelnen Nachricht senden

Problem

Sie wollen eine Nachricht senden, die mehr als eine Information (ein Feld) enthält. Zum Beispiel könnte die Nachricht Werte von zwei oder mehr Sensoren enthalten. Sie wollen diese Werte in einem Programm wie Processing nutzen, das auf Ihrem PC oder Mac läuft.

Lösung

Die einfachste Lösung besteht darin, einen Textstring zu senden, der alle Felder enthält und sie durch Trennzeichen, beispielsweise durch ein Komma, voneinander abgrenzt:

```
// CommaDelimitedOutput Sketch

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

void loop()
{
 int value1 = 10;  // Einige fest kodierte Werte, die wir senden wollen
 int value2 = 100;
 int value3 = 1000;

 Serial.print('H'); // Eindeutiger Kopf (Header), um den Anfang der Nachricht identifizieren zu
                    // können
 Serial.print(",");
 Serial.print(value1,DEC);
 Serial.print(",");
 Serial.print(value2,DEC);
 Serial.print(",");
 Serial.print(value3,DEC);
 Serial.print(","); // Beachten Sie, dass ein Komma nach dem letzten Feld gesendet wird
```

```
 Serial.println(); // CR/LF senden
 delay(100);
}
```

Hier ein Processing-Sketch, der diese Daten über den seriellen Port einliest:

```
// Processing-Sketch zum Einlesen kommaseparierter Daten
// über den seriellen Port.
// Das erwartete Format ist: H,1,2,3,

import processing.serial.*;

Serial myPort;     // Objekt der Serial-Klasse
char HEADER = 'H';  // Zeichen zur Identifikation des Anfangs einer Nachricht
short LF = 10;     // ASCII-Linefeed

// WARNUNG!
// Falls nötig, in der nachfolgenden Definition den korrekten Port eintragen
short portIndex = 1;  // com-Port wählen, 0 ist der erste Port

void setup() {
 size(200, 200);
 println(Serial.list());
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this,Serial.list()[portIndex], 9600);
}

void draw() {
}

void serialEvent(Serial p)
{
 String message = myPort.readStringUntil(LF); // Serielle Daten einlesen

 if(message != null)
 {
  print(message);
  String [] data = message.split(","); // Kommaseparierte Nachricht zerlegen
  if(data[0].charAt(0) == HEADER && data.length > 3) // check validity
  {
   for( int i = 1; i < data.length-1; i++) // Kopf und Zeilenende überspringen
   {
    println("Wert " + i + " = " + data[i]); // Felder ausgeben
   }
   println();
  }
 }
}
```

Diskussion

Der Arduino-Code dieser Lösung sendet den folgenden Textstring an den seriellen Port (\r steht für das Carriage Return und \nn)^Linefeed (\n)",4>n (Linefeed)^\n (Linefeed)",4> für das Linefeed):

```
H,10,100,1000,\r\n
```

Sie müssen ein Trennzeichen wählen, das in den eigentlichen Daten niemals vorkommt. Wenn die Daten nur aus numerischen Werten bestehen, ist das Komma als Trennzeichen eine gute Wahl. Sie müssen außerdem sicherstellen, dass der Empfänger den Anfang der Nachricht erkennen kann, damit auch wirklich die Daten aller Felder eingelesen werden. Sie erreichen dies, indem Sie ein Header-Zeichen senden, das den Beginn der Nachricht kennzeichnet. Das Header-Zeichen muss ebenfalls eindeutig sein, d.h., es sollte nicht in den Datenfeldern vorkommen und sich auch vom Trennzeichen unterscheiden. Unser Beispiel verwendet das große *H*, um den Anfang der Nachricht anzuzeigen. Die Nachricht besteht aus dem Header, drei kommaseparierten numerischen Werten in Form von ASCII-Strings sowie einem Carriage Return und einem Linefeed.

Die Carriage Return- und Linefeed-Zeichen werden immer dann gesendet, wenn Arduino etwas über die Funktion `println()`ausgibt. Das hilft der Empfangsseite zu erkennen, wann der Nachrichten-String vollständig empfangen wurde. Ein Komma wird auch nach dem letzten numerischen Wert gesendet, um der Empfangsseite dabei zu helfen, das Ende der Werte zu erkennen.

Der Processing-Code liest die Nachricht als String ein und nutzt die Java-Methode `split()`, um ein Array kommaseparierter Felder zu erzeugen.

In den meisten Fällen werden Sie bei einem Mac den ersten seriellen Port nutzen wollen, während Sie beim PC den letzten nutzen. Der Processing-Sketch enthält Code, der die verfügbaren und den gerade ausgewählten Port anzeigt. Stellen Sie sicher, dass das auch der Port ist, an dem der Arduino hängt.

Die Verwendung von Processing zur Darstellung von Sensordaten kann einem viele Stunden beim Debugging ersparen, da es Ihnen hilft, die Sensordaten zu visualisieren. Der folgende Processing-Sketch visualisiert bis zu 12 vom Arduino gesendete Werte in Echtzeit. Diese Version stellt 8-Bit-Werte im Bereich von –127 bis +127 dar und wurde als Demonstration für den Nunchuck-Sketch in Rezept 13.2 entwickelt:

```
/*
 * ShowSensorData.
 *
 * Erzeugt ein Balkendiagramm aus CSV-Sensordaten im Bereich von -127 bis 127
 * Das erwartete Format ist: "Data,s1,s2,...s12\n" (unterstützt bis zu 12 Sensoren)
 * Label können wie folgt gesendet werden: "Labels,label1, label2,...label12\n");
 */

import processing.serial.*;

Serial myPort; // Objekt der Serial-Klasse erzeugen
String message = null;
PFont fontA;  // Font zur Darstellung der Servo-Pin-Nummer
int fontSize = 12;

int maxNumberOfLabels = 12;

int rectMargin = 40;
int windowWidth = 600;
```

```
int windowHeight = rectMargin + (maxNumberOfLabels + 1) * (fontSize *2);
int rectWidth = windowWidth - rectMargin*2;
int rectHeight = windowHeight - rectMargin;
int rectCenter = rectMargin + rectWidth / 2;

int origin = rectCenter;
int minValue = -127;
int maxValue = 127;

float scale = float(rectWidth) / (maxValue - minValue);

String [] sensorLabels = {"s1", "s2", "s3", "s4", "s5", "s6", "s7", "s8", "s9",
                          "s10", "s11", "s12"};
// Wird auf die Anzahl tatsächlich empfangener Label geändert
int labelCount = maxNumberOfLabels;

void setup() {
 size(windowWidth, windowHeight);
 short portIndex = 1;  // com-Port wählen, 0 ist der erste Port
 String portName = Serial.list()[portIndex];
 println(Serial.list());
 println(" Verbinde mit -> " + portName) ;
 myPort = new Serial(this, portName, 57600);
 fontA = createFont("Arial.normal", fontSize);
 textFont(fontA);
 labelCount = sensorLabels.length;
}

void drawGrid() {
 fill(0);
 text(minValue, xPos(minValue), rectMargin-fontSize);
 line(xPos(minValue), rectMargin, xPos(minValue), rectHeight + fontSize);
 text((minValue+maxValue)/2, rectCenter, rectMargin-fontSize);
 line(rectCenter, rectMargin, rectCenter, rectHeight + fontSize);
 text(maxValue, xPos(maxValue), rectMargin-fontSize);
 line( xPos(maxValue), rectMargin, xPos(maxValue), rectHeight + fontSize);

 for (int i=0; i < labelCount; i++) {
  text(sensorLabels[i], fontSize, yPos(i));
  text(sensorLabels[i], xPos(maxValue) + fontSize, yPos(i));
 }
}

int yPos(int index) {
 return rectMargin + fontSize + (index * fontSize*2);
}

int xPos(int value) {
 return origin  + int(scale * value);
}

void drawBar(int yIndex, int value) {
 rect(origin, yPos(yIndex)-fontSize, value * scale, fontSize);  // Wert zeichnen
}

void draw() {
```

```
while (myPort.available () > 0) {
 try {
  message = myPort.readStringUntil(10);
  if (message != null) {
   print(message);
   String [] data  = message.split(","); // CSV-Nachricht zerlegen
   if ( data[0].equals("Labels") ) { // Auf Label-Header überprüfen
    labelCount = min(data.length-1, maxNumberOfLabels) ;
    arrayCopy(data, 1, sensorLabels, 0, labelCount );
   }
   else if ( data[0].equals("Data"))// Auf Daten-Header prüfen
   {
    background(255);
    drawGrid();
    fill(204);
    println(data.length);
    for ( int i=1; i <= labelCount && i < data.length-1; i++)
    {
      drawBar(i-1, Integer.parseInt(data[i]));
    }
   }
  }
 }
 catch (Exception e) {
  e.printStackTrace(); // Mögliche Fehler ausgeben
 }
}
}
```

Abbildung 4-3 zeigt, wie die Nunchuck-Beschleunigungswerte (`aX`,`Ay`,`aZ`) und die Joystick-Werte (`jX`,`Jy`) dargestellt werden. Die Balken erscheinen, wenn die Nunchuck-Buttons (`bC` und `bZ`) gedrückt werden.

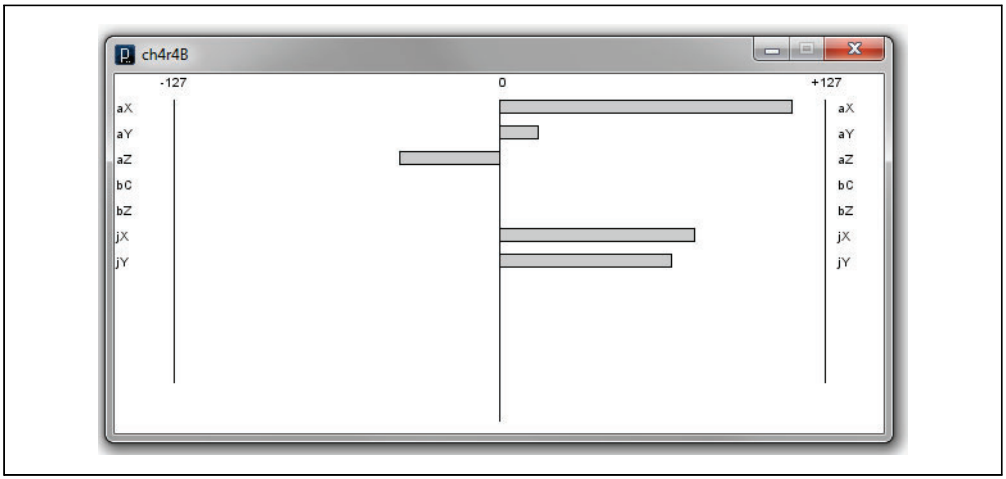


Abbildung 4-3: Processing-Bildschirm mit Nunchuck-Sensordaten

Wertebereich und Ursprung des Diagramms können bei Bedarf einfach angepasst werden. Um zum Beispiel Diagramme darzustellen, die von der Z-Achse ausgehen und einen Wertebereich von 0 bis 1024 umfassen, verwenden Sie Folgendes:

```
int origin = rectMargin; // rectMargin ist die linke Ecke des Grafikbereichs
int minValue = 0;
int maxValue = 1024;
```

Wenn Sie keinen Nunchuck besitzen, können Sie mit dem folgenden einfachen Sketch Analogwerte generieren. Sind keine Sensoren angeschlossen, können Sie mit den Fingern unten an den Analogpins vorbeistreichen. Damit erzeugen Sie Signalpegel, die mit dem Processing-Sketch dargestellt werden können. Die Werte liegen im Bereich von 0 bis 1023, d.h., Sie müssen den Ursprung und den Minimal-/Maximalwert im Processing-Sketch wie oben beschrieben ändern:

```
void setup() {
 Serial.begin(57600);
 delay(1000);
 Serial.println("Labels,A0,A1,A2,A3,A4,A5");
}

void loop() {
 Serial.print("Data,");
 for(int i=0; i < 6; i++)
 {
  Serial.print( analogRead(i) );
  Serial.print(",");
 }
 Serial.print('\n'); // Newline-Zeichen
 delay(100);
}
```

Siehe auch

Die Processing-Website enthält weitere Informationen zur Installation und Verwendung der Programmierumgebung. Siehe *http://processing.org/*.

4.5 Mit dem Arduino mehrere Textfelder in einer Nachricht empfangen

Problem

Sie wollen eine Nachricht empfangen, die mehr als ein Feld enthält. Zum Beispiel könnte eine Nachricht den Bezeichner (Identifier) für ein bestimmtes Bauelement (etwa einen Motor der Aktuator) und den für ihn gedachten Wert (z.B. die Geschwindigkeit) enthalten.

Lösung

Arduino kennt keine split()-Funktion, wie wir sie im Processing-Code in Rezept 4.4 nutzen, aber man kann (wie in diesem Rezept gezeigt) eine vergleichbare Funktionalität implementieren. Der folgende Code empfängt eine Nachricht mit drei numerischen Feldern, die durch Kommata voneinander getrennt sind. Er nutzt die in Rezept 4.4 beschriebene Technik zum Empfang von Ziffern und wurde um Code erweitert, der kommaseparierte Felder erkennt und die Werte in einem Array speichert:

```
/*
 * SerialReceiveMultipleFields Sketch
 * Der Code erwartet eine Nachricht im Format 12,345,678
 * Der Code erwartet ein Newline-Zeichen, der das Ende der Daten anzeigt
 * Richten Sie den seriellen Monitor so ein, dass er Newline-Zeichen sendet
 */

const int NUMBER_OF_FIELDS = 3; // Wie viele kommaseparierte Felder erwarten wir?
int fieldIndex = 0;        // Das aktuell empfangene Feld
int values[NUMBER_OF_FIELDS];  // Array mit den Werten aller Felder



void setup()
{
 Serial.begin(9600); // Serieller Port sendet und empfängt mit 9600 Baud
}

void loop()
{
 if( Serial.available())
 {
  char ch = Serial.read();
  if(ch >= '0' && ch <= '9') // ASCII-Zeichen zwischen 0 und 9?
  {
   // Ja, Wert akkumulieren, solange fieldIndex gültig ist
   // Überzählige Felder werden nicht gespeichert
   if(fieldIndex < NUMBER_OF_FIELDS) {
    values[fieldIndex] = (values[fieldIndex] * 10) + (ch - '0');
   }
  }
  else if (ch == ',')  // Komma ist unser Trennzeichen, also weiter zum nächsten Feld
  {
    fieldIndex++;  // Feldindex inkrementieren
  }
  else
  {
   // Jedes Zeichen außer Ziffern und Komma beendet das Einlesen der Felder.
   // In diesem Beispiel ist dies das vom seriellen Monitor gesendete Newline-Zeichen

   // Alle gespeicheren Felder ausgeben
   for(int i=0; i < min(NUMBER_OF_FIELDS, fieldIndex+1); i++)
   {
    Serial.println(values[i]);
    values[i] = 0; // Werte für nächste Nachricht auf 0 zurücksetzen
   }
```

```
    fieldIndex = 0;  // Index für Neustart vorbereiten
   }
  }
}
```

Diskussion

Der Sketch akkumuliert die Werte (wie in Rezept 4.3beschrieben), hält aber jeden Wert in einem Array fest (das groß genug sein muss, um alle Felder aufnehmen zu können), wenn ein Komma empfangen wird. Jedes Zeichen, das keine Ziffer und kein Komma ist (etwa das Newline-Zeichen, siehe Rezept 4.3) stößt die Ausgabe aller im Array gespeicherten Werte an. Sie können also ein Zeichen eingeben, das keine Ziffer und kein Komma ist, oder das »No line ending«-Menü am unteren rechten Rand im seriellen Monitor auf einen anderen Wert setzen.

Arduino 1.0 hat die Methode `parseInt` eingeführt, die es einem leicht macht, Informationen aus seriellen und Web-Streams zu extrahieren. Hier ein Beispiel für ihren Einsatz (Kapitel 15 enthält weitere Beispiele zum Stream-Parsing).

Der folgende Sketch bietet die gleiche Funktionalität wie oben, nutzt aber `parseInt`:

```
// Mehrere numerische Felder mittels Arduino 1.0-Stream-Parsing verarbeiten

const int NUMBER_OF_FIELDS = 3; // Wie viele kommaseparierte Felder erwarten wir?
int fieldIndex = 0;        // Das aktuell empfangene Feld
int values[NUMBER_OF_FIELDS];  // Array mit den Werte aller Felder


void setup()
{
 Serial.begin(9600); // Serieller Port sendet und empfängt mit 9600 Baud
}

void loop()
{

 if( Serial.available()) {
  for(fieldIndex = 0; fieldIndex  < 3; fieldIndex ++)
  {
   values[fieldIndex] = Serial.parseInt(); // Numerischen Wert einlesen

  }
  Serial.print( fieldIndex);
  Serial.println(" Felder empfangen:");
  for(int i=0; i <  fieldIndex; i++)
  {
    Serial.println(values[i]);
  }
  fieldIndex = 0;  // und von vorn anfangen
 }
}
```

Die Stream-Parsing-Funktionen nutzen ein Timeout, während sie auf ein Zeichen warten. Voreingestellt ist eine Sekunde. Werden innerhalb dieser Zeit keine Zeichen

von `parseInt` empfangen, gibt sie 0 zurück. Sie können den Timeout mit `Stream.setTimeout(timeoutPeriod)` ändern. Der Timeout-Parameter ist ein long-Wert, der den Timeout in Millisekunden angibt. Der Wertebereich für den Timeout reicht also von 1 Millisekunde bis zu 2.147.483.647 Millisekunden.

`Stream.setTimeout(2147483647);` ändert das Timeout-Interval auf etwas unter 25 Tage.

Hier eine Zusammenfassung der von Arduino 1.0 unterstützten Stream-Parsing-Methoden (nicht alle werden im obigen Beispiel verwendet):

`boolean find(char *target);`
: Liest aus dem Stream, bis das angegebene Ziel (»target«) gefunden wurde. Gibt `true` zurück, wenn der Zielstring gefunden wurde. Der Rückgabewert `false` bedeutet, dass die Daten im Stream nicht gefunden wurden und keine weiteren Daten verfügbar sind. Beachten Sie, dass der Stream beim Parsing nur einmal verarbeitet wird, d.h., Sie können nicht zurückgehen, um etwas anderes zu suchen (siehe hierzu `findUntil`).

`boolean findUntil(char *target, char *terminate);`
: Ähnelt der `find`-Methode, aber die Suche endet erst, wenn der Terminierungsstring gefunden wird. Gibt nur dann `true` zurück, wenn das Ziel gefunden wurde. Nützlich, um die Suche bei einem Schlüsselwort oder einem Terminator zu beenden. Zum Beispiel sucht

  ```
  finder.findUntil("target", "\n");
  ```

  nach dem String `"target"`, bricht aber bei einem Newline-Zeichen ab. Der Sketch kann also etwas anderes machen, wenn das Ziel nicht gefunden wird.

`long parseInt();`
: Gibt den ersten gültigen (langen) Integerwert zurück. Führende Zeichen, die keine Ziffern und kein Minuszeichen sind, werden übersprungen. Der Integerwert wird bei der ersten auf die Zahl folgenden Nicht-Ziffer abgeschlossen. Werden keine Ziffern gefunden, gibt die Funktion `0` zurück.

`long parseInt(char skipChar);`
: Wie `parseInt`, aber der angegebene `skipChar` wird innerhalb des numerischen Wertes ignoriert. Kann recht nützlich sein, wenn Sie einen einzelnen numerischen Wert einlesen wollen, der durch Punkte (oder Kommata) getrennt ist (wie das bei großen Zahlen der Fall ist). Denken Sie aber daran, dass mit Kommata formatierte Zahlen nicht in kommaseparierten Strings verarbeitet werden können (32,767 würde dann als 32767 erkannt werden).

`float parseFloat();`
: Die `float`-Version von `parseInt`.

`size_t readBytes(char *buffer, size_t length);`
: Liest die eingehenden Zeichen in den angegebenen Puffer ein, bis es zum Timeout kommt oder die angegebene Anzahl Zeichen eingelesen wurde. Gibt die Anzahl der im Puffer abgelegten Zeichen zurück.

`size_t readBytesUntil(char terminator,char *buf,size_t length);`

Liest die eingehenden Zeichen in den angegebenen Puffer ein, bis das `terminator`-Zeichen erkannt wird. Strings, die die angegebene Länge (`length`) überschreiten, werden abgeschnitten. Die Funktion gibt die Anzahl der im Puffer abgelegten Zeichen zurück.

Siehe auch

Kapitel 15 enthält weitere Stream-Parsing-Beispiele, die Daten in einem Stream finden und extrahieren.

4.6 Binäre Daten vom Arduino senden

Problem

Sie müssen Daten binär übertragen, weil Sie Informationen mit so wenig Daten wie möglich senden wollen, oder weil die Anwendung nur Binärdaten verarbeiten kann.

Lösung

Dieser Sketch sendet einen Header und dann zwei Integerwerte (16 Bit) im Binärformat. Die Werte werden mit der Arduino-Funktion `random` erzeugt (siehe Rezept 3.11):

```
/*
 * SendBinary Sketch
 * Sendet einen Header gefolgt von zwei zufälligen Integerwerten im Binärformat.
*/

int intValue;   // Ein Integerwert (16 Bit)

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

void loop()
{
 Serial.print('H'); // Header-Zeichen senden

 // Zufallszahl senden
 intValue = random(599); // Zufallszahl zwischen 0 und 598 erzeugen
 // Sende die beiden Bytes, aus denen der Integerwert besteht
 Serial.write(lowByte(intValue));  // Niederwertiges Byte senden
 Serial.write(highByte(intValue)); // Höherwertiges Byte senden

 // Weitere Zufallszahl senden
 intValue = random(599); // Zufallszahl zwischen 0 und 598 erzeugen
 // Sende die beiden Bytes, aus denen der Integerwert besteht
 Serial.write(lowByte(intValue));  // Niederwertiges Byte senden
 Serial.write(highByte(intValue)); // Höherwertiges Byte senden

 delay(1000);
}
```

Diskussion

Das Senden binärer Daten verlangt sorgfältige Planung, weil nur Wortsalat herauskommt, solange sich Sender und Empfänger nicht genau darüber verständigt haben, wie die Daten gesendet werden müssen. Im Gegensatz zu Textdaten, bei denen das Textende durch das abschließende Carriage Return (oder ein anderes von Ihnen festgelegtes Zeichen) bestimmt wird, kann man bei Binärdaten möglicherweise nicht sagen, wo eine Nachricht beginnt, wenn man sich die Daten einfach nur ansieht. Wenn Daten jeden Wert enthalten dürfen, können sie auch den Wert eines Header- oder Terminierungszeichen enthalten.

Sie können das verhindern, indem Sie die Nachrichten so entwerfen, dass Sender und Empfänger genau wissen, wie viele Bytes zu erwarten sind. Das Ende der Nachricht wird durch die Zahl der gesendeten Bytes bestimmt, nicht durch die Erkennung eines bestimmten Zeichens. Sie können das implementieren, indem Sie einen Startwert senden, der angibt, wie viele Bytes folgen. Oder Sie können die Größe der Nachricht so festlegen, dass Sie groß genug ist, um die Daten aufnehmen zu können, die Sie senden wollen. Beides ist nicht immer leicht, weil unterschiedliche Plattformen und Sprachen verschiedene Größen für Ihre binären Datentypen verwenden können – sowohl die Anzahl der Bytes als auch ihre Reihenfolge können sich vom Arduino unterscheiden. Zum Beispiel definiert Arduino ein `int` als zwei Bytes, während Processing (Java) ein `int` als vier Bytes definiert (während `short` der Java-Typ für ein 16-Bit-Integer ist). Das Senden eines `int`-Werts als Text (wie in den früheren Text-Rezepten) vereinfacht die Sache, weil jede Zahl als Folge von Ziffern gesendet wird. Die Gegenseite erkennt das Ende des Empfangs über ein Carriage Return oder ein andere Nicht-Ziffer. Binärübertragungen wissen nur etwas über den Aufbau der Nachricht, wenn er im Vorfeld definiert wurde oder in der Nachricht spezifiziert wird.

Die Lösung verlangt die Kenntnis der Datentypen der sendenden und empfangenden Plattform und sorgfältige Planung. Rezept 4.7 zeigt ein Beispiel, bei dem Processing diese Nachrichten empfängt.

Das Senden einzelner Bytes ist einfach. Verwenden Sie `Serial.write(byteVal)`. Um einen Integerwert vom Arduino zu senden, müssen Sie das nieder- und das höherwertige Byte übertragen, aus denen der Integerwert besteht (Rezept 2.2 enthält weiterführende Informationen zu Datentypen). Das geschieht mit Hilfe der Funktionen `lowByte` und `highByte` (siehe Rezept 3.14):

```
Serial.write(lowByte(intValue), BYTE);
Serial.write(highByte(intValue), BYTE);
```

Bei long-Werten brechen Sie die vier Bytes, aus denen ein `long` besteht, in zwei Schritten auf. Der `long`-Wert wird zuerst in zwei 16-Bit-Werte zerlegt, die dann jeweils mit den eben beschriebenen Methoden zum Senden von Integerwerten übertragen werden:

```
int longValue = 1000;
int intValue;
```

Zuerst senden Sie den niederwertigen 16-Bit-Wert:

```
intValue = longValue & 0xFFFF; // Wert der niederwertigen 16 Bit ermitteln
Serial.write(lowByte(intVal));
Serial.write(highByte(intVal));
```

Dann senden Sie den höherwertigen 16-Bit-Wert:

```
intValue = longValue >> 16; // Wert der höherwertigen 16 Bit ermitteln
Serial.write(lowByte(intVal));
Serial.write(highByte(intVal));
```

Sie könnten es bequemer finden, zum Senden der Daten eine Funktion anzulegen. Die folgende Funktion nutzt den oben vorgestellte Code, um einen 16-Bit-Integerwert über den seriellen Port auszugeben:

```
// Sendet den angegebene Integerwert über den seriellen Port
void sendBinary(int value)
{
 // Sende die beiden Bytes, aus denen ein 16-Bit-Integer besteht
 Serial.write(lowByte(value)); // Niederwertiges Byte senden
 Serial.write(highByte(value)); // Höherwertiges Byte senden
}
```

Die folgende Funktion sendet einen `long`-Wert (4 Byte), indem sie zuerst die beiden niederwertigen (rechts stehenden) Bytes und dann die beiden höherwertigen (links stehenden) Bytes überträgt:

```
// Funktion zum Senden eines long-Werts über den seriellen Port
void sendBinary(long value)
{
 // Zuerst wird der niederwertige 16-Bit-Wert gesendet
 int temp = value & 0xFFFF; // Wert der niederwertigen 16 Bits ermitteln
 sendBinary(temp);
 // Dann wird der höherwertige 16-Bit-Wert gesendet
 temp = value >> 16; // Wert der höherwertigen 16 Bit ermitteln
 sendBinary(temp);
}
```

Diese Funktionen zum Senden binärer `int`- und `long`-Werte haben den gleichen Namen: `sendBinary`. Der Compiler unterscheidet sie anhand des Typs, der für den Parameter verwendet wird. Ruft Ihr Code `sendBinary` mit einem 2-Byte-Wert auf, wird die als `void sendBinary(int value)` deklarierte Version aufgerufen. Ist der Parameter ein `long`-Wert, wird die als `void sendBinary(long value)` deklarierte Version genutzt. Dieses Verhalten wird *Funktionsüberladung* genannt. Rezept 4.2 zeigt hierfür ein weiteres Beispiel. Die unterschiedliche Funktionalität von `Serial.print` wird erreicht, indem der Compiler die verschiedenen Variablentypen unterscheidet.

Sie können Binärdaten auch mit Hilfe von *Strukturen* senden. Strukturen sind ein Mechanismus zur Organisation von Daten. Wenn Sie mit ihrem Einsatz nicht vertraut sind, sollten Sie besser bei der eben beschriebenen Lösung bleiben. Für diejenigen, die mit dem Konzept von Zeigern auf Strukturen vertraut sind, zeigt das folgende Beispiel eine Funktion, die die Bytes innerhalb einer Struktur als Binärdaten an den seriellen Port sendet:

```
void sendStructure( char *structurePointer, int structureLength)
{
 int i;

 for (i = 0 ; i < structureLength ; i++)
  serial.write(structurePointer[i]);
 }

sendStructure((char *)&myStruct, sizeof(myStruct));
```

Daten in Form binärer Bytes zu senden, ist effizienter als das Senden der Daten in Textform, funktioniert aber nur dann zuverlässig, wenn sich Sender und Empfänger im Bezug auf den Aufbau der Daten einig sind. Hier eine Übersicht der Dinge, auf die Sie beim Schreiben Ihres Codes achten müssen:

Variablengröße
: Stellen Sie sicher, dass die Größe der gesendeten Daten auf beiden Seiten gleich ist. Ein Integerwert ist bei Arduino 2 Byte groß, auf den meisten anderen Plattformen aber 4 Byte. Prüfen Sie in der Dokumentation immer die Größe des Datentyps, damit sie übereinstimmen. Es ist für Processing kein Problem, ein 2-Byte-Arduino-Integer als 4-Byte-Integer in Processing einzulesen, solange es weiß, dass es nur zwei Bytes zu erwarten hat. Stellen Sie aber sicher, dass die Sendeseite den auf der Empfangsseite verwendeten Typ nicht überlaufen lässt.

Byteordnung
: Stellen Sie sicher, dass die Bytes innerhalb eines `int` oder `long` in der Reihenfolge gesendet werden, die der Empfänger erwartet.

Synchronisation
: Stellen Sie sicher, dass der Empfänger den Anfang und das Ende einer Nachricht erkennt. Wenn Sie mitten im Stream mit der Verarbeitung anfangen, erhalten Sie keine gültigen Daten. Sie erreichen das, indem Sie eine Bytefolge senden, die in den Nutzdaten selbst nicht vorkommt. Wenn Sie zum Beispiel Binärwerte von `analogRead` senden wollen, liegen sie im Bereich von 0 bis 1023, d.h., das höherwertige Byte muss kleiner als 4 sein (der `int`-Wert 1023 wird in den Bytes 3 und 255 gespeichert). Es kann daher keine Daten geben, bei denen zwei aufeinanderfolgende Bytes größer als 3 sind. Zwei Bytes mit dem Wert 4 (oder jeder Wert höher als 3) können daher keine gültigen Daten sein und können genutzt werden, um den Anfang oder das Ende einer Nachricht anzuzeigen.

Strukturversatz
: Wenn Sie Daten als Strukturen senden oder empfangen, müssen Sie darauf achten, dass der *Versatz* auf beiden Seiten der gleiche ist (Informationen hierzu finden Sie in der Compiler-Dokumentation). Der Versatz beschreibt das vom Compiler verwendete Auffüllen (engl. padding) zum Ausrichten der Datenelemente unterschiedlicher Größen in einer Struktur.

Fluss-Steuerung
: Nutzen Sie entweder eine Übertragungsgeschwindigkeit, die sicherstellt, dass der Empfänger mit dem Sender mithalten kann, oder verwenden Sie irgendeine Form der *Fluss-Steuerung*. Die Fluss-Steuerung verwendet ein Signal (engl. Handshake), der dem Sender mitteilt, dass der Empfänger bereit ist, weitere Daten zu empfangen.

Siehe auch

In Kapitel 2 finden Sie weiterführende Informationen zu den in Arduino-Sketches verwendeten Variablentypen.

Sehen Sie sich auch die Arduino- Referenzen für `lowByte` unter *http://www.arduino.cc/en/Reference/LowByte* und `highByte` unter *http://www.arduino.cc/en/Reference/HighByte* an.

Der Arduino-Compiler packt Strukturen an Bytegrenzen. Sehen Sie in der Dokumentation des auf Ihrem Computer verwendeten Compilers nach, wie Sie den gleichen Versatz hinbekommen. Wenn Ihnen nicht klar ist, wie Sie das machen können, sollten Sie Daten nicht über Strukturen versenden.

Weiterführende Informationen zur Fluss-Steuerung finden Sie unter *http://en.wikipedia.org/wiki/Flow_control*.

4.7 Binärdaten vom Arduino auf einem Computer empfangen

Problem

Sie wollen mit einer Programmiersprache wie Processing vom Arduino gesendete Binärdaten verarbeiten. Zum Beispiel wollen Sie auf Arduino-Nachrichten reagieren, die in Rezept 4.6 gesendet wurden.

Lösung

Die Lösung hängt von der Programmierumgebung ab, die Sie auf Ihrem PC oder Mac verwenden. Wenn Sie noch kein Programmierwerkzeug bevorzugen und eines suchen, das leicht zu lernen ist und gut mit Arduino zusammenarbeitet, dann ist Processing eine ausgezeichnete Wahl.

Hier zwei Zeilen Processing-Code, die ein Byte einlesen. Es stammt aus dem `SimpleRead`-Beispiel (aus der Einführung zu diesem Kapitel):

```
if ( myPort.available() > 0) {  // Wenn Daten verfügbar sind,
 val = myPort.read();      // einlesen und in val speichern
```

Wie Sie sehen können, ähnelt das stark dem Arduino-Code, den Sie schon in früheren Rezepten gesehen haben.

Nachfolgend ein Processing-Sketch, der die Größe eines Rechtecks proportional zu den Integerwerten festlegt, die vom Arduino-Sketch in Rezept 4.6 gesendet werden:

```
/*
 * ReceiveBinaryData_P
 *
 * portIndex muss auf den Port gesetzt werden, mit dem der Arduino verbunden ist
 */
import processing.serial.*;

Serial myPort;     // Serial-Objekt erzeugen
short portIndex = 1;  // com-Port wählen, 0 ist der erste Port

char HEADER = 'H';
int value1, value2;      // Vom seriellen Port empfangene Daten

void setup()
{
 size(600, 600);
 // Seriellen Port öffnen, mit dem der Arduino verbunden ist.
 String portName = Serial.list()[portIndex];
 println(Serial.list());
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this, portName, 9600);
}

void draw()
{
 // Header und zwei binäre Integerwerte (16 Bit) einlesen:
 if ( myPort.available() >= 5)  // Sobald 5 Bytes verfügbar sind,
 {
  if( myPort.read() == HEADER) // Header-Zeichen?
  {
   value1 = myPort.read();          // Lese niederwertiges Byte ein
   value1 =  myPort.read() * 256 + value1; // Füge höherwertiges Byte hinzu

   value2 = myPort.read();          // Lese niederwertiges Byte ein
   value2 =  myPort.read() * 256 + value2; // Füge das höherwertige Byte hinzu

   println("Empfangene Nachricht: " + value1 + "," + value2);
  }
 }
 background(255);        // Hintergrundfarbe ist weiß
 fill(0);            // Füllfarbe ist schwarz

 // Zeichne Rechteck mit den Koordinaten, die vom Arduino empfangen wurden
 rect(0, 0, value1,value2);
}
```

Diskussion

Processing hat Arduino beeinflusst, und die beiden sind sich bewusst sehr ähnlich. Die `setup`-Funktion wird bei Processing zur Einmal-Initialisierung genutzt, genauso wie bei Arduino. Processing besitzt ein Ausgabefenster, und `setup` legt dessen Größe auf 600 × 600 Pixel fest, indem es `size(600,600)` aufruft.

Die Zeile `String portName = Serial.list()[portIndex];` wählt den seriellen Port aus – bei Processing sind alle verfügbaren seriellen Ports im Objekt `Serial.list` enthalten, und dieses Beispiel nutzt den Wert einer Variablen namens `portIndex`. `println(Serial.list())` gibt alle verfügbaren Ports aus und die Zeile `myPort = new Serial(this, portName, 9600);` öffnet den mit `portName` angegebenen Port. Sie müssen sicherstellen, dass `portIndex` auf den seriellen Port gesetzt ist, mit dem Ihr Arduino verbunden ist (üblicherweise mit dem ersten Port bei einem Mac; unter Windows ist es üblicherweise der letzte Port, wenn der Arduino als letztes serielles Gerät installiert wurde).

Die `draw`-Funktion funktioniert bei Processing wie der `loop` in Arduino, d.h., sie wird wiederholt aufgerufen. Der Code in `draw` prüft, ob Daten am seriellen Port verfügbar sind. Ist dass der Fall, werden die Bytes gelesen und in die Integerwerte umgewandelt, die diese Bytes repräsentieren. Ein Rechteck wird dann basierend auf den empfangenen Integerwerten gezeichnet.

Siehe auch

Weiterführende Informationen zu Processing erhalten Sie auf der Processing-Website (*http://processing.org/*).

4.8 Binäre Werte aus Processing an den Arduino senden

Problem

Sie wollen binäre Bytes, Integer- oder long-Werte von Processing an den Arduino senden. Zum Beispiel wollen Sie eine Nachricht senden, die aus einem Identifier-»Tag« und zwei 16-Bit-Werten besteht.

Lösung

Verwenden Sie den folgenden Code:

```
// Processing Sketch

/* SendingBinaryToArduino
 * Sprache: Processing
 */
import processing.serial.*;

Serial myPort;  // Serial-Objekt erzeugen
public static final char HEADER   = 'H';
public static final char MOUSE_TAG = 'M';

void setup()
{
 size(512, 512);
 String portName = Serial.list()[1];
 myPort = new Serial(this, portName, 9600);
}
```

```
void draw(){
}

void serialEvent(Serial p) {
 // handle incoming serial data
 String inString = myPort.readStringUntil('\n');
 if(inString != null) {
  print( inString );  // Textstring vom Arduino ausgeben
 }
}

void mousePressed() {
 sendMessage(MOUSE_TAG, mouseX, mouseY);
}

void sendMessage(char tag, int x, int y){
 // Sende gegebenen Tag und Wert an seriellen Port
 myPort.write(HEADER);
 myPort.write(tag);
 myPort.write((char)(x / 256)); // MSB
 myPort.write(x & 0xff); //LSB
 myPort.write((char)(y / 256)); // MSB
 myPort.write(y & 0xff); //LSB
}
```

Wird die Maus im Processing-Fenster angeklickt, wird `sendMessage` mit einem 8-Bit-Tag aufgerufen, der anzeigt, dass es sich um eine Maus-Nachricht handelt, sowie die `x`- und `y`-Koordinaten der Maus in zwei 16-Bit-Werten. Die `sendMessage`-Funktion sendet die 16-Bit `x`- und `y`-Werte in zwei Bytes, wobei das höherwertige Byte (most significant byte, MSB) zuerst übertragen wird.

Hier der Arduino-Code, der diese Nachricht empfängt und das Ergebnis wieder an Processing zurückgibt:

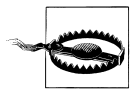
```
// BinaryDataFromProcessing
// Diese Definitionen müssen denen des Senders entsprechen:
const char HEADER     = 'H';
const char MOUSE_TAG  = 'M';
const int  TOTAL_BYTES = 6 ; // Gesamtgröße der Nachricht

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

void loop(){
 if ( Serial.available() >= TOTAL_BYTES)
 {
   if( Serial.read() == HEADER)
  {
    char tag = Serial.read();
    if(tag == MOUSE_TAG)
    {
     int x = Serial.read() * 256;
     x = x + Serial.read();
```

```
      int y = Serial.read() * 256;
      y = y + Serial.read();
      Serial.print("Maus-Nachricht empfangen, x = ");
      Serial.print(x);
      Serial.print(", y = ");
      Serial.println(y);
     }
     else
     {
      Serial.print("Unbekannter Tag in Nachricht: ");
      Serial.write(tag);
     }
    }
   }
}
```

Diskussion

Der Processing-Code sendet ein Header-Byte, das den Beginn einer gültigen Nachricht anzeigt. Das ist notwendig, damit sich Arduino synchronisieren kann, falls es in der Mitte einer Nachricht einsteigt oder falls bei der seriellen Verbindung Daten verloren gehen können (etwa bei einer Drahtlosverbindung). Der Tag bietet eine weitere Möglichkeit, die Gültigkeit einer Nachricht zu überprüfen und ermöglicht es darüber hinaus, zusätzliche Nachrichtentypen individuell zu verarbeiten. In diesem Beispiel wird die Funktion mit drei Parametern aufgerufen: einem Tag und den 16-Bit x- und y-Koordinaten der Maus.

Der Arduino-Code stellt sicher, dass mindestens `MESSAGE_BYTES` empfangen wurden, damit die Nachricht nicht verarbeitet wird, bevor alle benötigten Daten verfügbar sind. Nachdem Header und Tag überprüft wurden, werden die 16-Bit-Werte in zwei Bytes eingelesen. Das erste Byte wird mit 256 multipliziert, um den ursprünglichen Wert des höherwertigen Bytes wiederherzustellen.



Sender und Empfänger müssen die gleiche Nachrichtengröße nutzen, damit Binärnachrichten korrekt verarbeitet werden können. Soll die Anzahl der zu sendenden Bytes erhöht oder verringert werden, müssen Sie `TOTAL_BYTES` im Arduino-Code entsprechend anpassen.

4.9 Den Wert mehrerer Arduino-Pins senden

Problem

Sie wollen Gruppen binärer Byte-, Integer- oder long-Werte vom Arduino senden. Zum Beispiel könnten Sie die Werte der digitalen oder analogen Pins an Processing senden wollen.

Lösung

Das Rezept sendet einen Header gefolgt von einem Integerwert mit den Bitwerten der Digitalpins 2 bis 13. Darauf folgen sechs Integerzahlen mit den Werten der Analogpins 0 bis 5. Kapitel 5 enthält viele Rezepte, die die Werte der Analog- und Digitalpins setzen. Diese können Sie nutzen, um diesen Sketch zu testen:

```
/*
 * SendBinaryFields
 * Sendet Werte der Digital- und Analogpins als Binärdaten
 */

const char HEADER = 'H'; // Headerzeichen leitet den
                // Beginn der Nachricht ein

void setup()
{
 Serial.begin(9600);
 for(int i=2; i <= 13; i++)
 {
  pinMode(i, INPUT);    // Pins 2 bis 13 sind Eingänge
  digitalWrite(i, HIGH);  // Pullups einschalten
 }
}

void loop()
{
 Serial.write(HEADER); // Header senden
 // Bitwerte der Pins in Integer ablegen
 int values = 0;
 int bit = 0;

 for(int i=2; i <= 13; i++)
 {
  bitWrite(values, bit, digitalRead(i)); // Bit abhängig vom Wert des Pins
                        // auf 0 oder 1 setzen
  bit = bit + 1;              // Nächstes Bit
 }
 sendBinary(values); // Integer senden

 for(int i=0; i < 6; i++)
 {
  values = analogRead(i);
  sendBinary(values); // Integer senden
 }
 delay(1000); // Einmal pro Sekunde senden
}

// Funktion zum Senden des gegebenen Integerwertes über den seriellen Port
void sendBinary( int value)
{
 // Sende die zwei Bytes, aus denen ein Integer besteht
 Serial.write(lowByte(value)); // Niederwertiges Bytes senden
 Serial.write(highByte(value)); // Höherwertiges Byte senden
}
```

Diskussion

Der Code sendet einen Header (das Zeichen H), gefolgt von einem Integerwert, der die Werte der Digitalpins enthält. Er nutzt die bitRead-Funktion, um ein einzelnes Bit im Integerwert zu setzen, das dem Wert des Pins entspricht (siehe Kapitel 3). Er sendet dann sechs Integerwerte, die von den Analogports eingelesen wurden (weitere Informationen finden Sie in Kapitel 5). Alle Integerwerte werden mit sendBinary übertragen, das wir in Rezept 4.6 vorgestellt haben. Die Nachricht ist 15 Byte lang – 1 Byte für den Header, 2 Bytes für die Werte der Digitalpins und 12 Bytes für die sechs Werte der Analogpins. Der Code für die Digital- und Analogeingänge wird in Kapitel 5 erläutert.

Wenn wir annehmen, dass analog 0 an Pin 0, 100 an Pin 1 und 200 an Pin 2 bis 500 an Pin 5 anliegen, während die Digitalpins 2 bis 7 HIGH und die Pins 8 bis 13 LOW sind, dann sehen die Dezimalwerte der gesendeten Bytes wie folgt aus:

```
72  // Das Zeichen 'H' - das ist der Header
   // Zwei Bytes (niederwertig/höherwertig) enthalten die Bits für die Pins 2-13
63  // Binär 00111111 : die Pins 2-7 sind angeschaltet
0   // Die Pins 8-13 sind ausgeschaltet

   // Zwei Bytes für jeden Analogpin
0   // Pin 0 hat den Integerwert 0, der in zwei Bytes gesendet wird
0

100  // Pin 1 hat den Wert 100, was als ein Byte mit 100 und ein Byte mit 0 gesendet wird
0
...
   // Pin 5 hat den Wert 500
244  // Rest der Division von 500 durch 256
1   //  So oft kann 500 durch 256 geteilt werden
```

Der nachfolgende Processing-Code liest die Nachricht ein und gibt die Werte in der Processing-Konsole aus:

```
// Processing Sketch

/*
 * ReceiveMultipleFieldsBinary_P
 *
 * portIndex muss auf den Port gesetzt sein, mit dem der Arduino verbunden ist
*/

import processing.serial.*;

Serial myPort;     // Serial-Objekt erzeugen
short portIndex = 1;  // com-Port wählen, 0 ist der erste Port

char HEADER = 'H';

void setup()
{
 size(200, 200);
 // Verbindung mit dem Arduino herstellen.
 String portName = Serial.list()[portIndex];
 println(Serial.list());
```

```
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this, portName, 9600);
}

void draw()
{
int val;

 if ( myPort.available() >= 15)  // Warten, bis gesamte Nachricht verfügbar ist
 {
  if( myPort.read() == HEADER) // Header-Zeichen?
  {
   println("Nachricht empfangen:");
   // Header gefunden
   // Integer mit Bitwerten einlesen
   val = readArduinoInt();
   // Wert jedes Pins ausgeben
   for(int pin=2, bit=1; pin <= 13; pin++){
    print("Digitalpin " + pin + " = " );
    int isSet = (val & bit);
    if( isSet == 0) {
     println("0");
    }
    else{
     println("1");
    }
    bit = bit * 2; //Bit zur nächsten Binärstelle schieben
   }
   println();
   // Die sechs Analogwerte ausgeben
   for(int i=0; i < 6; i ++){
    val = readArduinoInt();
    println("Analogport " + i + "= " + val);
   }
   println("----");
  }
 }
}

// Integerwert aus seriellen Bytes erzeugen  (niederwertig/höherwertig)
int readArduinoInt()
{
 int val;    // Über seriellen Port empfangene Daten

 val = myPort.read();         // Niederwertiges Byte einlesen
 val =  myPort.read() * 256 + val; // Höherwertiges Byte hinzufügen
 return val;
}
```

Der Processing-Code wartet, bis 15 Zeichen eingegangen sind. Ist das erste Zeichen der Header, ruft er eine Funktion namens `readArduinoInt` auf, um zwei Bytes einzulesen und wieder in einen Integerwert umzuwandeln. Mit Hilfe einer mathematischen Umkehroperation (als Gegenstück zu den durchgeführten Arduino-Operationen) gelangen wir dann an die einzelnen Bits der Digitalpins. Die folgenden sechs Integerwerte repräsentieren die Analogwerte.

Siehe auch

Um Arduino-Werte an den Computer zu senden, oder die Pins über den Computer zu setzen, ohne sich um das Board Gedanken machen zu müssen, können Sie Firmata (*http://www.firmata.org* nutzen. Die Firmata-Bibliothek und Beispiel-Sketches (File→Examples→Firmata) sind in der Arduino-Software-Distribution enthalten, und es gibt auch eine Bibliothek für Processing. Sie laden den Firmata-Code auf den Arduino hoch, legen mit dem Computer fest, welche Pins Ein- und Ausgänge sind, und setzen oder lesen dann diese Pins.

4.10 Den Mauszeiger eines PCs oder Macs bewegen

Problem

Sie wollen Arduino durch Bewegen des Mauszeigers mit einer Anwendung auf Ihrem Computer interagieren lassen, etwa durch die Positionierung der Maus basierend auf Informationen vom Arduino. Stellen Sie sich beispielsweise vor, Sie hätten einen Wii Nunchuck (siehe Rezept 13.2) an Ihren Arduino angeschlossen und möchten, dass Ihre Handbewegungen die Position des Mauszeigers in einem PC-Programm steuern.

Lösung

Sie können serielle Befehle senden, die einem auf dem Zielrechner laufenden Programm die Position des Mauszeigers angeben. Hier ein Sketch, der den Mauszeiger basierend auf der Position zweier Potentiometer steuert:

```
// SerialMouse Sketch
const int buttonPin = 2;  //LOW am Digitalpin aktiviert Maus

const int potXPin = 4;   // Analogpins für Potis
const int potYPin = 5;

void setup()
{
 Serial.begin(9600);
 pinMode(buttonPin, INPUT);
 digitalWrite(buttonPin, HIGH); // Pullups einschalten
}

void loop()
{
 int x = (512 - analogRead(potXPin)) / 4;  // Bereich ist -127 bis +127
 int y = (512 - analogRead(potYPin)) / 4;
 Serial.print("Data,");
 Serial.print(x,DEC);
 Serial.print(",");
 Serial.print(y,DEC);
 Serial.print(",");
 if(digitalRead(buttonPin) == LOW)
  Serial.print(1); // Sende 1, wenn Button gedrückt ist
else
```

```
  Serial.print(0);
 Serial.println(",");
 delay(50); // Sende Position 20-mal pro Sekunde
}
```

Abbildung 4-4 zeigt den Anschluss zweier Potentiometer (Details finden Sie in Kapitel 5 for more details). Der Schalter wurde einbezogen, damit Sie die Arduino-Maussteuerung durch Schließen oder Öffnen des Kontakts kontrollieren können.

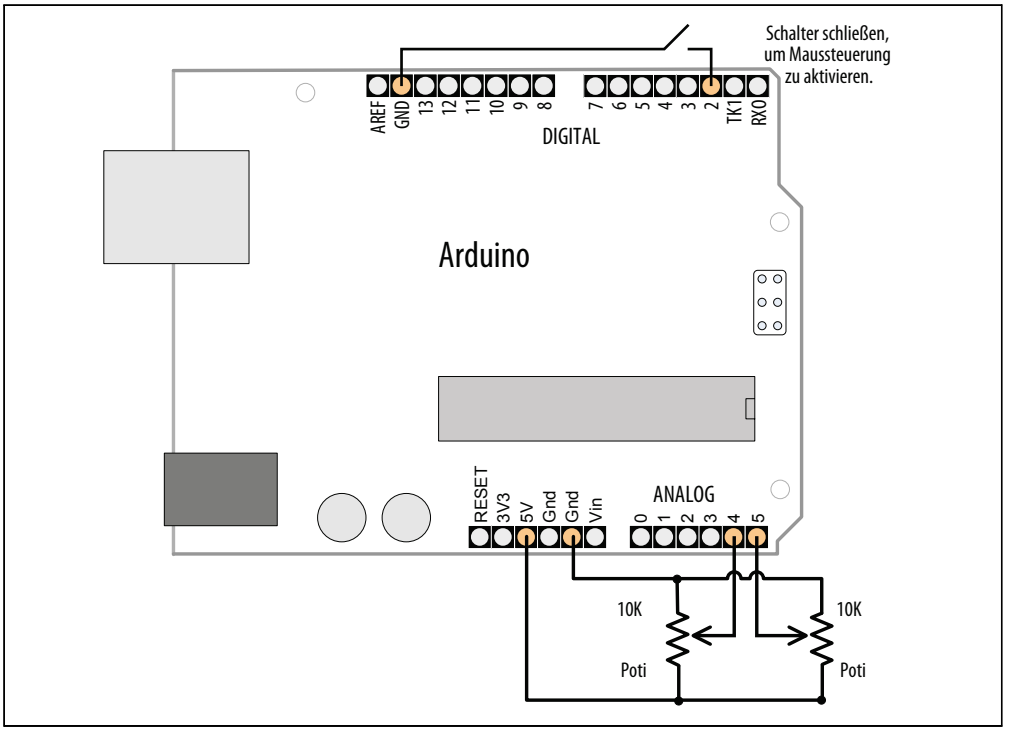


Abbildung 4-4: Anschluss zweier Potentiometer zur Maussteuerung

Der Processing-Code basiert auf dem Code aus Rezept 4.4, wurde aber um Code zur Maussteuerung ergänzt:

```
// Processing Sketch

/*
 * ArduinoMouse.pde (Processing Sketch)
 */

/* WARNING: Der Sketch übernimmt die Maus
 Drücken Sie Escape, um den laufenden Sketch zu schließen */

import java.awt.AWTException;
import java.awt.Robot;
import processing.serial.*;
```

```
Serial   myPort;  // Serial-Objekt erzeugen
arduMouse myMouse;  // Arduino-kontrollierte Maus erzeugen

public static final short LF = 10;     // ASCII-Linefeed
public static final short portIndex = 1;  // com-Port wählen,
                        // 0 ist der erste Port

int posX, posY, btn; // Daten der Nachricht werden hier abgelegt

void setup() {
 size(200, 200);
 println(Serial.list());
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this,Serial.list()[portIndex], 9600);
 myMouse = new arduMouse();
 btn = 0; // Maus ausschalten, bis von Arduino angefordert
}

void draw() {
  if ( btn != 0)
    myMouse.move(posX, posY); // Maus an empfangene x- und y-Position bewegen
}

void serialEvent(Serial p) {
 String message = myPort.readStringUntil(LF); // Serielle Daten einlesen
 if(message != null)
 {
  //print(message);
  String [] data  = message.split(","); // Kommaseparierte Nachricht zerlegen
  if ( data[0].equals("Data"))// Auf Header prüfen
  {
   if( data.length > 3 )
   {
    try {
     posX = Integer.parseInt(data[1]);
     posY = Integer.parseInt(data[2]);
     btn  = Integer.parseInt(data[3]);
    }
    catch (Throwable t) {
     println("."); // Parsing-Fehler
     print(message);
    }
   }
  }
 }
}

class arduMouse {
 Robot myRobot;   // Objekt der Robot-Klasse erzeugen
 static final short rate = 4; // Multiplikator zur Korrektur der Bewegungsgeschwindigkeit
 int centerX, centerY;
 arduMouse() {
  try {
   myRobot = new Robot();
  }
  catch (AWTException e) {
   e.printStackTrace();
```

```
  }
  Dimension screen = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
  centerY =  (int)screen.getHeight() / 2 ;
  centerX =  (int)screen.getWidth() / 2;
 }
 // Methode bewegt die Maus von der Mitte des Bildschirms um den angegebenen Offset
 void move(int offsetX, int offsetY) {
  myRobot.mouseMove(centerX + (rate* offsetX), centerY - (rate * offsetY));
 }
}
```

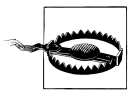
Der Processing-Code zerlegt die Nachricht mit den x- und y-Koordinaten und sendet die Daten an die `mouseMove`-Methode der Java-Klasse `Robot`. Im Beispiel nutzt die Robot-Klasse einen Wrapper namens `arduMouse`, der eine `move`-Methode zur Verfügung stellt, die eine Skalierung auf die Bildschirmgröße vornimmt.

Diskussion

Diese Technik zur Steuerung von auf dem Computer laufenden Anwendungen ist einfach zu implementieren und sollte mit allen Betriebssystemen funktionieren, die Processing ausführen können. Muss die Bewegungsrichtung der x- oder y-Achse umgekehrt werden, ändern Sie einfach das Vorzeichen der Achse im Processing-Sketch:

```
posX = -Integer.parseInt(data[1]); // Minuszeichen invertiert Achse
```

Einige Plattformen verlangen besondere Privilegien oder Erweiterungen, um auf unterster Ebene auf Eingaben zugreifen zu können. Wenn Sie keine Kontrolle über die Maus erlangen, sehen Sie in der Dokumentation Ihres Betriebssystems nach.



Ein außer Kontrolle geratenes `Robot`-Objekt kann Ihnen die Kontrolle über die Maus und die Tastatur entziehen, wenn es in einer Endlosschleife läuft. In diesem Rezept senden wir den Pegel von Digitalpin 2 an Processing, um die Kontrolle zu aktivieren bzw. zu deaktivieren.

Boards mit dem ATmeg32U4-Controller können eine USB-Maus direkt emulieren. Das Arduino Leonardo und das PJRC Teensy liefern Beispiele mit, die zeigen, wie man eine USB-Maus emuliert.

Leonardo-Board:
http://blog.makezine.com/archive/2011/09/arduino-leonardo-opens-doors-to-product-development.html

Teensy-Beispiel für USB-Maus:
http://www.pjrc.com/teensy/usb_mouse.html

Siehe auch

Unter *http://java.sun.com/j2se/1.3/docs/api/java/awt/Robot.html* finden Sie weiterführende Informationen zur Java `Robot`-Klasse.

Einen Artikel zum Einsatz der `Robot`-Klasse finden Sie unter *http://www.developer.com/java/other/article.php/10936_2212401_1*.

Wenn Sie eine Windows-Programmiersprache bevorzugen, können Sie die Low-Level-Funktion `SendInput` der Windows-API nutzen, um Tastatur- und Maus-Events in den Eingabestream einzufügen. Weitere Informationen finden Sie unter *http://msdn.microsoft.com/en-us/library/ms646310(VS.85).aspx*.

Rezept 4.11 zeigt, wie man diese Technik nutzt, um Google Earth zu steuern.

4.11 Google Earth per Arduino steuern

Problem

Sie wollen über mit dem Arduino verbundene Sensoren die Bewegungen in einer Anwendung wie Google Earth steuern. Zum Beispiel sollen Sensoren, die Ihre Handbewegungen erkennen, zur Steuerung des Flugsimulators in Google Earth verwendet werden. Die Sensoren könnten einen Joystick (siehe Rezept 6.17) oder einen Wii Nunchuck (siehe Rezept 13.2) nutzen.

Lösung

Google Earth erlaubt einen »Flug« über die Erde, wobei man sich Satellitenbilder, Land- und Geländerkarten sowie Gebäude in 3D ansehen kann (siehe Abbildung 4-5). Es verfügt über einen Flugsimulator, der über eine Maus gesteuert werden kann. Dieses Rezept verwendet die Techniken, die in Rezept 4.10 genutzt werden und kombiniert sie mit einem Joystick, der an den Arduino angeschlossen ist.

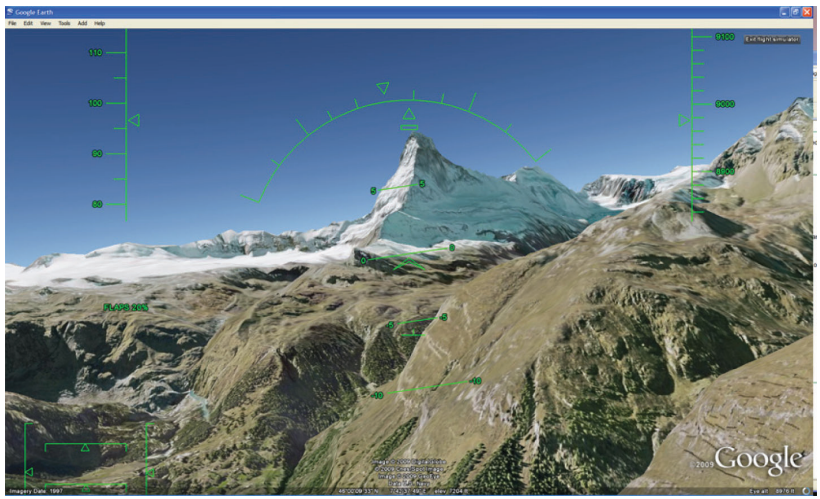


Abbildung 4-5: Google Earth-Flugsimulator

Der Arduino-Code sendet die horizontale und vertikale Position, die durch das Lesen eines Eingabegerätes wie etwa eines Joysticks bestimmt wird. Dazu gibt es die unterschiedlichsten Möglichkeiten, etwa die Schaltung aus Rezept 4.10, die ausgezeichnet funktioniert, wenn Sie einen alten, mit Potentiometern arbeitenden Joystick finden, den Sie dazu nutzen können.

Diskussion

Google Earth steht als kostenloser Download zur Verfügung. Sie können es von der Google-Website *http://earth.google.com/download-earth.html* herunterladen. Laden Sie die Version für Ihr Betriebssystem herunter und installieren Sie sie auf Ihrem Computer. Starten Sie Google Earth und wählen Sie aus dem Tools-Menü den Flight Simulator. Wählen Sie ein Flugzeug (die SR22 ist einfacher zu fliegen als die F16) und einen Flughafen. Die Joystick-Unterstützung muss deaktiviert bleiben – Sie werden die Arduino-kontrollierte Maus verwenden, um das Flugzeug zu fliegen. Klicken Sie den Start Flight-Button an (fliegt das Flugzeug beim Start bereits, können Sie die Leertaste drücken, um den Simulator anzuhalten, damit Sie den Processing-Sketch starten können).

Laden Sie den Arduino-Sketch aus Rezept 4.10 hoch und führen Sie den Processing-Sketch aus diesem Rezept auf Ihrem Computer aus. Machen Sie Google Earth zum aktiven Fenster, indem Sie es anklicken. Aktivieren Sie die Arduino-Maussteuerung, indem Sie Pin 2 mit Masse verbinden.

Sie sind nun startklar. Drücken Sie ein paarmal die Seite-hoch-Taste, um Gas zu geben (und drücken Sie dann die Leeraste, falls Sie den Simulator angehalten haben). Wenn die SR22 eine Geschwindigkeit von etwas über 100 Knoten erreicht, können Sie den Stick »hochziehen« und fliegen. Informationen zur Simulatorsteuerung finden Sie im Google-Hilfemenü.

Sobald Sie Ihren Flug beendet haben, können Sie die Arduino-Maussteuerung ausschalten, indem Sie Pin 2 von Masse trennen, und die Kontrolle wieder an die Computermaus übergeben.

Hier eine weitere Variante, die Nachrichten an den Processing-Sketch sendet. Sie kombiniert den Wii Nunchuck-Code aus Rezept 13.2 mit einer in Rezept 16.5 diskutierten Bibliothek. Die Verschaltung ist in Rezept 13.2 zu sehen:

```
/*
 * WiichuckSerial
 *
 * Verwendet die Nunchuck-Bibliothek aus Rezep 16.5
 * Sendet Daten in kommaseparierten Feldern
 * Kommagetrennte Label-Strings können vom Empfänger
 * genutzt werden, um die Felder zu identifizieren
 */


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

```
// Werte, die zum Sensor hinzuaddiert werden, um
// bei zentriertem Cursor Nullwerte zu erreichen
int offsetX, offsetY, offsetZ;

#include <Wire.h>
#include "Nunchuck.h"
void setup()
{
  Serial.begin(57600);
  nunchuckSetPowerpins();
  nunchuckInit(); // Initialisierungs-Handshake senden
  nunchuckRead(); // Erstes Mal ignorieren
  delay(50);
}
void loop()
{
 nunchuckRead();
 delay(6);
 nunchuck_get_data();
 boolean btnC = nunchuckGetValue(wii_btnC);
 boolean btnZ = nunchuckGetValue(wii_btnZ);

 if(btnC) {
  offsetX = 127 - nunchuckGetValue(wii_accelX) ;
  offsetY = 127 - nunchuckGetValue(wii_accelY) ;
 }
 Serial.print("Data,");
 printAccel(nunchuckGetValue(wii_accelX),offsetX) ;
 printAccel(nunchuckGetValue(wii_accelY),offsetY) ;
 printButton(nunchuckGetValue(wii_btnZ));

 Serial.println();
}

void printAccel(int value, int offset)
{
 Serial.print(adjReading(value, 127-50, 127+50, offset));
 Serial.print(",");
}

void printJoy(int value)
{
 Serial.print(adjReading(value,0, 255, 0));
 Serial.print(",");
}

void printButton(int value)
{
 if( value != 0)
   value = 127;
 Serial.print(value,DEC);
 Serial.print(",");
}

int adjReading( int value, int min, int max, int offset)
{
  value = constrain(value + offset, min, max);
```

```
  value = map(value, min, max, -127, 127);
  return value;
}
```

Diese Sketches nutzen eine serielle Geschwindigkeit von 57600 Baud, um die Latenz zu minimieren. Wenn Sie die Arduino-Ausgaben über den seriellen Monitor beobachten wollen, müssen Sie die Baudrate entsprechend anpassen. Für die meisten anderen Sketches in diesem Buch müssen Sie die Baudrate aber wieder auf 9600 Baud zurücksetzen. Wenn Sie keinen Wii Nunchuck besitzen, können Sie den Arduino-Sketch aus Rezept 4.10 verwenden, müssen dann aber die Baudrate des Sketches auf 57600 erhöhen und den Sketch auf den Arduino hochladen.

Sie können statt der Werte des Beschleunigungsmessers auch die Werte des Nunchuck-Joysticks zurückliefern, indem Sie die beiden mit `printAccel` beginnenden Zeilen wie folgt ändern:

```
printJoy(nunchuckGetValue(wii_joyX));
printJoy(nunchuckGetValue(wii_joyY));
```

Sie können den Processing-Sketch aus Rezept 4.10 verwenden, doch die nachfolgende verbesserte Version gibt die Steuerposition im Processing-Fenster aus und aktiviert den Flugsimulator mit dem Nunchuck-Button 'z':

```
/**
 * GoogleEarth_FS.pde
 *
 * Steuert Google-Flugsimulator über CSV-Sensordaten
 */

import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.InputEvent;
import processing.serial.*;
Serial myPort;  // Serial-Objekt erzeugen

arduMouse myMouse;

String message = null;
int maxDataFields = 7; // 3 Achsen Beschleunigung, 2 Buttons, 2 Joystick-Achsen
boolean isStarted = false;
int accelX, accelY, btnZ; // Daten der Nachrichtenfelder werden hier abgelegt


void setup() {
 size(260, 260);
 PFont fontA = createFont("Arial.normal", 12);
 textFont(fontA);

 short portIndex = 1;  // com-Port wählen, 0 ist der erste Port
 String portName = Serial.list()[portIndex];
 println(Serial.list());
 println(" Verbinde mit -> " + portName) ;
 myPort = new Serial(this, portName, 57600);
```

```
 myMouse = new arduMouse();

 fill(0);
 text("Starte Google FS in der Mitte des Bildschirms", 5, 40);
 text("Richte den Mauszeiger in Google Earth mittig aus", 10, 60);
 text("Zum Spielen Nunchuck Z-Button drücken und loslassen", 10, 80);
 text("Z-Button erneut drücken, um Maus anzuhalten", 20, 100);
}

void draw() {
 processMessages();
 if (isStarted == false) {
  if ( btnZ != 0) {
   println("Button loslassen, um zu starten");
   do{ processMessages();}
     while(btnZ != 0);
   myMouse.mousePress(InputEvent.BUTTON1_MASK); // SIM starten
   isStarted = true;
  }
 }
 else
 {
  if ( btnZ != 0) {
   isStarted = false;
   background(204);
   text("Zum Spielen Z-Button loslassen", 20, 100);
   print("Angehalten, ");
  }
  else{
   myMouse.move(accelX, accelY); // Maus an empfangene x- und y-Position bewegen
   fill(0);
   stroke(255, 0, 0);
   background(#8CE7FC);
   ellipse(127+accelX, 127+accelY, 4, 4);
  }
 }
}

void processMessages() {
 while (myPort.available () > 0) {
  message = myPort.readStringUntil(10);
  if (message != null) {
   //print(message);
   String [] data  = message.split(","); // CSV-Nachricht zerlegen
   if ( data[0].equals("Data"))// Header?
   {
    try {
     accelX = Integer.parseInt(data[1]);
     accelY = Integer.parseInt(data[2]);
     btnZ = Integer.parseInt(data[3]);
    }
    catch (Throwable t) {
     println("."); // Parsing-Fehler
    }
   }
  }
 }
```

```
}

class arduMouse {
 Robot myRobot;   // Robot-Objekt erzeugen;
 static final short rate = 4; // Zurückzulegende Pixel
 int centerX, centerY;
 arduMouse() {
  try {
   myRobot = new Robot();
  }
  catch (AWTException e) {
   e.printStackTrace();
  }
  Dimension screen = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
  centerY =  (int)screen.getHeight() / 2 ;
  centerX =  (int)screen.getWidth() / 2;
 }
 // Maus von der Mitte des Bildschirms zum angegeben Offset bewegen
 void move(int offsetX, int offsetY) {
  myRobot.mouseMove(centerX + (rate* offsetX), centerY - (rate * offsetY));
 }
 // Drücken der Maustaste simulieren
 void mousePress( int button) {
  myRobot.mousePress(button) ;
 }
}
```

Siehe auch

Die Google-Earth-Website enthält den Code zum Herunterladen und Instruktionen, um ihn auf dem Computer ans Laufen zu bringen: *http://earth.google.com/*.

4.12 Arduino-Daten in einer Datei auf dem Computer festhalten

Problem

Sie wollen eine Datei anlegen, die vom Arduino über den seriellen Port empfangene Informationen aufnimmt. Zum Beispiel wollen Sie die Werte der digitalen und analogen Pins in regelmäßigen Zeitabständen in einer Logdatei speichern.

Lösung

Wir haben in früheren Rezepten erläutert, wie man Informationen vom Arduino an den Computer schickt. Diese Lösung verwendet den gleichen Arduino-Code wie in Rezept 4.9. Der Processing-Sketch, der das Logging übernimmt, basiert ebenfalls auf dem in diesem Rezept beschriebenen Processing-Sketch.

Der Processing-Sketch erzeugt eine Datei (mit dem aktuellen Datum und der Uhrzeit als Dateiname) im gleichen Verzeichnis, in dem auch der Processing-Sketch liegt. Vom Ar-

duino empfangene Nachrichten werden der Datei hinzugefügt. Das Drücken einer beliebigen Taste speichert die Datei und beendet das Programm:

```
/*
 * ReceiveMultipleFieldsBinaryToFile_P
 *
 * portIndex muss den Port angeben, mit dem der Arduino verbunden ist
 * Diese Version basiert auf ReceiveMultipleFieldsBinary und speichert die Daten in einer Datei
 * Drücken Sie eine beliebige Taste, um das Logging zu beenden und die Datei zu speichern
 */

import processing.serial.*;


PrintWriter output;
DateFormat fnameFormat= new SimpleDateFormat("yyMMdd_HHmm");
DateFormat  timeFormat = new SimpleDateFormat("hh:mm:ss");
String fileName;

Serial myPort;     // Serial-Objekt erzeugen
short portIndex = 0;  // com-Port wählen, 0 ist der erste Port
char HEADER = 'H';

void setup()
{
 size(200, 200);
 // Verbindung zum Arduino herstellen.
 String portName = Serial.list()[portIndex];
 println(Serial.list());
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this, portName, 9600);
 Date now = new Date();
 fileName = fnameFormat.format(now);
 output = createWriter(fileName + ".txt"); // Datei im Sketch-Ordner speichern
}

void draw()
{
 int val;
 String time;

 if ( myPort.available() >= 15)  // Auf vollständige Nachricht warten
 {
  if( myPort.read() == HEADER) // Header-Zeichen?
  {
   String timeString = timeFormat.format(new Date());
   println("Nachricht empfangen am " + timeString);
   output.println(timeString);
   // Header gefunden
   // Integer mit Bitwerten einlesen
   val = readArduinoInt();
   // Wert jedes Bits ausgeben
   for(int pin=2, bit=1; pin <= 13; pin++){
    print("Digitalpin " + pin + " = " );
    output.print("Digitalpin " + pin + " = " );
    int isSet = (val & bit);
    if( isSet == 0){
```

```
      println("0");
      output.println("0");
     }
     else {
      println("1");
      output.println("0");
     }
     bit = bit * 2; // Bit verschieben
    }
    // Sechs Analogwerte ausgeben
    for(int i=0; i < 6; i ++){
     val = readArduinoInt();
     println("Analogport " + i + "= " + val);
     output.println("Analogport " + i + "= " + val);
    }
    println("----");
    output.println("----");
   }
  }
}

void keyPressed() {
 output.flush(); // Restliche Daten in Datei schreiben
 output.close(); // Datei schließen
 exit(); // Programm beenden
}

// Integerwert aus Bytes zusammensetzen
int readArduinoInt()
{
 int val;    // Vom seriellen Port empfangene Daten

 val = myPort.read();      // Niederwertiges Byte einlesen
 val =  myPort.read() * 256 + val; // Höherwertiges Byte hinzuaddieren
 return val;
}
```

Denken Sie daran, `portIndex` auf den seriellen Port einzustellen, mit dem der Arduino verbunden ist.

Diskussion

Der Basisname für die Logdatei wird mit Hilfe der `DateFormat`-Funktion von Processing erzeugt:

```
DateFormat fnameFormat= new SimpleDateFormat("yyMMdd_HHmm");
```

Den vollständigen Dateinamen erzeugt dann Code, der außerdem ein Verzeichnis und eine Dateiendung hinzufügt:

```
output = createWriter(fileName + ".txt");
```

Die Datei wird im gleichen Verzeichnis erzeugt, in dem auch der Processing-Sketch liegt (der Sketch muss mindestens einmal abgespeichert werden, damit das Verzeichnis auch wirklich existiert). Um das Verzeichnis zu öffnen, wählen Sie Sketch→Show Sketch Folder

im Processing-Menü. `createWriter` ist die Processing-Funktion, die die Datei öffnet. Sie erzeugt ein Objekt namens `output`, das für die eigentliche Dateiausgabe sorgt. Der in die Datei geschriebene Text entspricht genau dem, was auf der Konsole in Rezept 4.9 ausgegeben wird, doch Sie können den Datei-Inhalt mit den normalen Processing-Fähigkeiten zur Stringverarbeitung ganz an Ihre Bedürfnisse anpassen. Zum Beispiel erzeugt die folgende Variante der `draw`-Routine eine kommaseparierte Datei, die von einer Tabellenkalkulation oder einer Datenbank gelesen werden kann. Der Rest des Processing-Sketches kann gleichbleiben, auch wenn Sie die Dateiendung vielleicht von *.txt* in *.csv* ändern sollten:

```
void draw()
{
 int val;
 String time;

 if ( myPort.available() >= 15)  // Auf die gesamte Message warten
 {
  if( myPort.read() == HEADER) // Header-Zeichen?
  {
    String timeString = timeFormat.format(new Date());
    output.print(timeString);
    val = readArduinoInt(); // Digitalwerte einlesen, aber nicht ausgeben

   // Sechs Analogswerte durch Kommata getrennt ausgeben
   for(int i=0; i < 6; i ++){
    val = readArduinoInt();
    output.print("," + val);
   }
   output.println();
  }
 }
}
```

Siehe auch

Weiterführende Informationen zu `createWriter` finden Sie unter *http://processing.org/reference/createWriter_.html*.

4.13 Daten an zwei serielle Geräte gleichzeitig senden

Problem

Sie wollen Daten an ein serielles Gerät, z.B. an ein serielles LCD, senden, verwenden den eingebauten seriellen Port aber schon zur Kommunikation mit Ihrem Computer.

Lösung

Bei einem Mega ist das kein Problem, da er vier serielle Hardware-Ports besitzt. Erzeugen Sie einfach zwei serielle Objekte und nutzen Sie einen für das LCD und den anderen für den Computer:

```
void setup() {
 // Zwei serielle Ports beim Mega initialisieren
 Serial.begin(9600);  // Primärer serieller Port
 Serial1.begin(9600); // Mega kann Serial1 bis Serial3 nutzen
}
```

Bei einem Standard-Arduino-Board (wie dem Uno oder dem Duemilanove) gibt es nur einen seriellen Hardware-Port, und Sie müssen einen zusätzlichen seriellen Port per Software emulieren.

Sie können die mitgelieferte SoftwareSerial-Bibliothek verwenden, um Daten an mehrere Geräte zu senden.

Arduino nutzt seit der Release 1.0 eine verbesserte SoftwareSerial-Bibliothek, die auf Mikal Harts NewSoftSerial-Bibliothek basiert. Falls Sie eine Arduino-Release vor 1.0 verwenden, können Sie NewSoftSerial von *http://arduiniana.org/libraries/newsoftserial* herunterladen.

Wählen Sie zwei freie Digitalpin, jeweils einen für das Senden und das Empfangen, und schließen Sie das serielle Gerät daran an. Es ist praktisch, den seriellen Hardware-Port für die Kommunikation mit dem Computer zu verwenden, weil er einen USB-Adapter auf dem Board integriert hat. Verbinden Sie die Sendeleitung des Geräts mit dem Empfangspin und die Empfangsleitung mit dem Sendepin. In Abbildung 4-6 haben wir Pin 2 als Empfangs- und Pin 3 als Sendepin gewählt.

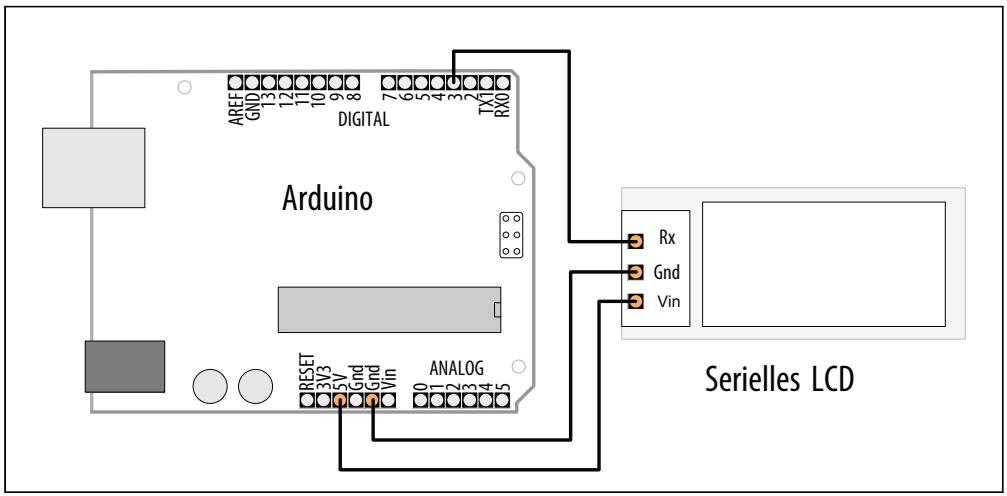


Abbildung 4-6: Serielles Gerät mit seriellem »Soft-Port« verbinden

In Ihrem Sketch erzeugen Sie ein `SoftwareSerial`-Objekt, und teilen ihm mit, welche Pins Sie für den emulierten seriellen Port verwenden. In unserem Beispiel haben wir ein Objekt namens `serial_lcd` erzeugt, das die Pins 2 und 3 nutzt:

```
/*
 * SoftwareSerialOutput Sketch
```

```
 * Daten über seriellen "Software-Port" ausgeben
 */

#include <SoftwareSerial.h>

const int rxpin = 2;       // Zum Empfang verwendeter Pin (wird in dieser Version nicht genutzt)
const int txpin = 3;       // Zum Senden an das LCD verwendeter Pin
SoftwareSerial serial_lcd(rxpin, txpin); // Neuer serieller Port an den Pins 2 und 3

void setup()
{
 Serial.begin(9600); // 9600 Baud für den fest eingebauten seriellen Port
 serial_lcd.begin(9600); // Software-Port ebenfalls mit 9600 initialisieren
}

int number = 0;

void loop()
{
 serial_lcd.print("Die Zahl ist "); // Text an LCD senden
 serial_lcd.println(number);  // Zahl auf LCD ausgeben
 Serial.print("Die Zahl ist ");
 Serial.println(number);     // Zahl auf Konsole ausgeben

 delay(500); // Halbe Sekunde zwischen den Zahlen warten
 number++;  // Zur nächsten Zahl
}
```

Wenn Sie Arduino-Versionen vor 1.0 verwenden, laden Sie die NewSoftSerial-Bibliothek herunter und ersetzen Sie die Referenzen auf SoftwareSerial durch NewSoftSerial:

```
// NewSoftSerial-Version

#include <NewSoftSerial.h>

const int rxpin = 2;       // Zum Empfang verwendeter Pin
const int txpin = 3;       // Zum Senden verwendeter Pin
NewSoftSerial serial_lcd(rxpin, txpin); // Neuer serieller Port an den Pins 2 + 3
```

Der Sketch setzt voraus, dass das serielle LCD mit Pin 3 (siehe Abbildung 4-6) und die serielle Konsole mit dem eingebauten Port verbunden ist. Die Schleife gibt bei jedem Durchlauf die gleiche Meldung aus:

```
Die Zahl ist 0
Die Zahl ist 1
...
```

Diskussion

Jeder Arduino-Mikrocontroller verfügt über mindestens eine eingebaute serielle Schnittstelle. Diese spezielle Hardware ist für die Generierung einer Folge zeitlich genau festgelegter Impulse verantwortlich, die die Gegenstelle als Daten interpretiert, ebenso wie

den Datenstrom, der im Gegenzug empfangen wird. Zwar besitzt der Mega vier dieser Ports, aber die meisten Arduino-Varianten besitzen nur einen. Wenn Sie bei einem Projekt zwei oder mehr serielle Geräte anschließen müssen, benötigen Sie eine Software-Bibliothek, die diese zusätzlichen Ports emuliert. Eine solche Bibliothek für einen »seriellen Software-Port« macht aus einem beliebigen Paar digitaler E/A-Pins einen neuen seriellen Port.

Um einen solchen seriellen Software-Port aufzubauen, wählen Sie ein Paar Pins aus, die als Sende- und Empfangsleitungen fungieren (genau wie die Pins 1 und 0 von Arduinos eingebautem Port). In Abbildung 4-6 werden die Pins 3 und 2 genutzt, aber Sie können alle verfügbaren Digitalpins verwenden. Die Pins 0 und 1 sollten Sie aber meiden, da sie bereits vom eingebauten Port genutzt werden.

Die Syntax beim Schreiben an den Soft-Port ist mit der beim Hardware-Port identisch. Im Beispiel-Sketch werden Daten sowohl an den »echten« als auch an den emulierten Port mit `print()` und `println()` gesendet:

```
serial_lcd.print("Die Zahl ist "); // Text an LCD senden
serial_lcd.println(number);      // Zahl an LCD senden

Serial.print("Die Zahl ist ");    // Text an Hardware-Port senden
Serial.println(number);         // und über seriellen Monitor ausgeben
```

Wenn Sie ein *unidirektionales* serielles Gerät verwenden – das nur sendet oder nur empfängt –, können Sie Ressourcen sparen, indem Sie für die nicht benötigte Leitung einen nicht existierenden Pin im `SoftwareSerial`-Konstruktor angeben. Beispielsweise ist ein serielles LCD grundsätzlich ein reines Ausgabegerät. Wenn Sie keine Daten von ihm erwarten (oder empfangen wollen), können Sie das SoftwareSerial mit folgender Syntax mitteilen:

```
#include <SoftwareSerial.h>
...
const int no_such_pin = 255;
const int txpin = 3;
SoftwareSerial serial_lcd(no_such_pin, txpin); // Nur TX an Pin 3
```

In diesem Fall würden Sie physikalisch nur einen einzelnen Pin (3) mit der »Eingangs-« oder »RX«-Leitung verbinden.

Siehe auch

SoftwareSerial für Arduino 1.0 (und höher) basiert auf NewSoftSerial. Mehr über NewSoftSerial erfahren Sie auf Mikal Harts Website (*http://arduiniana.org/libraries/newsoftserial/*)

4.14 Serielle Daten von zwei Geräten gleichzeitig empfangen

Problem

Sie wollen Daten von einem seriellen Gerät, etwa einem GPS-Modul, empfangen, nutzen den eingebauten seriellen Port aber bereits zur Kommunikation mit Ihrem Computer.

Lösung

Dieses Problem ähnelt dem aus Rezept 4.13, und natürlich ist auch die Lösung sehr ähnlich. Ist der serielle Port des Arduino bereits mit der Konsole verbunden, wenn Sie ein zweites serielles Gerät anschließen wollen, dann müssen Sie mit einer Software-Bibliothek einen seriellen Port emulieren. In diesem Fall empfangen wir Daten über den emulierten Port, statt sie zu schreiben, doch die grundlegende Lösung ist fast gleich.

Beachten Sie die Hinweise zur NewSoftSerial-Bibliothek im vorstehenden Rezept, wenn Sie eine Arduino-Release vor 1.0 verwenden.

Wählen Sie zwei Pins für die Sende- und Empfangsleitungen.

Verbinden Sie Ihr GPS wie in Abbildung 4-7 zu sehen. Rx (die Empfangsleitung) wird in diesem Beispiel nicht verwendet, d.h., Sie können die Rx-Verbindung mit Pin 3 ignorieren, wenn Ihr GPS keinen Empfangspin besitzt.

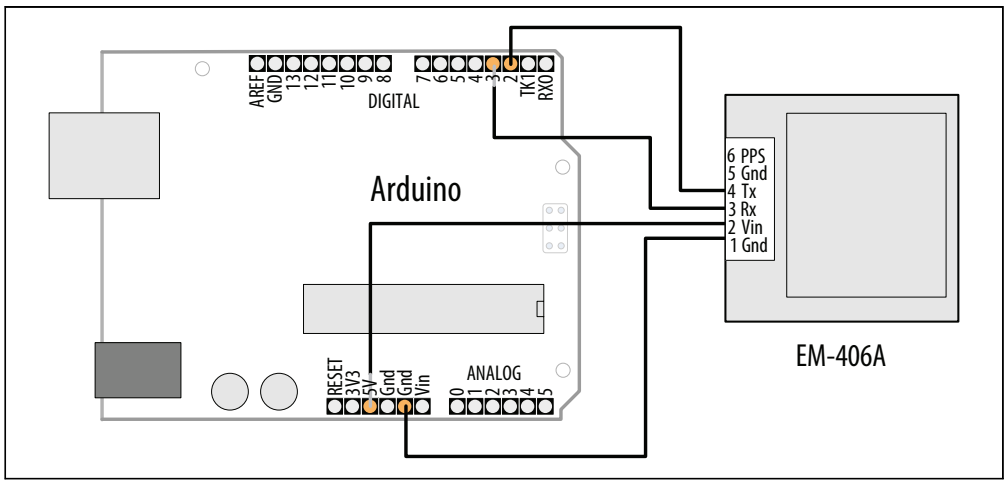


Abbildung 4-7: Serielles GPS-Gerät mit seriellem »Soft-Port« verbinden

Wie in Rezept 4.13 erzeugen Sie ein `SoftwareSerial`-Objekt in Ihrem Sketch und teilen ihm mit, welche Pins verwendet werden. Im folgenden Beispiel definieren wir einen

seriellen Software-Port namens serial_gps und verwenden die Pins 2 und 3 zum Empfangen und Senden:

```
/*
 * SoftwareSerialInput Sketch
 * Daten über seriellen Soft-Port einlesen
 */

#include <SoftwareSerial.h>
const int rxpin = 2;            // GPS-Empfangspin
const int txpin = 3;            // GPS-Sendepin
SoftwareSerial serial_gps(rxpin, txpin); // Neuer serieller Port an Pins 2 und 3

void setup()
{
 Serial.begin(9600); // 9600 Baud für eingebauten seriellen Port
 serial_gps.begin(4800); // Port initialisieren; die meisten GPS-Geräte
              // verwenden 4800 Baud
}

void loop()
{
 if (serial_gps.available() > 0) // Zeichen eingegangen?
 {
  char c = serial_gps.read();  // Dann vom GPS einlesen
  Serial.write(c);        // und über die serielle Konsole ausgeben
 }
}
```

Wenn Sie Arduino-Versionen vor 1.0 verwenden, laden Sie die NewSoftSerial-Bibliothek herunter und ersetzen Sie die Aufrufe von SoftwareSerial durchNewSoftSerial:

```
// NewSoftSerial-Version
#include <NewSoftSerial.h>
const int rxpin = 2;            // GPS-Empfangspin
const int txpin = 3;            // GPS-Sendepin
NewSoftSerial serial_gps(rxpin, txpin); // neuer serieller Port an Pins 2 und 3
```

Dieser kurze Sketch leitet einfach alle vom GPS eingehenden Daten an den seriellen Monitor des Arduino weiter. Wenn Ihr GPS funktioniert und richtig verdrahtet ist, sollten GPS-Daten auf dem seriellen Monitor erscheinen.

Diskussion

Sie initialisieren den emulierten SoftwareSerial-Port, indem Sie die Sende- und Empfangspins übergeben. Der folgende Code richtet den Port so ein, dass an Pin 2 empfangen und an Pin 3 gesendet wird:

```
const int rxpin = 2;            // GPS-Empfangspin
const int txpin = 3;            // GPS-Sendepin
SoftwareSerial serial_gps(rxpin, txpin); // Neuer serieller Port an Pins 2 und 3
```

Der txpin wird in diesem Beispiel nicht verwendet und kann (wie im vorigen Rezept erläutert) auf 255 gesetzt werden, um Pin 3 frei zu lassen.

Die Syntax zum Lesen des emulierten Ports ähnelt stark dem des Lesens vom eingebauten Port. Zuerst wird mit `available()` geprüft, ob ein Zeichen vom GPS eingegangen ist, und dann wird es mit `read()` eingelesen.

Es ist wichtig, daran zu denken, dass serielle Software-Ports Zeit und Ressourcen verbrauchen. Ein emulierter serieller Port muss all das machen, was auch ein Hardware-Port macht, muss dabei aber den gleichen Prozessor nutzen, mit dem Ihr Sketch seine eigentliche Arbeit erledigen will. Geht ein neues Zeichen ein, unterbricht der Prozessor seine aktuelle Arbeit, um es zu verarbeiten. Das kann recht zeitaufwendig sein. Bei 4800 Baud braucht Arduino zum Beispiel etwa zwei Millisekunden, um ein einzelnes Zeichen zu verarbeiten. Nun hören sich zwei Millisekunden nicht nach viel an, doch stellen Sie sich vor, dass die Gegenstelle (z.B. das oben erwähnte GPS-Gerät) 200 bis 250 Zeichen pro Sekunde sendet. Dann verbringt Ihr Sketch 40 bis 50 Prozent seiner Zeit damit, die seriellen Daten zu empfangen. Da bleibt nur sehr wenig Zeit, diese Daten auch tatsächlich zu *verarbeiten*. Das Fazit lautet, dass bei zwei seriellen Geräten das mit der höheren Bandbreitennutzung den eingebauten (Hardware-) Port nutzen sollte. Muss ein viel Bandbreite benötigendes Gerät mit einem seriellen Software-Port verbunden werden, muss der Rest des Sketches sehr effizient sein.

Daten von mehreren SoftwareSerial-Ports empfangen

Mit der SoftwareSerial-Bibliothek, die bei Arduino 1.0 mitgeliefert wird, können Sie mehrere serielle »Soft-Ports« aufbauen. Das ist praktisch, wenn man beispielsweise mehrere XBee-Radios oder serielle Displays im gleichen Projekt steuern will. Der Haken ist, dass zu jedem Zeitpunkt nur einer dieser Ports aktiv Daten empfangen kann. Die zuverlässige Kommunikation über einen Software-Port verlangt die ungeteilte Aufmerksamkeit des Prozessors, weshalb SoftwareSerial nur mit jeweils einem Port aktiv kommunizieren kann.

Es *ist* möglich, in einem Sketch etwas von zwei verschiedenen SoftwareSerial-Ports gleichzeitig zu empfangen. Es gibt viele erfolgreiche Designs, die beispielsweise ein serielles GPS überwachen und dann später Daten von einem XBee verarbeiten. Die Lösung besteht darin, langsam zwischen ihnen zu wechseln und das zweite Gerät nur zu nutzen, wenn die Übertragung beim ersten abgeschlossen ist.

Nehmen wir zum Beispiel an, dass im folgenden Sketch ein entferntes XBee-Modul Befehle sendet. Der Sketch verarbeitet den Befehls-Stream vom »xbee«-Port, bis er ein Signal erhält, dass er Daten des mit dem zweiten SoftwareSerial-Port verbundenen GPS-Gerätes verarbeiten soll. Der Sketch überwacht das GPS dann für 10 Sekunden – gerade lang genug, um eine Ortung vornehmen zu können –, bevor er sich wieder dem XBee widmet.

Bei einem System mit mehreren »Soft-Ports« kann nur jeweils einer aktiv Daten empfangen. Standardmäßig ist der »aktive« Port derjenige, für den `begin()` zuletzt aufgerufen wurde. Sie können den aktiven Port ändern, indem Sie dessen `listen()`-Methode aufrufen. `listen()` weist das SoftwareSerial-System an, den Datenempfang auf einem Port zu unterbrechen und mit einem anderen Port fortzufahren.

Das folgende Code-Fragment zeigt, wie Sie einen Sketch entwerfen können, der Daten zuerst von einem und dann von einem anderen Port einliest:

```
/*
 * MultiRX Sketch
 * Daten von zwei seriellen Software-Ports empfangen
 */
#include <SoftwareSerial.h>
const int rxpin1 = 2;
const int txpin1 = 3;
const int rxpin2 = 4;
const int txpin2 = 5;

SoftwareSerial gps(rxpin1, txpin1); // GPS an Pin 2 und 3
SoftwareSerial xbee(rxpin2, txpin2); // xbee an Pin 4 und 5

void setup()
{
 xbee.begin(9600);
 gps.begin(4800);
 xbee.listen(); // »xbee« ist aktives Gerät
}

void loop()
{
 if (xbee.available() > 0) // xbee ist aktiv. Daten vorhanden?
 {
  if (xbee.read() == 'y') // xbee hat 'y'-Zeichen empfangen
  {
   gps.listen(); // Jetzt GPS verarbeiten

   unsigned long start = millis(); // GPS-Abfrage beginnt
   while (start + 100000 > millis())
   // 10 Sekunden abfragen
   {
    if (gps.available() > 0) // Jetzt ist GPS aktiv
    {
     char c = gps.read();
     // *** GPS-Daten hier verarbeiten
    }
   }
   xbee.listen(); // Nach 10 Sekunden wieder xbee verarbeiten
  }
 }
}
```

Der Sketch ist so entworfen, dass er das XBee-Radio als aktiven Port behandelt, bis ein y-Zeichen empfangen wird. An diesem Punkt wird das GPS zum aktiven Gerät. Nachdem die GPS-Daten für 10 Sekunden verarbeitet wurden, wendet sich der Sketch wieder dem XBee-Port zu. An einem inaktiven Port eingehende Daten werden einfach verworfen.

Beachten Sie, dass diese »aktiver Port«-Beschränkung nur für mehrere *Soft*-Ports gilt. Muss Ihr Design wirklich Daten von mehr als einem seriellen Gerät simultan empfangen, sollten Sie es an einen der eingebauten Hardware-Ports anschließen. Alternativ können Sie Ihre Projekte über externe Chips, sog. *UARTs*, um zusätzliche Hardware-Ports erweitern.

4.15 Serielle Daten mit Processing Senden und Empfangen

Problem

Sie wollen die Processing-Entwicklungsumgebung nutzen, um serielle Daten zu senden und zu empfangen.

Lösung

Sie können Processing vom Download-Bereich der Processing-Website *http://processing.org* herunterladen. Dateien stehen für alle wichtigen Betriebssysteme zur Verfügung. Laden Sie die Datei für Ihr Betriebssystem herunter und entpacken Sie sie an dem Ort, an dem Sie Anwendungen üblicherweise speichern. Bei einem Windows-Computer könnte das ein Ort wie *C:\Programme\Processing* sein, bei einem Mac vielleicht */Programme/Processing.app*.

Wenn Sie Processing auf dem gleichen Computer installiert haben, auf dem auch die Arduino-IDE läuft, dann müssen Sie in Processing nur noch den seriellen Port festlegen. Der folgende Processing-Sketch gibt die verfügbaren seriellen Ports aus:

```
/**
 * GettingStarted
 *
 * Listet die verfügbaren seriellen Ports auf
 * und gibt empfangene Zeichen aus
 */


import processing.serial.*;

Serial myPort;    // Serial-Objekt erzeugen
int portIndex = 0;  // Arduino-Port
int val;       // Vom seriellen Port empfangene Daten

void setup()
{
 size(200, 200);
 println(Serial.list()); // Liste aller Ports ausgeben
 println(" Verbinde mit -> " + Serial.list()[portIndex]);
 myPort = new Serial(this, Serial.list()[portIndex], 9600);
}

void draw()
{
 if ( myPort.available() > 0) // Wenn Daten verfügbar sind,
 {
  val = myPort.read();      // einlesen und in val speichern
  print(val);
 }
}
```

Falls Processing auf einem Computer läuft, auf dem die Arduino-Entwicklungsumgebung nicht installiert ist, müssen Sie möglicherweise die Arduino-USB-Treiber installieren (was in Kapitel 1 beschrieben wird).

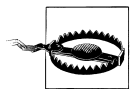
Die Variable `portIndex` muss den Port angeben, der vom Arduino genutzt wird. Sie können die Portnummern im Processing-Textfenster (dem Bereich unter dem Quellcode, nicht dem separaten Ausgabefenster, sehen. Siehe *http://processing.org/reference/environment*). Rezept 1.4 zeigt, wie Sie den seriellen Port ermitteln können, den Ihr Arduino-Board nutzt.

KAPITEL 5

Einfacher digitaler und analoger Input

5.0 Einführung

Die Fähigkeit des Arduino, digitale und analoge Eingänge abzufragen, erlaubt es ihm, auf Sie und die Welt um Sie herum zu reagieren. Dieses Kapitel stellt Techniken vor, die es Ihnen erlauben, nützliche Dinge mit diesen Eingängen anzustellen. Dies ist das erste vieler noch folgender Kapitel, die elektrische Verbindungen zum Arduino behandeln. Wenn Ihnen ein Elektronik-Hintergrund fehlt, sollten Sie sich Anhang A zu elektronischen Komponenten, Anhang B zu Schaltplänen und Datenblättern, Anhang C zum Schaltungsaufbau und Anhang E zur Hardware-Fehlersuche ansehen (alle Anhänge stehen auf unserer Webseite als downloadbare PDF-Texte zur Verfügung). Darüber hinaus stehen viele gute Einführungen zur Verfügung. Drei für Arduino besonders relevante sind *Arduino für Einsteiger* (ISBN 978-3-86899-232-8) von Massimo Banzi, *Making Things Talk* ISBN 978-3-86899-162-8 von Tom Igoe und *Die elektronische Welt mit Arduino entdecken* (ISBN 978-3-89721-319-7) von Erik Bartmann (beide O'Reilly; suchen Sie auf oreilly.de). Andere Bücher, die Hintergrundwissen zu den in diesem und den nächsten Kapiteln behandelten Themen bieten, sind *Getting Started in Electronics* von Forrest Mims (Master Publishing) und *Physical Computing* von Tom Igoe (Cengage).



Wenn die Verdrahtung von Komponenten mit dem Arduino Neuland für Sie ist, müssen Sie sorgfältig darauf achten, wie Sie Dinge anschließen und mit Strom versorgen. Arduino verwendet einen recht robusten Controller-Chip, der einiges verträgt, doch Sie können den Chip beschädigen, wenn Sie die falsche Spannung anlegen oder Ausgabepins kurzschließen. Die meisten Arduino-Controller-Chips werden mit 5 Volt betrieben, und Sie dürfen keine externe Spannung an die Arduino-Pins anlegen, die über dieser Grenze liegt (bzw. 3,3 Volt, wenn der Arduino mit dieser Spannung betrieben wird).

Die meisten Arduino-Boards haben einen Sockel für den Haupt-Chip, damit dieser entfernt und ersetzt werde kann. Falls der Chip beschädigt wird, müssen Sie nicht gleich das ganze Board ersetzen.

Abbildung 5-1 zeigt die Anordnung der Pins bei einem Standard- Arduino-Board. Unter *http://www.arduino.cc/en/Main/Hardware* finden Sie eine Liste aller offiziellen Boards, zusammen mit Links zu Anschlussinformationen. Wenn Ihr Board hier nicht aufgeführt ist, müssen Sie auf der Website des Herstellers nach Anschlussinformationen suchen.

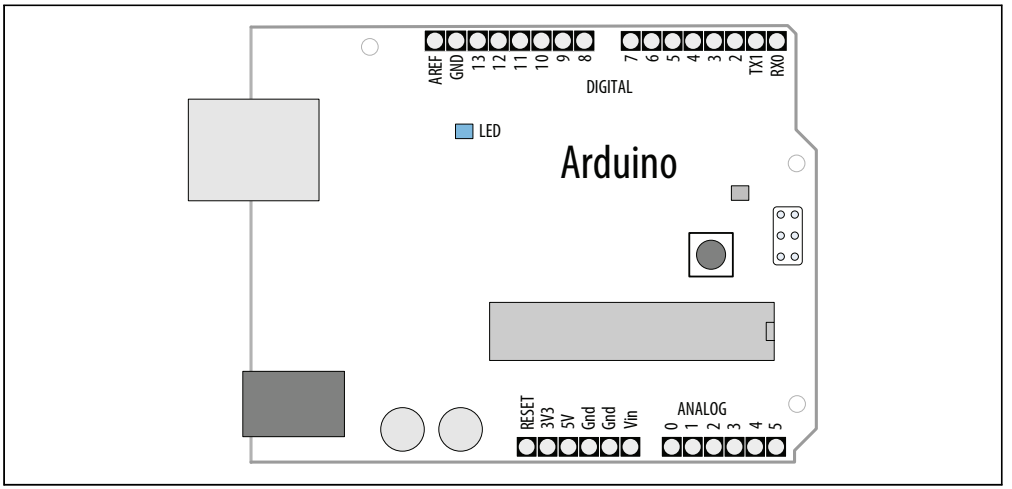


Abbildung 5-1: Digital- und Analogpins bei einem Standard-Arduino-Board

Dieses Kapitel behandelt die Arduino-Pins, die als *digitale* und *analoge* Eingänge dienen können. Digitale Eingangspins erkennen das Vorhandensein oder Fehlen einer Spannung am Pin. Analoge Eingangspins messen einen Spannungspegel an einem Pin.

Die Arduino-Funktion, die einen digitalen Eingang abfragt, ist `digitalRead`. Sie teilt dem Sketch mit, ob an dem Pin eine Spannung anliegt (`HIGH`, 5 Volt) oder nicht (`LOW`, 0 Volt). Die Arduino-Funktion, die einen Pin als Eingang konfiguriert, ist `pinMode(`*`pin`*`, INPUT)`.

Bei einem typischen Board gibt es 14 Digitalpins (von 0 bis 13), die am oberen Rand von Abbildung 5-1 zu sehen sind. Die Pins 0 und 1 (RX und TX) werden für die USB-Verbindung verwendet und sollten für nichts anderes genutzt werden. Sollten Sie bei einem Standard-Board weitere Digitalpins benötigen, können Sie Analogpins als Digitalpins nutzen (die Analogpins 0 bis 5 können als Digitalpins 14 bis 19 verwendet werden).

Arduino 1.0 hat für viele der Pins logische Namen eingeführt. Die Konstanten in Tabelle 5-1 können in allen Funktionen genutzt werden, die eine Pin-Nummer erwarten.

Tabelle 5-1: Bei Arduino 1.0 eingeführte Pin-Konstanten

Konstante	Pin-Nummer	Konstante	Pin-Nummer
A0	Analoger Eingang 0 (Digital 14)	LED_BUILTIN	Onboard-LED (Digital 13)
A1	Analoger Eingang 1 (Digital 15)	SDA	I2C Data (Digital 18)
A2	Analoger Eingang 2 (Digital 16)	SCL	I2C Clock (Digital 19)
A3	Analoger Eingang 3 (Digital 17)	SS	SPI Select (Digital 10)

Tabelle 5-1: Bei Arduino 1.0 eingeführte Pin-Konstanten (Fortsetzung)

Konstante	Pin-Nummer	Konstante	Pin-Nummer
A4	Analoger Eingang 4 (Digital 18)	MOSI	SPI Input (Digital 11)
A5	Analoger Eingang 5 (Digital 19)	MISO	SPI Output (Digital 12)
		SCL	SPI Clock (Digital 13)

Das Mega-Board verfügt über wesentlich mehr digitale und analoge Pins. Die Digitalpins 0 bis 13 und die Analogpins 0 bis 5 befinden sich an der gleichen Stelle wie beim Standard-Board, so dass für das Standard-Board entwickelte Hardware- Shields auch auf den Mega passen. Wie beim Standard-Board können Sie die Analogpins als Digitalpins nutzen, doch beim Mega sind die Analogpins 0 bis 15 die Digitalpins 54 bis 69. Abbildung 5-2 zeigt die Anschlussbelegung des Mega.

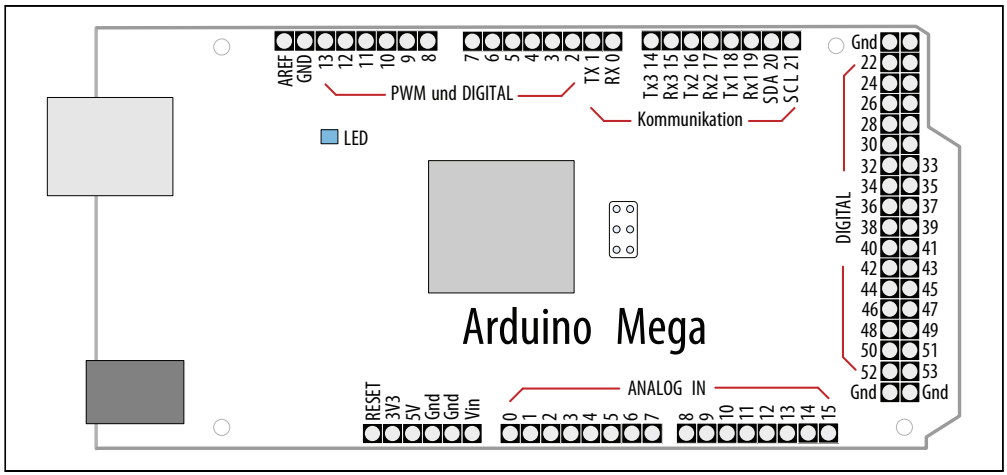


Abbildung 5-2: Arduino Mega Board

Bei den meisten Boards ist die LED mit Pin 13 verbunden, und einige Rezepte nutzen sie als Ausgabeanzeige. Wenn Ihr Board keine LED an Pin 13 besitzt, sehen Sie sich Rezept 7.1 an, um zu erfahren, wie man eine LED an einen digitalen Pin anschließt.

Digitale Eingänge nutzende Rezepte verwenden manchmal externe Widerstände, um die Spannung bereitzustellen, die `digitalRead` verlangt. Diese Widerstände werden *Pullup*-Widerstände (weil sie die Spannung auf die 5V »hochziehen«, engl. »pull up«, mit denen der Widerstand verbunden ist) oder *Pulldown*-Widerstände (weil die Spannung auf 0 Volt »heruntergezogen«, engl. »pull down«, wird) genannt. 10 K-Ohm (und mehr) funktioniert. Informationen zu den in diesem Kapitel verwendeten Bauteilen finden Sie in Anhang A.

Im Gegensatz zu Digitalwerten, die nur an oder aus sind, sind Analogwerte variabel. Der Lautstärkeregler eines Geräts ist dafür ein gutes Beispiel. Die Lautstärke ist nicht einfach nur an oder aus, sonder variiert von laut bis leise. Viele Sensoren variieren die Spannung

so, dass sie der Sensormessung entspricht. Arduino-Code verwendet eine Funktion namens `analogRead`, um einen Wert zurückzuliefern, der proportional zur Spannung ist, die an den Analogpins anliegt. Der Wert liegt bei 0 für 0 Volt und 1023 für 5 Volt. Die Werte entsprechen proportional der Spannung am Pin, d.h., 2,5 Volt (die Hälfte von 5 Volt) liefert einen Wert von etwa 511 (der Hälfte von 1023) zurück. Sie erkennen sechs analoge Eingangspins (mit 0 bis 5 gekennzeichnet) am unteren Ende von Abbildung 5-1. Diese Pins können auch als Digitalpins 14 bis 19 genutzt werden, wenn sie für analoge Aufgaben nicht gebraucht werden. Einige Analog-Rezepte nutzen ein *Potentiometer* (kurz *Poti*, ein *variabler Widerstand*), um die Spannung an einem Pin zu variieren. Ein Wert von 10K ist die beste Wahl für ein Potentiometer, das an einen Analogpin angeschlossen werden soll.

Die meisten Schaltungen in diesem Kapitel lassen sich recht einfach verdrahten, doch Sie sollten die Anschaffung eines lötfreien Steckbretts in Erwägung ziehen, um die Verdrahtung externer Komponenten zu vereinfachen. Verschiedene Modelle wie das Jameco 20723 (zwei Busreihen pro Seite), das RadioShack 276-174 (eine Busreihe pro Seite), das Digi-Key 438-1045-ND und das SparkFun PRT-00137 stehen zur Wahl.

Ein weiteres nützliches Gerät ist ein einfaches Multimeter. Sie können nahezu jedes verwenden, solange es die Spannung und den Widerstand messen kann. Durchgangsprüfung und Strommessung sind nette zusätzliche Optionen. (Jameco 220812, RadioShack 22-810 und SparkFun TOL-00078 bieten diese Features an.)

5.1 Einen Schalter verwenden

Problem

Ihr Sketch soll auf das Schließen eines elektrischen Kontakts reagieren, etwa auf einen Taster oder einen Schalter, oder ein anderes externes Bauelement, das eine elektrische Verbindung herstellt.

Lösung

Verwenden Sie `digitalRead`, um die Stellung eines Schalters zu ermitteln, der mit einem als Eingang eingestellten Digitalpin des Arduino verbunden ist. Der folgende Code schaltet eine LED ein, wenn eine Taste gedrückt wird (Abbildung 5-3 zeigt, wie das verschaltet werden muss):

```
/*
  Pushbutton Sketch
  Schalter an Pin 2 steuert die LED an Pin 13
*/

const int ledPin = 13;       // LED-Pin
const int inputPin = 2;      // Eingangspin für Taster

void setup() {
```

```
 pinMode(ledPin, OUTPUT);      // LED als Ausgang deklarieren
 pinMode(inputPin, INPUT);     // Taster als Eingang deklarieren
}

void loop(){
 int val = digitalRead(inputPin);  // Eingangswert einlesen
 if (val == HIGH)            // Eingang HIGH?
 {
  digitalWrite(ledPin, HIGH);   // LED einschalten, wenn Taster gedrückt
 }
 else
 {
  digitalWrite(ledPin, LOW);    // LED ausschalten
 }
}
```

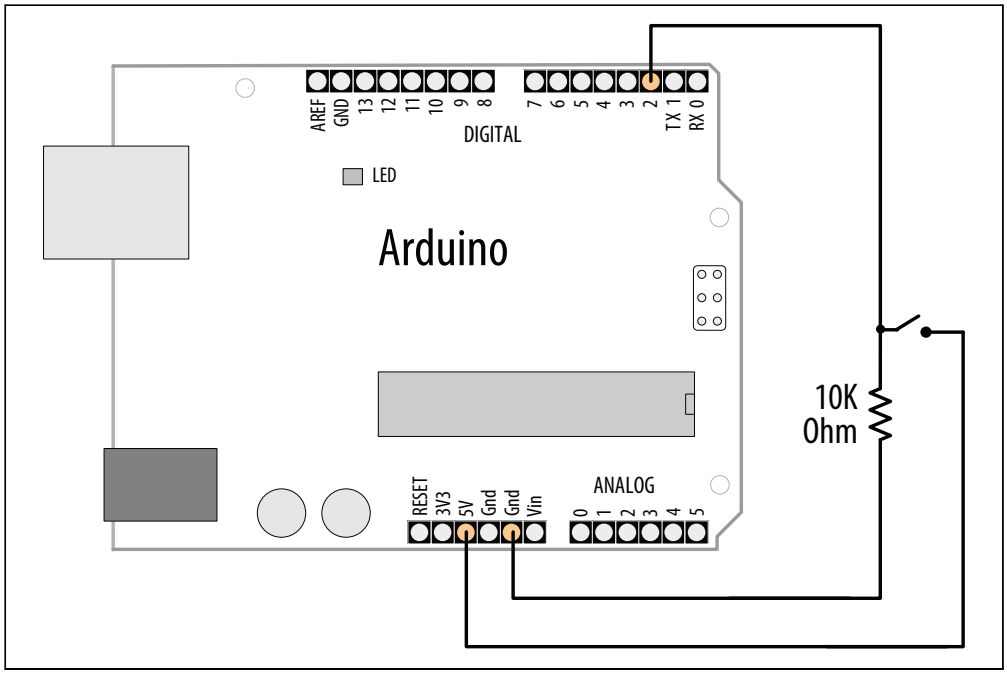


Abbildung 5-3: Über Pulldown-Widerstand angeschlossener Schalter

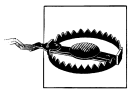
Bei Standard-Arduino-Boards ist eine LED integriert und fest mit Pin 13 verschaltet. Ist das bei Ihrem Board nicht der Fall, zeigt Rezept 7.1, wie man eine LED an einen Arduino-Pin anschließt.

Diskussion

Die `setup`-Funktion konfiguriert den LED-Pin als Ausgang (`OUTPUT`) und den Schalter-Pin als Eingang (`INPUT`).

Ein Pin muss im `OUTPUT`-Modus betrieben werden, damit `digitalWrite` die Ausgangsspannung des Pins kontrollieren kann. Er muss den `INPUT`-Modus nutzen, um den Digitaleingang lesen zu können.

Die Funktion `digitalRead` überwacht die Spannung am Eingangspin (`inputPin`) und gibt `HIGH` zurück, wenn eine Spannung (5 Volt) anliegt, und `LOW`, wenn nicht (0 Volt). Genau genommen wird jede Spannung über 2,5 Volt (der Hälfte der Versorgungsspannung) als `HIGH` betrachtet und jede Spannung darunter als `LOW`. Ist der Pin nicht angeschlossen (*potentialfrei*), kann der von `digitalRead` zurückgelieferte Wert nicht eindeutig vorhergesagt werden (er kann `HIGH` oder `LOW` sein). Der Widerstand in Abbildung 5-3 stellt sicher, dass die Spannung am Pin auf 0 sinkt, wenn der Taster nicht gedrückt ist, da er die Spannung auf Masse (0 Volt) »runterzieht« (engl »pull down«). Wird der Taster gedrückt, entsteht eine Verbindung zwischen dem Pin und +5 Volt, und der von `digitalRead` gelesene Wert wechselt von `LOW` zu `HIGH`.



Schließen Sie nicht mehr als 5 Volt (bzw. 3,3 Volt bei 3,3-Volt-Boards) an die Digital- oder Analogpins an. Das könnte den Pin beschädigen und möglicherweise den ganzen Chip zerstören. Stellen Sie auch sicher, dass Sie die 5 Volt nicht direkt (also ohne Widerstand) mit Masse verbinden. Das beschädigt zwar den Arduino-Chip nicht, tut der Spannungsversorgung aber nicht gut.

Im obigen Beispiel wird der Wert von `digitalRead` in der Variablen `val` gespeichert. Der ist `HIGH`, wenn der Taster gedrückt wird, anderenfalls `LOW`.

Der in diesem Beispiel (und nahezu allen Beispielen in diesem Buch) verwendete Taster stellt einen elektrischen Kontakt her, wenn man ihn drückt. Anderenfalls ist der Kontakt unterbrochen. Diese Taster nennt man Schließer. Artikelnummern finden Sie auf der Website zum Buch (*http://shop.oreilly.com/product/0636920022244.do*). Die andere Art Taster nennt sich Öffner.

Der Ausgangspin, mit dem die LED verbunden ist, wird eingeschaltet, wenn `val` auf `HIGH` gesetzt wird. Die LED leuchtet dann.

Zwar sind beim Arduino alle Digitalpins standardmäßig als Eingänge geschaltet, doch es hat sich in der Praxis bewährt, sie im Sketch explizit zu setzen, um sich bewusst zu machen, welche Pins man verwendet.

Manchmal werden Sie ähnlichen Code sehen, der `true` statt `HIGH` verwendet. Beide Werte sind austauschbar (und manchmal wird auch einfach die `1` verwendet). Ebenso ist `false` mit `LOW` und `0` identisch. Verwenden Sie die Form, die die Logik Ihrer Anwendung am besten ausdrückt.

Sie können nahezu jeden Schalter verwenden, aber die sog. *taktilen Taster* sind besonders beliebt, weil sie günstig sind und direkt auf ein Steckbrett aufgesteckt werden können.

Einige Artikelnummern finden Sie auf der Website zu diesem Buch (*http://shop.oreilly.com/product/0636920022244.do*).

Hier eine andere Möglichkeit, die Logik des obigen Sketches zu implementieren:

```
void loop()
{
  digitalWrite(ledPin, digitalRead(inputPin)); // LED einschalten, wenn Eingangspin
                          // HIGH, sonst ausschalten
}
```

Hier wird der Zustand des Tasters gar nicht erst in einer Variablen gespeichert. Stattdessen wird die LED direkt über den Wert ein- und ausgeschaltet, der von `digitalRead` eingelesen wird. Das ist eine praktische Kurzform, doch wenn Ihnen das zu knapp ist, können Sie auch die andere Variante nutzen. Einen praktischen Performance-Unterschied gibt es nicht.

Der nachfolgende Pullup-Code entspricht dem der Pulldown-Version, kehrt aber die Logik um: Der Wert des Pins sinkt auf `LOW`, wenn der Taster gedrückt wird (die dazugehörige Schaltung sehen Sie in Abbildung 5-4). Stellen Sie sich das so vor, dass die Spannung »runtergeht«, wenn Sie den Taster drücken:

```
void loop()
{
 int val = digitalRead(inputPin);  // Eingangswert einlesen
 if (val == HIGH)           // Eingang HIGH?
 {
  digitalWrite(ledPin, LOW); // LED ausschalten
 }
 else
 {
  digitalWrite(ledPin, HIGH); // LED einschalten
 }
}
```

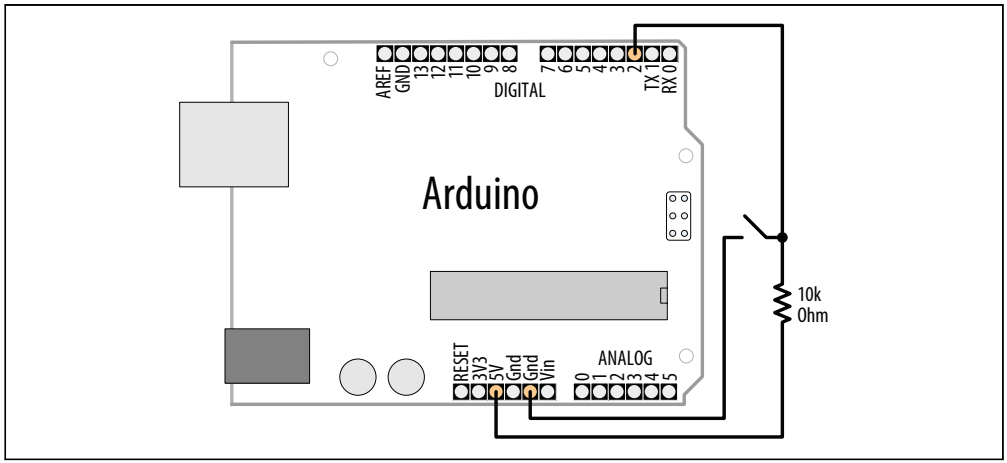


Abbildung 5-4: Über Pullup-Widerstand angeschlossener Taster

Siehe auch

Die Arduino-Referenz zu `digitalRead`: *http://arduino.cc/en/Reference/DigitalRead*

Die Arduino-Referenz zu `digitalWrite`: *http://arduino.cc/en/Reference/DigitalWrite*

Die Arduino-Referenz zu `pinMode`: *http://arduino.cc/en/Reference/PinMode*

Die Arduino-Referenzen zu Konstanten (`HIGH`, `LOW`, etc.): *http://arduino.cc/en/Reference/Constants*

Arduino-Tutorial zu Digitalpins: *http://arduino.cc/en/Tutorial/DigitalPins*

5.2 Taster ohne externen Widerstand verwenden

Problem

Sie wollen eine Schaltung vereinfachen, indem Sie Schalter ohne externe Pullup-Widerstände anschließen.

Lösung

Wie in Rezept 5.1, erläutert, benötigen digitale Eingänge einen Widerstand, um den Pin auf einem bekannten Pegel zu halten, wenn der Taster nicht gedrückt wird. Arduino besitzt interne Pullup-Widerstände, die aktiviert werden können, indem man `HIGH` an einen Pin im `INPUT`-Modus schreibt (den dazugehörigen Code sehen Sie in Rezept 5.1).

Für dieses Beispiel wird der Taster wie in Abbildung 5-5 zu sehen verschaltet. Das entspricht eigentlich genau der Schaltung aus Abbildung 5-4, aber ohne den externen Widerstand.

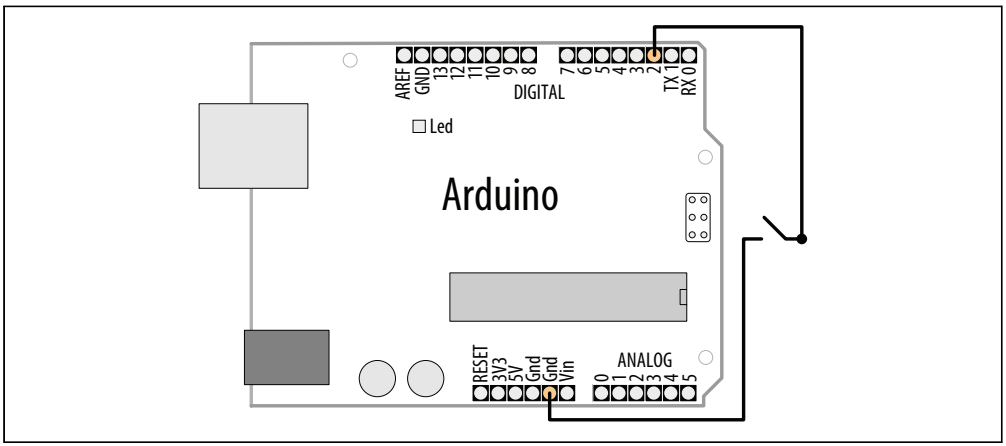


Abbildung 5-5: Verschaltung des Tasters mit internem Pullup-Widerstand

Der Taster ist einfach mit Pin 2 und »Gnd« verbunden. Gnd steht für *Ground*, also Masse, und ist per Definition mit 0 Volt definiert:

```
/*
 Pullup Sketch
 Taster an Pin 2 steuert die LED an Pin 13
*/

const int ledPin = 13;      // Ausgangspin für LED
const int inputPin = 2;     // Eingangspin für Taster

void setup() {
 pinMode(ledPin, OUTPUT);
 pinMode(inputPin, INPUT);
 digitalWrite(inputPin,HIGH); // Internen Pullup für inputPin einschalten
 }
void loop(){
 int val = digitalRead(inputPin); // Einganswert einlesen
 if (val == HIGH)           // Eingang HIGH?
 {
  digitalWrite(ledPin, HIGH);   // LED einschalten
 }
 else
 {
  digitalWrite(ledPin, LOW);    // LED ausschalten
 }
}
```

Auf dem Arduino-Board gibt es mehr als einen Masse-Anschluss. Sie sind alle untereinander verbunden, d.h., Sie können sich den aussuchen, der am besten passt.

Diskussion

Sie aktivieren den internen Pullup-Widerstand, indem Sie ein `HIGH` an den Pin im Eingangsmodus senden. Ein `digitalWrite(`*`pin`*`, HIGH)` für einen Pin im Eingangsmodus ist vielleicht nicht besonders intuitiv, aber man gewöhnt sich daran. Sie können den Pullup deaktivieren, indem Sie ein `LOW` an den Pin senden.

Wenn Sie in Ihrer Anwendung den Pinmodus zwischen Ein- und Ausgang wechseln, müssen Sie daran denken, dass der Zustand des Pins bei `HIGH` oder `LOW` bleibt, wenn Sie den Zustand ändern. Ist also ein Ausgangspin auf `HIGH` gesetzt, wenn Sie in den Eingangsmodus wechseln, bleibt der Pullup aktiv und der Pin erzeugt beim Lesen ein `HIGH`. Ist der Pin mit `digitalWrite(`*`pin`*`, LOW)` auf `LOW` gesetzt worden und wechseln Sie dann mit `pinMode(`*`pin`*`, INPUT)` in den Eingabemodus, bleibt der Pullup aus. Schalten Sie einen Pullup ein, setzt das Wechsel in den Ausgangsmodus den Pin auf `HIGH`, was beispielsweise ungewollt eine angeschlossene LED aufleuchten lassen könnte.

Die internen Pullup-Widerstände sind 20 K-Ohm groß (zwischen 20K und 50K). Das ist für die meisten Anwendungen geeignet, doch einige Bauelemente verlangen kleinere Widerstände. Sehen Sie auf dem Datenblatt des Bauelements nach, welche internen Pullups geeignet sind und welche nicht.

5.3 Das Schließen eines Schalters zuverlässig erkennen

Problem

Sie wollen falsche Eingabewerte aufgrund *prellender Kontakte* vermeiden (das Prellen erzeugt falsche Signale, wenn die Kontakte des Schalters schließen oder öffnen). Diese falschen Signale zu eliminieren, bezeichnet mal als *Entprellen*.

Lösung

Es gibt viele Möglichkeiten, dieses Problem zu lösen. Hier verwenden wir die Schaltung aus Abbildung 5-3 from Rezept 5.1:

```
/*
 * Debounce Sketch
 * Taster an Pin 2 steuert die LED an Pin 13
 * Entprell-Logik verhindert Fehlablesung des Taster-Zustands
 */

const int inputPin = 2;     // Eingangspin
const int ledPin = 13;      // Ausgangspin
const int debounceDelay = 10; // Wartezeit zur Stabilisierng in Millisekunden

// debounce gibt wahr zurück, wenn der Schalter am angegebenen Pin geschlossen und stabil ist
boolean debounce(int pin)
{
 boolean state;
 boolean previousState;

 previousState = digitalRead(pin);      // Zustand des Schalters speichern
 for(int counter=0; counter < debounceDelay; counter++)
 {
   delay(1);          // 1 Millisekunde warten
   state = digitalRead(pin); // Pin einlesen
   if( state != previousState)
   {
     counter = 0; // Zähler zurücksetzen, wenn sich der Zustand ändert
     previousState = state; // und aktuellen Zustand speichern
   }
 }
 // Zustand des Schalters war über die Entprell-Periode hinaus stabil
 return state;
}

void setup()
{
 pinMode(inputPin, INPUT);
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 if (debounce(inputPin))
 {
```

```
  digitalWrite(ledPin, HIGH);
 }
}
```

Die debounce-Funktion wird mit der Nummer des Pins aufgerufen, der entprellt werden soll. Die Funktion liefert true zurück, wenn der Schalter gedrückt wurde und stabil ist. Sie gibt false zurück, wenn nichts gedrückt wurde oder der Zustand nicht stabil ist.

Diskussion

Die debounce-Funktion prüft, ob der Schalter nach einer gewissen Zeitspanne noch den gleichen Wert zurückliefert. Diese Zeitspanne muss so lang sein, dass der Schalter nicht mehr prellt. Möglicherweise müssen Sie eine längere Zeitspanne festlegen (manche Schalter brauchen 50 ms und mehr). Die Funktion prüft über die in debounce definierte Dauer wiederholt den Zustand des Schalters. Bleibt der Schalterwert für diese Zeitspanne gleich, wird dieser Wert zurückgegeben (true, wenn er geschlossen ist, anderenfalls false). Ändert sich der Zustand innerhalb der Entprellperiode, wird der Zähler zurückgesetzt und die Prüfung wird innerhalb der Entprellzeit wiederholt.

Wenn Sie mit Pullup- anstelle von Pulldown-Widerständen arbeiten (siehe Rezept 5.2), müssen Sie den von der debounce-Funktion zurückgelieferten Wert umkehren, da der Zustand auf LOW fällt, wenn der Schalter bei Pullups gedrückt wird. Die Funktion sollte aber true (true ist das Gleiche wie HIGH) zurückgeben, wenn der Taster gedrückt wird. Der debounce-Code für Pullups ist nachfolgend zu sehen. Nur die letzten vier (hervorgehobenen) Zeilen wurden geändert:

```
boolean debounce(int pin)
{
 boolean state;
 boolean previousState;

 previousState = digitalRead(pin);      // Zustand des Schalters speichern
 for(int counter=0; counter < debounceDelay; counter++)
 {
   delay(1);           // 1 Millisekunde warten
   state = digitalRead(pin);  // Pin einlesen
   if( state != previousState)
   {
     counter = 0; // Zähler zurücksetzen, wenn sich der Zustand ändert
     previousState = state;  // und aktuellen Zustand speichern
   }
 }
 // Zustand des Schalters war über die Entprell-Periode hinaus stabil
 if(state == LOW) // LOW heißt gedrückt (da Pullups verwendet werden)
   return true;
 else
  return false;
}
```

Zu Testzwecken können Sie eine count-Variable einbinden, um die Zahl der Tastendrucke auszugeben. Wenn Sie das im seriellen Monitor verfolgen (siehe Kapitel 4), erkennen Sie, ob der Zähler bei jedem Tastendruck um 1 erhöht wird. Erhöhen Sie den Wert für

debounceDelay, bis Zähler und Tastendrucke übereinstimmen. Das folgende Code-Fragment gibt einen count-Wert bei Nutzung der debounce-Funktion aus:

```
int count; // Enthält die Zahl der Tastendrucke

void setup()
{
 pinMode(inPin, INPUT);
 pinMode(outPin, OUTPUT);
 Serial.begin(9600); // Diese Zeile in setup-Funktion einfügen
}

void loop()
{
 if(debounce(inPin))
 {
  digitalWrite(outPin, HIGH);
  count++; // increment count
  Serial.println(count); // Zähler im seriellen Monitor ausgeben
 }
}
```

Diese debounce()-Funktion funktioniert mit einer beliebigen Zahl von Schaltern, doch Sie müssen sicherstellen, dass die genutzten Pins im Eingangsmodus sind.

Ein potentieller Nachteil dieser Methode liegt bei einigen Anwendungen darin, dass beim Aufruf der debounce-Funktion alles warten muss, bis der Schalter stabil ist. In den meisten Fällen spielt das keine Rolle, doch Ihr Sketch könnte sich um andere Dinge kümmern müssen, statt darauf zu warten, dass sich der Schalter stabilisiert. Sie können den Code in Rezept 5.4 nutzen, um dieses Problem zu umgehen.

Siehe auch

Der bei Arduino mitgelieferte Debounce-Beispiel-Sketch. Aus dem File-Menü wählen Sie Examples→Digital→Debounce

5.4 Ermitteln, wie lange eine Taste gedrückt wird

Problem

Ihre Anwendung muss bestimmen, wie lange ein Schalter seinen aktuellen Zustand beibehalten hat. Oder Sie wollen einen Wert inkrementieren, während ein Taster gedrückt wird, und die Inkrementierungsrate soll sich erhöhen, je länger die Taste gedrückt wird (viele elektronische Uhren arbeiten so). Oder Sie wollen wissen, ob eine Taste lange genug gedrückt wurde, um einen stabilen Zustand erreicht zu haben (siehe Rezept 5.3).

Lösung

Der folgende Sketch demonstriert die Nutzung eines Countdown-Timers. Die Verschaltung entspricht der aus Abbildung 5-5 aus dem Rezept Rezept 5.2. Beim Drücken des

Tasters wird ein Timer durch das Erhöhen eines Timer-Zählers gesetzt. Das Loslassen des Tasters startet den Countdown. Der Code entprellt die Taste und erhöht die Zählerrate, je länger der Taster gedrückt bleibt. Der Zähler wird am Anfang um 1 inkrementiert, wenn der Taster (nach dem Entprellen) gedrückt wird. Wird der Taster länger als eine Sekunde gehalten, erhöht sich die Inkrement-Rate um das Vierfache. Wird er länger als vier Sekunden gedrückt, erhöht sie sich um das Zehnfache. Wird der Taster losgelassen, startet der Countdown. Sobald der Zähler bei 0 ankommt, wird ein Pin auf HIGH gesetzt (in diesem Beispiel wird die LED eingeschaltet):

```
/*
 SwitchTime Sketch
 Countdown-Timer mit einem Dekrement von 1/10 Sekunde.
 Schaltet bei 0 die LED ein
 Taster drücken erhöht den Zähler
 Längeres Drücken erhöht die Inkrement-Rate

 */
const int ledPin = 13;          // Ausgangspin
const int inPin = 2;           // Eingangspin

const int  debounceTime = 20;      // Zeit in Millisekunden
                    // bis sich der Schalter stabilisiert
const int  fastIncrement = 1000;    // Nach dieser Zeitspanne (in Millisekunden)
                    // schneller erhöhen
const int  veryFastIncrement = 4000;  // Und nach dieser Zeitspanne
                    // noch schneller erhöhen
int count = 0;              // Zähler wird jede zehntel Sekunde dekrementiert,
                    // bis die 0 erreicht ist

void setup()
{
 pinMode(inPin, INPUT);
 digitalWrite(inPin, HIGH); // Pullup-Widerstand aktivieren
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
}



void loop()
{
 int duration = switchTime();
 if( duration > veryFastIncrement)
  count = count + 10;
 else if ( duration > fastIncrement)
  count = count + 4;
 else if ( duration > debounceTime)
  count = count + 1;

 else
 {
  // Taster nicht gedrückt, also Timer bedienen
  if( count == 0)
   digitalWrite(ledPin, HIGH);  // LED einschalten, wenn Zähler auf 0
  else
```

```
  {
   digitalWrite(ledPin, LOW);  // LED ausschalten, wenn Zähler nicht 0,
   count = count - 1;       // und Zähler dekrementieren
  }
 }
 Serial.println(count);
 delay(100);
}

// Zeit in Millisekunden zurückgeben, die der Taster gedrückt (LOW) wurde
long switchTime()
{
 // Statische Variablen - eine Erklärung finden Sie in der Diskussion
 static unsigned long startTime = 0;  // Zeit, bei der die erste Zustandsänderung erkannt wurde
 static boolean state;          // Aktueller Zustand des Tasters

 if(digitalRead(inPin) != state) // Prüfen, ob sich der Zustand des Tasters geändert hat
 {
  state = ! state;     // Ja, Zustand invertieren
  startTime = millis(); // Zeit speichern
 }
 if( state == LOW)
  return millis() - startTime;  // Taster gedrückt, Zeit in Millisekunden zurückgeben
 else
  return 0; // Wir geben 0 zurück, wenn der Taster nicht gedrückt wurde (HIGH ist);
}
```

Diskussion

Das Herzstück dieses Rezepts bildet die `switchTime`-Funktion. Sie gibt die Zeit in Millisekunden zurück, während der der Taster gedrückt war. Da das Rezept die internen Pullup-Widerstände nutzt (siehe Rezept 5.2), gibt `digitalRead` beim Taster-Pin `LOW` zurück, wenn der Taster gedrückt wurde.

Im `loop` wird der Rückgabewert von `switchTime` untersucht und dann entschieden, was weiter passieren soll. Wurde der Taster lange genug für das höchste Inkrement gedrückt, wird der Zähler um diesen Wert erhöht. Ist das nicht der Fall, wird geprüft, ob der `fast`-Wert verwendet werden soll. Ist auch das nicht der Fall, wird überprüft, ob die Taste lange genug gedrückt wurde, um ein Prellen zu verhindern, und der Zähler wird um den Minimalwert erhöht. Jeweils einer dieser Fälle kann eintreten, und wenn keiner `true` ist, wurde der Taster nicht gedrückt, oder nicht lange genug, um entprellt zu sein. Der Zählerwert wird dann untersucht und die LED bei 0 eingeschaltet. Ist der Zähler noch nicht bei 0 angekommen, bleibt die LED aus, und der Zähler wird dekrementiert.

Sie können die `switchTime`-Funktion auch nur zum Entprellen eines Tasters nutzen. Der folgende Code nutzt die `switchTime`-Funktion zum Entprellen:

```
// Zeit in Millisekunden, die der Taster zur Stabilisierung braucht
const int  debounceTime = 20;

if( switchTime() > debounceTime);
  Serial.print("Taster ist entprellt");
```

Dieser Entprell-Ansatz ist bei mehr als einem Taster recht praktisch, weil Sie einfach schauen können, ob ein Taster schon entprellt ist, und sich dann anderen Aufgaben zuwenden können, während sich sein Zustand stabilisiert. Um das zu implementieren, müssen Sie den aktuellen Zustand des Tasters (gedrückt oder nicht) und den Zeitpunkt der letzten Zustandsänderung speichern. Auch das lässt sich auf unterschiedliche Art lösen – in diesem Beispiel verwenden Sie eine separate Funktion für jeden Taster. Sie können die Variablen für alle Taster zu Beginn des Sketches als *globale Variablen* definieren (»global«, weil sie von überall zugänglich sind). Doch es ist praktischer, die Variablen für jeden Taster innerhalb der Funktion vorzuhalten.

Um die Werte von Variablen zu erhalten, die in Funktionen definiert sind, arbeitet man mit *statischen Variablen*. Statische Variablen innerhalb einer Funktion behalten die Werte auch zwischen den Funktionsaufrufen bei. Der zuletzt gesetzte Wert steht also auch noch zur Verfügung, wenn die Funktion beim nächsten Mal aufgerufen wird. In dieser Hinsicht ähnelt eine statische Variable einer globalen Variablen (die üblicherweise am Anfang des Sketches, außerhalb einer Funktion deklariert wurde), die Sie aus anderen Rezepten kennen. Doch im Gegensatz zu globalen Variablen sind statische Variablen nur innerhalb dieser Funktion erreichbar. Der Vorteil statischer Variablen besteht darin, dass sie von anderen Funktion nicht versehentlich geändert werden können.

Der folgende Sketch zeigt beispielhaft, wie man separate Funktionen für verschiedene Taster hinzufügen kann. Die Verschaltung entspricht der aus Rezept 5.2. Der zweite Taster ist verschaltet wie der erste (siehe Abbildung 5-5), ist aber mit Pin 3 und Masse verbunden:

```
/*
 SwitchTimeMultiple Sketch
 Gibt aus, wie lange mehrere Taster gedrückt wurden
 */

const int switchAPin = 2;         // Pin für Taster A
const int switchBPin = 3;         // Pin für Taster B

// Funktionen mit Referenzen müssen explizit deklariert werden
unsigned long switchTime(int pin, boolean &state, unsigned long &startTime);

void setup()
{
 pinMode(switchAPin, INPUT);
 digitalWrite(switchAPin, HIGH); // Pullup aktivieren
 pinMode(switchBPin, INPUT);
 digitalWrite(switchBPin, HIGH); // Pullup aktivieren
 Serial.begin(9600);
}

void loop()
{
unsigned long time;

 Serial.print("Zeit für Taster A =");
 time = switchATime();
 Serial.print(time);
```

```
 Serial.print(", Zeit für Taster B=");
 time = switchBTime();
 Serial.println(time);
 delay(1000);
}

unsigned long switchTime(int pin, boolean &state, unsigned long  &startTime)
{
 if(digitalRead(pin) != state) // Zustandsänderung des Tasters überprüfen
 {
  state = ! state;    //Ja, Status invertieren
  startTime = millis(); // Zeit speichern
 }
 if( state == LOW)
  return millis() - startTime;  // Zeit in Millisekunden zurückgeben
 else
  return 0; // 0 zurückgeben, wenn Taster nicht gedrückt wurde (HIGH ist);
}


long switchATime()
{
 // Diese Variablen sind statisch  - Erläuterung im Text
 // Zeitpunkt der ersten erkannten Zustandsänderung des Tasters
 static unsigned long startTime = 0;
 static boolean state;          // Aktueller Zustand des Tasters
 return switchTime(switchAPin, state, startTime);
}

long switchBTime()
{
 // Diese Variablen sind statisch  - Erläuterung im Text
 // Zeitpunkt der ersten erkannten Zustandsänderung des Tasters
 static unsigned long startTime = 0;
 static boolean state;          // Aktueller Zustand des Tasters
 return switchTime(switchBPin, state, startTime);
}
```

Die Zeitberechnung erfolgt in der Funktion `switchTime()`. Diese Funktion untersucht und aktualisiert Zustand und Dauer des Tasters. Die Funktion verwendet Referenzen zur Verwaltung der Parameter. Referenzen wurden in Rezept 2.11 erklärt. Eine separate Funktion für jeden Taster (`switchATime()` und `switchBTime()`) wird genutzt, um Startzeit und Zustand jedes Tasters festzuhalten. Da die Variablen, die diese Werte enthalten, als statisch deklariert sind, bleiben die Werte auch erhalten, wenn die Funktion verlassen wird. Die Variablen innerhalb einer Funktion vorzuhalten, stellt sicher, dass die richtige Variable verwendet wird. Die von den Tastern verwendeten Pins sind in globalen Variablen deklariert, da die Werte von `setup` zur Konfiguration der Pins benötigt werden. Da diese Variablen als `const` deklariert sind, erlaubt der Compiler keine Änderung ihrer Werte, d.h., es gibt keine Möglichkeit, sie im Sketch-Code versehentlich zu ändern.

Die Sichtbarkeit von Variablen zu beschränken, wird wichtiger, je komplexer die Projekte werden. Die Arduino-Umgebung bietet hierfür aber eine elegantere Lösung. In Rezept 16.4 wird beschrieben, wie man so etwas über Klassen implementiert.

5.5 Von einer Tastatur lesen

Problem

Sie besitzen eine Matrix-Tastatur und wollen sie in Ihrem Sketch abfragen. Sie könnten beispielsweise eine telefonartige Tastatur wie die SparkFun COM-08653 mit 12 Tasten nutzen wollen.

Lösung

Verschalten Sie die Zeilen und Spalten der Tastatur wie in Abbildung 5-6 zu sehen.

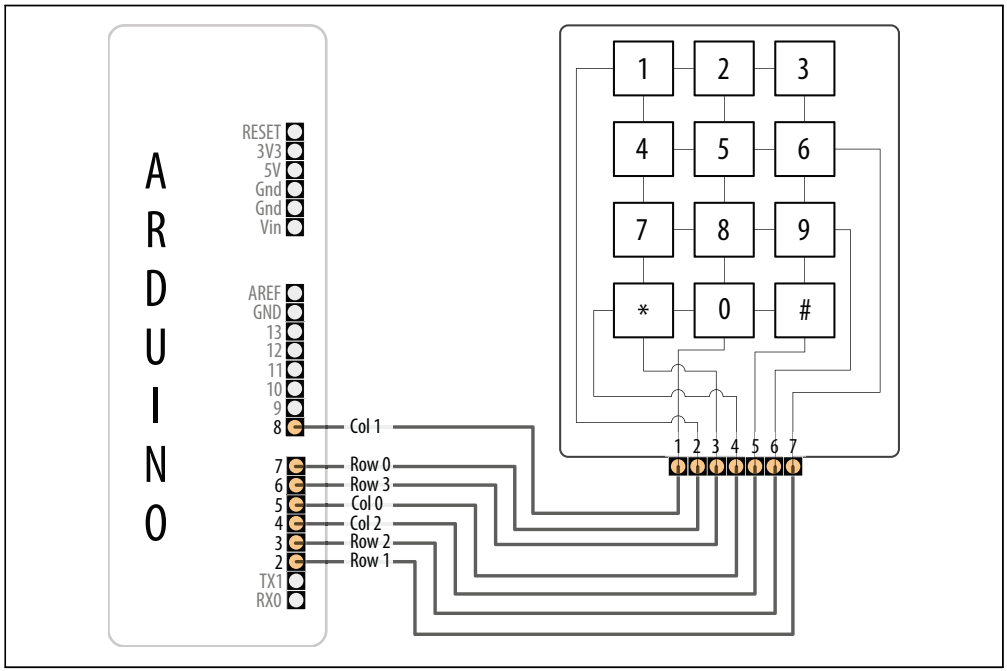


Abbildung 5-6: Verschaltung der SparkFun-Tastatur

Wenn Sie den Arduino wie in Abbildung 5-6 mit der Tastatur verschaltet haben, können Sie mit dem folgenden Sketch über den seriellen Monitor verfolgen, welche Tasten gedrückt worden sind:

```
/*
 Keypad Sketch
 gibt gedrückte Tasten über den seriellen Port aus
*/

const int numRows = 4;     // Zeilen der Tastatur
const int numCols = 3;     // Spalten der Tastatur
const int debounceTime = 20; // Zeit in Millisekunden, bis sich die Taste stabilisiert
```

```
// keymap definiert die Zeichen, die zurückgegeben werden, wenn die entsprechende Taste gedrückt
wird
const char keymap[numRows][numCols] = {
 { '1', '2', '3' } ,
 { '4', '5', '6' } ,
 { '7', '8', '9' } ,
 { '*', '0', '#' }
};

// Diese Arrays bestimmen die Pins, die für die Zeilen und Spalten verwendet werden
const int rowPins[numRows] = { 7, 2, 3, 6 }; // Zeilen 0 bis 3
const int colPins[numCols] = { 5, 8, 4 };   // Spalten 0 bis 2

void setup()
{
 Serial.begin(9600);
 for (int row = 0; row < numRows; row++)
 {
  pinMode(rowPins[row],INPUT);     // Pins für Zeilen als Eingänge schalten
  digitalWrite(rowPins[row],HIGH);  // Pullups aktivieren
 }
 for (int column = 0; column < numCols; column++)
 {
  pinMode(colPins[column],OUTPUT);   // Pins für Spalten als Ausgänge schalten
  digitalWrite(colPins[column],HIGH); // Alle Spalten sind inaktiv
 }
}

void loop()
{
 char key = getKey();
 if( key != 0) {     // Ist das Zeichen nicht 0,
             // wurde eine gültige Taste gedrückt
  Serial.print("Taste: ");
  Serial.println(key);
 }
}

// gibt gedrückte Taste zurück bzw. 0, wenn keine Taste gedrückt wurde
char getKey()
{
 char key = 0;                    // 0 bedeutet, es wurde keine Taste gedrückt

 for(int column = 0; column < numCols; column++)
 {
  digitalWrite(colPins[column],LOW);      // Aktuelle Spalte aktivieren.
  for(int row = 0; row < numRows; row++)   // Alle Zeilen auf
                          // Tastendruck untersuchen.
  {
   if(digitalRead(rowPins[row]) == LOW)   // Taste gedrückt?
   {
    delay(debounceTime);           // Entprellen     while(digitalRead(rowPins[row]) == LOW)
      ;                    // Auf Tastenfreigabe warten
    key = keymap[row][column];        // Festhalten, welche
                          // Taste gedrückt wurde.
   }
  }
```

```
    digitalWrite(colPins[column],HIGH);   // Aktuelle Spalte deaktivieren.
  }
  return key; // gedrückte Taste (oder 0) zurückgeben
}
```

Dieser Sketch funktioniert nur dann, wenn die Verschaltung mit dem Code übereinstimmt. Tabelle 5-2 zeigt, welche Zeilen und Spalten mit welchen Arduino-Pins verbunden sein müssen. Wenn Sie eine andere Tastatur verwenden, müssen Sie die Verbindungen für die Zeilen und Spalten auf dem Datenblatt nachsehen. Achten Sie auf die richtige Verschaltung, da es anderenfalls zu Kurzschlüssen kommen kann, die den Controller beschädigen können.

Tabelle 5-2: Zuordnung der Arduino-Pins zum SparkFun-Anschluss und den Tastatur-Zeilen/-Spalten

Arduino-Pin	Tastatur-Anschluss	Tastatur-Zeile/-Spalte
2	7	Zeile 1
3	6	Zeile 2
4	5	Spalte 2
5	4	Spalte 0
6	3	Zeile 3
7	2	Zeile 0
8	1	Spalte 1

Diskussion

Matrix-Tastaturen bestehen üblicherweise aus (normal offenen) Tastern, die eine Zeile mit einer Spalte verbinden, wenn man sie drückt. (Ein »normal offener« Taster stellt die elektrische Verbindung her, wenn er gedrückt wird.) Abbildung 5-6 zeigt, wie die internen Leitungen die Zeilen und Spalten der Tastatur mit dem Tastaturanschluss verbinden. Jede der vier Zeilen ist mit einem Eingangspin und jede der Spalten mit einem Ausgangspin verbunden. Die `setup`-Funktion setzt die Pin-Modi und aktiviert die Pullup-Widerstände für die Eingangspings (siehe hierzu die Pullup-Rezepte zu Beginn des Kapitels).

Die Funktion `getkey` setzt nacheinander die Pins für jede Spalte auf `LOW` und prüft dann, ob einer der Zeilen-Pins `LOW` ist. Da Pullup-Widerstände genutzt werden, sind die Spalten HIGH, bis eine Taste gedrückt wird (das Drücken einer Taste erzeugt ein `LOW`-Signal am Eingangspin). Ein `LOW` zeigt an, dass die Taste für die Zeile und Spalte gedrückt wird. Eine Zeitverzögerung stellt sicher, dass die Taste nicht prellt (siehe Rezept 5.3). Der Code wartet dann, bis die Taste losgelassen wird, und das mit der Taste verknüpfte Zeichen wird aus dem `keymap`-Array herausgesucht und zurückgegeben. Eine `0` wird zurückgegeben, wenn keine Taste gedrückt wurde.

Eine Bibliothek im Arduino Playground arbeitet ähnlich wie das obige Beispiel, bietet aber eine größere Funktionalität. Die Bibliothek vereinfacht die Arbeit mit einer unterschiedlichen Anzahl von Tasten und kann sich einige Pins auch mit einem LCD teilen. Sie finden die Bibliothek unter *http://www.arduino.cc/playground/Main/KeypadTutorial*.

Siehe auch

Weitere Informationen zur SparkFun 12-Tasten-Tastatur finden Sie unter *http://www.sparkfun.com/commerce/product_info.php?products_id=8653*.

5.6 Analogwerte einlesen

Problem

Sie wollen die Spannung an einem Analogpin einlesen. Vielleicht wollen Sie den Wert eines Potentiometers (Potis) abfragen, oder eines Bauelements oder Sensors, der eine Spannung zwischen 0 und 5 Volt zurückgibt.

Lösung

Der folgende Sketch liest die Spannung an einem Analogpin ein und lässt eine LED mit einer Geschwindigkeit blinken, die proportional zu dem Wert ist, den `analogRead` zurückgibt. Die Spannung wird mit einem Potentiometer geregelt, der wie in Abbildung 5-7 angeschlossen ist:

```
/*
 Pot Sketch
 LED mit einer Geschwindigkeit blinken lassen, die durch die Position eines Potentiometers
bestimmt wird
*/

const int potPin = 0;   // Eingangspin für Potentiometer
const int ledPin = 13;  // Pin für LED
int val = 0;        // Diese Variable enthält den Wert vom Sensor

void setup()
{
 pinMode(ledPin, OUTPUT); // ledPin als Ausgang deklarieren
}

void loop() {
 val = analogRead(potPin);   // Spannung am Poti einlesen
 digitalWrite(ledPin, HIGH); // ledPin einschalten
 delay(val);          // Blinkgeschwindigkeit (in Millisekunden) wird durch Poti-Wert bestimmt
 digitalWrite(ledPin, LOW); // ledPin ausschalten
 delay(val);          // LED bleibt die gleiche Zeitspanne aus
}
```

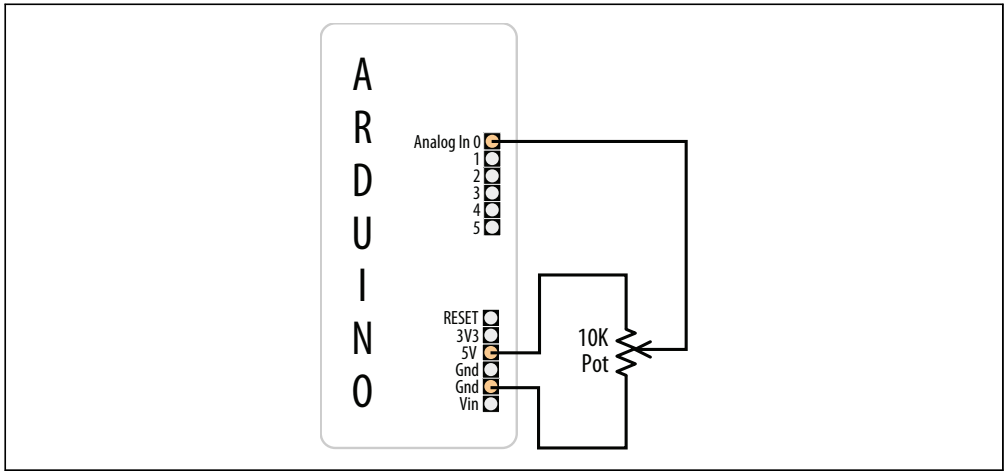


Abbildung 5-7: Ein Potentiometer mit dem Arduino verbinden

Diskussion

Dieser Sketch nutzt die Funktion `analogRead`, um die Spannung am *Schleifer* (dem mittleren Pin) des Potentiometers einzulesen. Ein Poti besitzt drei Pins, von denen zwei mit einem resistiven Material verbunden sind, während der dritte Pin (üblicherweise der mittlere) mit einem Schleifer verbunden ist, den man drehen kann, um den Widerstand beliebig einzustellen. Während das Poti gedreht wird, erhöht sich der Widerstand zwischen dem Schleifer und einem der Pins, während er zwischen Schleifer und dem anderen Pin sinkt. Das Schaltschema in Abbildung 5-7 verbildlicht, wie ein Potentiometer funktioniert. Bewegt man den Schleifer (die Linie mit dem Pfeil) nach unten, verringert sich der Widerstand zur Masse hin, während er sich zu 5V hin erhöht. Während sich der Schleifer nach unten bewegt, nimmt die Spannung am Analogpin ab (bis zu einem Minimum von 0 Volt). Bewegt man den Schleifer nach oben, tritt das Gegenteil ein, d.h., die Spannung am Pin steigt (bis auf ein Maximum von 5 Volt).

Wenn die Spannung am Pin sinkt (und nicht steigt), wenn Sie den Poti »hochdrehen«, vertauschen Sie einfach die Anschlüsse von +5 Volt und Masse.

Die Spannung wird mit `analogRead` gemessen. Der zurückgelieferte Wert ist proportional zur tatsächlichen Spannung am Analogpin. Der Wert 0 wird zurückgegeben, wenn 0 Volt am Pin anliegen, und 1023 bei 5 Volt. Jeder Wert dazwischen ist proportional zum Verhältnis der Spannung am Pin und 5 Volt.

Potentiometer mit einem Wert von 10K-Ohm sind für den Anschluss an Analogpins die beste Wahl. Empfohlene Artikelnummern finden Sie auf der Website zu diesem Buch (*http://shop.oreilly.com/product/0636920022244.do*).

potPin muss nicht als Eingang geschaltet werden. (Das geschieht bei jedem Aufruf von analogRead automatisch.)

Siehe auch

Tipps zum Lesen von Schaltplänen finden Sie in Anhang B.

Arduino-Referenz zu analogRead: *http://www.arduino.cc/en/Reference/AnalogRead*

Arduino für Einsteiger (ISBN 978-3-86899-233-5) von Massimo Banzi

5.7 Wertebereiche ändern

Problem

Sie wollen einen Wertebereich ändern, etwa für einen Wert, den Sie mit analogRead von einem Potentiometer (oder einem anderen Bauelement mit variabler Spannung) eingelesen haben. Stellen Sie sich beispielsweise vor, dass Sie die Position eines Potentiometers als Wert zwischen 0 und 100 Prozent darstellen wollen.

Lösung

Verwenden Sie die Arduino-Funktion map, um Werte innerhalb des von Ihnen gewünschten Bereichs zu skalieren. Der folgende Sketch liest die Spannung am Poti in die Variable val ein und skaliert sie (je nach Position des Potis) auf Werte zwischen 0 und 100. Er lässt die LED mit einer Geschwindigkeit blinken, die zur Spannung am Pin proportional ist, und gibt den skalierten Bereich über den seriellen Port aus (Instruktionen zum Monitoring über den seriellen Port finden Sie in Rezept 4.2). Rezept 5.6 zeigt, wie das Poti anzuschließen ist (siehe Abbildung 5-7):

```
/*
 * Map Sketch
 * Skaliert den Wertebereich des Analogwertes eines Potis auf Werte zwischen 0 und 100
 * Die Blinkgeschwindigkeit der LED reicht von 0 bis 100 Millisekunden
 * und die prozentuale Drehung des Potis wird über den seriellen Port ausgegeben
 */

const int potPin = 0;      // Eingangspin für Potentiometer
int ledPin = 13;          // Pin für LED

void setup()
{
 pinMode(ledPin, OUTPUT);   // ledPin als Ausgang deklarieren
 Serial.begin(9600);
}

void loop() {
```

```
 int val;              // Der Wert vom Sensor
 int percent;            // Der abgebildete Wert

 val = analogRead(potPin);     // Spannung vom Poti einlesen (val liegt
                  // zwischen 0 und 1023)
 percent = map(val,0,1023,0,100); // Prozentwert liegt zwischen 0 und 100.
 digitalWrite(ledPin, HIGH);    // ledPin einschalten
 delay(percent);           // Prozentwert bestimmt Dauer
 digitalWrite(ledPin, LOW);     // ledPin ausschalten
 delay(100 - percent);       // Für 100 - Prozentwert aus bleiben
 Serial.println(percent);      // Prozentwert des Potis ausgeben
}
```

Diskussion

Rezept 5.6 beschreibt, wie die Stellung eines Potis in einen Wert umgewandelt wird. Hier nutzen wir diesen Wert mit der `map`-Funktion, um ihn auf den von Ihnen gewünschten Bereich zu skalieren. In diesem Beispiel wird der von `analogRead` gelieferte Wert (`0` bis `1023`) auf eine Prozentzahl (0 bis 100) abgebildet. Die von `analogRead` zurückgelieferten Werte reichen von `0` bis `1023`, wenn die Spannung zwischen 0 und 5 Volt liegt, aber Sie können alle geeigneten Werte für die Quell- und Zielbereiche verwenden. Zum Beispiel dreht sich ein typischer Poti von einem Ende zum anderen um 270 Grad. Wenn Sie also die Stellung des Potis in Grad angeben wollen, können Sie folgenden Code verwenden:

```
angle = map(val,0,1023,0,270); // Aus analogRead-Wert abgeleitete Position des Potis
```

Die Wertebereiche können auch negativ sein. Wenn Sie zum Beispiel `0` ausgeben wollen, wenn der Poti in der Mitte steht, und negative Werte, wenn er nach links bzw. positive Werte, wenn er nach recht gedreht wird, können Sie folgenden Code verwenden:

```
// Winkel eines 270-Grad-Potis mit 0 in der Mitte ausgeben
angle = map(val,0,1023,-135,135);
```

Die `map`-Funktion ist sehr praktisch, wenn der betrachtete Wertebereich nicht bei 0 beginnt. Wenn Sie zum Beispiel mit einer Batterie arbeiten, bei der die verfügbare Kapazität proportional zu einer Spannung zwischen 1,1 Volt (1100 Millivolt) und 1,5 Volt (1500 Millivolt) liegt, können Sie den folgenden Code nutzen:

```
const int empty = 5000 / 1100; // Leer bei 1,1 Volt (1100mV)
const int full  = 5000 / 1500; // Voll bei 1,5 Volt (1500mV)

int val = analogRead(potPin);          // Spannung einlesen
int percent = map(val, empty, full, 0,100); // Aktuelle Spannung in Prozent umrechnen
Serial.println(percent);
```

Wenn Sie Sensorwerte mit `map` bearbeiten, müssen Sie die Minimal- und Maximalwerte der Sensoren kennen. Sie können die Werte über die serielle Schnittstelle verfolgen, um die kleinsten und größten Werte des Sensors zu bestimmen. Verwenden Sie diese dann als Unter- und Obergrenze der `map`-Funktion.

Lässt sich der Wertebereich nicht im Vorfeld ermitteln, können Sie die Werte bestimmen, indem Sie den Sensor kalibrieren. Rezept 8.11 zeigt eine Technik zur Kalibrierung. Eine

weitere finden Sie im Calibration-Beispiel-Sketch, das mit dem Arduino geliefert wird (Examples→Analog→Calibration).

Wenn Sie `map` mit Werten füttern, die außerhalb der unteren und oberen Grenzen liegen, liegt auch das Ergebnis außerhalb des festgelegten Bereichs. Sie können das mit Hilfe der `constrain`-Funktion unterbinden (siehe Rezept 3.5).

`map` arbeitet mit ganzen Zahlen, d.h., es werden innerhalb des festgelegten Wertebereichs nur ganze Zahlen zurückgegeben. Alle Nachkommastellen werden abgeschnitten, nicht gerundet.

(In Rezept 5.9 finden Sie Details dazu, in welcher Beziehung `analogRead`-Werte zur tatsächlichen Spannung stehen.)

Siehe auch

Die Arduino-Referenz zu `map`: *http://www.arduino.cc/en/Reference/Map*

5.8 Mehr als sechs analoge Eingänge einlesen

Problem

Sie müssen mehr Analogeingänge verarbeiten, als Analogpins zur Verfügung stehen. Ein Standard-Arduino-Board besitzt sechs Analogeingänge (das Mega hat 16) und die Analogeingänge reichen für Ihre Anwendung nicht aus. Sie könnten beispielsweise acht Parameter in Ihrer Anwendung einstellen wollen, indem Sie acht Potentiometer entsprechend justieren.

Lösung

Nutzen Sie einen Multiplexer-Chip, um mehrere Spannungsquellen auszuwählen und mit einem analogen Eingang zu verbinden. Indem Sie die Quellen nacheinander auswählen, können jede nacheinander einlesen. Dieses Rezept nutzt den beliebten 4051-Chip, der wie in Abbildung 5-8 mit dem Arduino verbunden wird. Die Analogeingänge werden mit den 4051-Pins namens Ch 0 bis Ch 7 verbunden. Stellen Sie sicher, dass die Spannung an den Kanal-Eingangspins die 5 Volt niemals übersteigen:

```
/*
 * Multiplexer Sketch
 * Lese 1 von 8 Analogwerten mit Hilfe des 4051-Multiplexers über einen einzelnen Analogpin ein
 */

// Array von Pins, die zur Wahl eines der 8 Eingänge des Multiplexers genutzt werden
const int select[] = {2,3,4}; // Mit Select-Leitungen des 4051 verbundene Pins
const int analogPin = 0;    // Mit Multiplexer-Ausgang verbundener Analogpin

// Diese Funktion liefert den Analogwert für den angegebenen Kanal zurück
int getValue( int channel)
```

```
{
  // Setzt die Auswahlpins auf HIGH und LOW, damit sie dem Binärwert des Kanals entsprechen
  for(int bit = 0; bit < 3; bit++)
  {
    int pin = select[bit]; // Mit Multiplexer-Select-Bit verbundener Pin
    int isBitSet = bitRead(channel, bit); // Wahr, wenn Bit im Kanal gesetzt
    digitalWrite(pin, isBitSet);
  }
  return analogRead(analogPin);
}

void setup()
{
 for(int bit = 0; bit < 3; bit++)
   pinMode(select[bit], OUTPUT);  // Die drei Select-Bits als Ausgänge schalten
 Serial.begin(9600);
}
void loop () {
 // Die Werte aller Kanäle einmal pro Sekunde ausgeben
 for(int channel = 0; channel < 8; channel++)
 {
   int value = getValue(channel);
   Serial.print("Kanal ");
   Serial.print(channel);
   Serial.print(" = ");
   Serial.println(value);
 }
 delay (1000);
}
```

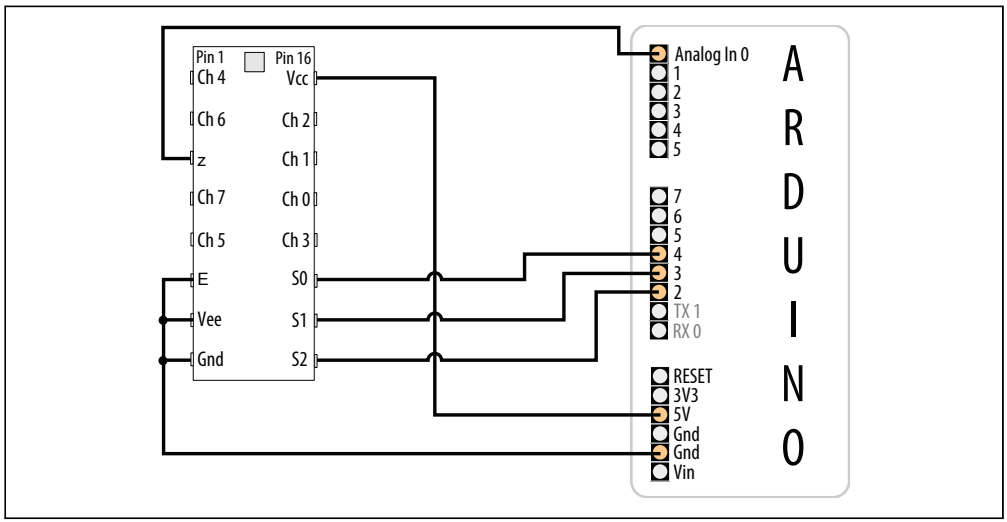


Abbildung 5-8: An Arduino angeschlossener 4051-Multiplexer

Diskussion

Analogmultiplexer sind digital gesteuerte Analogschalter. Der 4051 wählt über drei Selektorpins (S0, S1 und S2) einen von acht Eingängen aus. Für die drei Selektorpins gibt es acht verschiedene Wertekombinationen. Der Sketch wählt nacheinander jedes mögliche Bitmuster aus (siehe Tabelle 5-3).

Tabelle 5-3: Wahrheitstabelle für 4051-Multiplexer

Selektorpins			Eingang
S2	S1	S0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Vielleicht erkennen Sie das Muster in Tabelle 5-3 als Binärdarstellung der Dezimalwerte 0 bis 7.

Im obigen Sketch ist `getValue()` die Funktion, die die richtigen Selektor-Bits für den gewählten Kanal mit `digitalWrite(pin, isBitSet)` setzt und den analogen Wert des gewählten 4051-Eingangs mittels `analogRead(analogPin)` einliest. Der das Bitmuster erzeugende Code nutzt die fest eingebaute Funktion `bitRead` (siehe Rezept 3.12).

Denken Sie daran, die Masse der abzufragenden Bauelemente mit der Masse des 4051 und des Arduino zu verbinden (siehe Abbildung 5-8).

Beachten Sie, dass diese Technik die acht Eingänge nacheinander auswählt und abfragt, d.h., das Lesen eines Eingangs braucht im Vergleich zum direkten Einlesen über `analogRead` mehr Zeit. Wenn Sie acht Eingänge einlesen, brauchen Sie achtmal mehr Zeit, um jeden Eingang einzulesen. Diese Methode ist daher für Eingänge, deren Werte sich sehr schnell ändern, möglicherweise ungeeignet.

Siehe auch

Arduino Playground-Tutorial zum 4051: *http://www.arduino.cc/playground/Learning/4051*

CD4051-Datenblatt: *http://www.fairchildsemi.com/ds/CD%2FCD4052BC.pdf*

Datenblatt zum Analog/Digital-MUX-Breakout-Board: *http://www.nkcelectronics.com/analogdigital-mux-breakout.html*

5.9 Spannungen von bis zu 5V messen

Problem

Sie wollen eine Spannung zwischen 0 und 5 Volt messen und ausgeben. Beispielsweise wollen Sie die Spannung einer einzelnen 1,5-V-Zelle über den seriellen Monitor ausgeben.

Lösung

Verwenden Sie `AnalogRead`, um die Spannung an einem Analogpin zu messen. Sie können den Messwert in eine Spannung umwandeln, indem Sie das Verhältnis des Wertes zur Referenzspannung (5 Volt) ermitteln, wie in Abbildung 5-9 zu sehen.

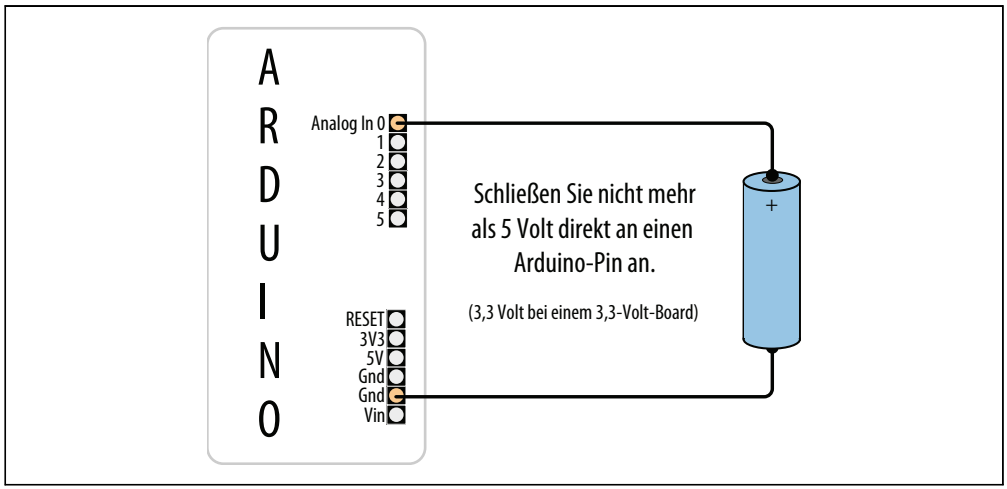


Abbildung 5-9: Spannungsmessung bis 5 Volt mit 5V-Board

Die einfachste Lösung nutzt eine Fließkomma-Berechnung zur Ausgabe der Spannung. Der folgende Beispiel-Sketch berechnet und gibt das Verhältnis als Spannung aus:

```
/*
 * Display5vOrless Sketch
 * Gibt die Spannung am Analogpin über den seriellen Port aus
 * Warnung - schließen Sie nicht mehr als 5V direkt an den Arduino-Pin an.
 */

const float referenceVolts = 5.0; // Referenzspannung eines 5-Volt-Boards
const int batteryPin = 0;      // Batterie ist mit Analogpin 0 verbunden

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

void loop()
{
```

```
  int val = analogRead(batteryPin);  // Wert vom Sensor einlesen
  float volts = (val / 1023.0) * referenceVolts;  // Verhältnis berechnen
  Serial.println(volts); // und Wert in Volt ausgeben
}
```

Die Formel lautet: Volt = (analoger Messwert / analoge Schritte) × Referenzspannung

Die Ausgabe eines Fließkommawerts über den seriellen Port mit `println` formatiert den Wert auf zwei Dezimalstellen genau.

Bei einem 3,3-V-Board nehmen Sie die folgende Änderung vor:

```
const int referenceVolts = 3.3;
```

Fließkommazahlen benötigen sehr viel Speicher. Wenn Ihr Sketch nicht auch an anderer Stelle Fließkommazahlen nutzt, ist es daher effizienter, mit ganzen Zahlen zu arbeiten. Der folgende Code sieht auf den ersten Blick vielleicht etwas seltsam aus, aber da `analogRead` den Wert `1023` für 5 Volt zurückgibt, werden die 5 Volt durch 1023 geteilt. In Millivolt entspricht das 5000 durch 1023.

Der folgende Code gibt den Wert in Millivolt aus:

```
const int  batteryPin = 0;

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

void loop()
{
 long val = analogRead(batteryPin);  // Wert vom Sensor einlesen -
                    // Hinweis: val ist ein long int
 Serial.println( (val * (500000/1023)) / 100);  // Wert in Millivolt ausgeben
}
```

Der folgende Code gibt den Wert mit einem Dezimalkomma aus, d.h., er gibt `1,5` aus, wenn die Spannung 1,5 Volt beträgt:

```
const int batteryPin = 0;

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

void loop()
{
 int val = analogRead(batteryPin); // Wert von Sensor einlesen
 long mv =  (val * (500000/1023L)) / 100; // Wert in Millivolt berechnen
 Serial.print(mv/1000); // Integerwert der Spanung ausgeben
 Serial.print(',');
 int fraction = (mv % 1000); // Nachkommastelle(n) berechnen
 if (fraction == 0)
```

```
  Serial.print("000");    // Drei Nullen ausgeben
 else if (fraction < 10)   // Bruch < 10
  Serial.print("00");     // Zwei Nullen ausgeben
 else if (fraction < 100)
  Serial.print("0");
 Serial.println(fraction); // Nachkommastelle(n) ausgeben
}
```

Bei einem 3,3-V-Board müssen Sie `(1023/5)` in `(int)(1023/3.3)` ändern.

Diskussion

Die `analogRead()`-Funktion gibt einen Wert zurück, der proportional zum Verhältnis der gemessenen Spannung zur Referenzspannung (5 Volt) ist. Um Fließkommazahlen zu vermeiden, gleichzeitig aber die Genauigkeit beizubehalten, arbeitet der Code nicht mit Volt, sondern mit Millivolt (1000 Millivolt sind 1 Volt). Da der Wert `1023` für 5000 Millivolt steht, repräsentiert jeder Schritt 5000 durch 1023 Millivolt (also 4,89 Millivolt).

Sie werden neben 1023 auch 1024 bei der Umwandlung von `analogRead`-Werten in Millivolt sehen. 1024 wird häufig von Ingenieuren verwendet, da es 1024 mögliche Werte zwischen 0 und 1023 gibt. Andererseits empfinden einige die 1023 als intuitiver, weil dass der höchstmögliche Wert ist. In der Praxis ist die Hardware-Ungenauigkeit aber größer als der Unterschied in den Berechnungen. Verwenden Sie also einfach den Wert, der Ihnen angenehmer ist.

Um das Dezimalkomma zu eliminieren, wird der Wert mit 100 multipliziert. Mit anderen Worten liefert 5000 Millivolt mal 100 durch1023 den Wert in Millivolt mal 100. Eine Division durch 100 ergibt den Wert in Millivolt. Durch die Multiplikation der Nachkommastellen mit 100 können wir den Compiler die Berechnung mit ganzen Zahlen durchführen lassen. Wenn Ihnen das zu umständlich ist, können Sie aber bei der langsameren und speicherhungrigeren Fließkomma-Methode bleiben.

Diese Lösung geht davon aus, dass Sie einen Standard-Arduino mit 5 Volt verwenden. Wenn Sie ein 3,3-V-Board nutzen, liegt die maximale Spannung, die Sie ohne Spannungsteiler messen können, bei 3,3 Volt – siehe Rezept 5.11.

5.10 Auf Spannungsänderungen reagieren

Problem

Sie wollen eine oder mehrere Spannungen überwachen und reagieren, wenn sie einen bestimmten Schwellwert übersteigen oder unterschreiten. Zum Beispiel könnten Sie eine LED blinken lassen, wenn die Batterie leer wird. Sie könnte etwa langsam anfangen zu blinken, wenn ein Schwellwert unterschritten wird, und immer schneller blinken, während die Spannung weiter fällt.

Lösung

Sie können die Verschaltung aus Abbildung 5-7 in Rezept 5.9 verwenden, doch hier überprüfen wir, ob der Wert von `analogRead` unter einen Schwellwert fällt. In diesem Beispiel beginnt die LED bei 1,2 Volt zu blinken und die Blinkgeschwindigkeit erhöht sich, während die Spannung weiter unter den Schwellwert sinkt. Fällt die Spannung unter einen zweiten Schwellwert, bleibt die LED an:

```
/*
 RespondingToChanges Sketch
 Bei niedriger Spannung LED blinken lassen
*/

long warningThreshold  = 1200; // Warn-Schwellwert - LED blinkt
long criticalThreshold = 1000; // Kritischer Spannungspegel - LED bleibt an

const int batteryPin = 0;
const int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
 int val = analogRead(batteryPin);   // Wert vom Sensor einlesen
 if( val < (warningThreshold * 1023L)/5000) {
   // Das auf die Zahl folgende L macht sie zu einem 32-Bit-Wert
   flash(val) ;
 }
}

// Diese Funktion läßt die LED blinken
// Die Ein/Aus-Dauer wird durch den übergebenn Prozentwert bestimmt
void flash(int percent)
{
 digitalWrite(ledPin, HIGH);
 delay(percent + 1);
 digitalWrite(ledPin, LOW);
 delay(100 - percent );  // Dauer == 0?
}
```

Diskussion

Die im Sketch hervorgehobene Zeile berechnet das Verhältnis der über den Analogport eingelesenen Spannung zum Schwellwert. Bei einem Warn-Schwellwert von 1 Volt und einer Referenzspannung von 5 Volt wollen Sie beispielsweise wissen, wann der Messwert ein Fünftel der Referenzspannung erreicht. Der Ausdruck `1023L` weist den Compiler an, mit long-Werten (32-Bit-Integer; siehe Rezept 2.2) zu arbeiten. Der Compiler macht daher alle Variablen dieses Ausdrucks zu long-Werten, um den Überlauf eines `int` (ein normaler 16-Bit-Integer) zu verhindern.

Sie können direkt mit den von `analogRead` gelieferten Werten (zwischen 0 und 1023) arbeiten, Sie können aber auch mit den tatsächlichen Spannungen arbeiten, für die sie stehen (siehe Rezept 5.7). Wenn Sie (wie bei diesem Rezept) keine Spannungen ausgeben müssen, ist es einfacher und effektiver, direkt die Werte von `analogRead` zu verwenden.

5.11 Spannungen über 5V messen (Spannungsteiler)

Problem

Sie wollen Spannungen über 5 Volt messen. Zum Beispiel könnten Sie die Spannung einer 9-V-Batterie ausgeben wollen und eine Warn-LED blinken lassen, wenn die Spannung unter eine kritische Grenze fällt.

Lösung

Die Lösung ähnelt der in Rezept 5.9, die Spannung wird hier aber über einen Spannungsteiler abgegriffen (siehe Abbildung 5-10). Bei Spannungen bis zu 10 Volt können Sie zwei 4,7-K-Ohm-Widerstände verwenden. Für höhere Spannungen können Sie die benötigten Widerstände aus Tabelle 5-4 ablesen.

Tabelle 5-4: Widerstandswerte

Max. Spannung	R1	R2	Berechnung R2/(R1 + R2)	Wert von `resistorFactor`
5	Ohne (+V an Analogpin)	Ohne (Masse an Masse)	None	1023
10	1K	1K	1(1 + 1)	511
15	2K	1K	1(2 + 1)	341
20	3K	1K	1(3 + 1)	255
30	4K (3,9K)	1K	1(4 + 1)	170

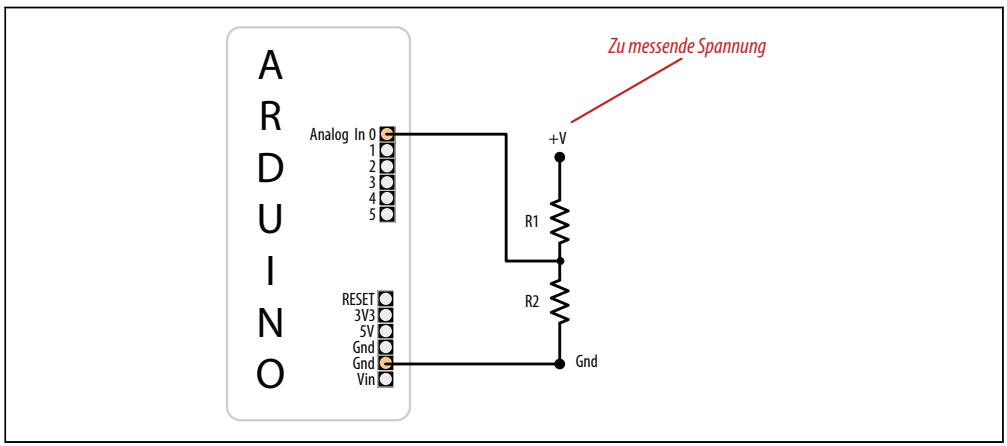


Abbildung 5-10: Spannungsteiler für Spannungsmessungen über 5 Volt

Wählen Sie die Zeile mit der höchsten Spannung, die Sie messen müssen, und suchen Sie sich die beiden Widerstandswerte heraus:

```
/*
 DisplayMoreThan5V Sketch
 Gibt die Spannung am Analogpin über den seriellen Port aus
 Schließen Sie nicht mehr als 5 Volt direkt an einen Arduino-Pin an.
*/

const float referenceVolts = 5;     // Standard-Referenzspannung eines 5-V-Boards
//const float referenceVolts = 3.3; // Nutzen Sie diesen Wert bei einem 3,3-V-Board

const float R1 = 1000; // Wert für max. Spannung von 10 Volt
const float R2 = 1000;
// Wird durch Spannungsteiler-Widerstände bestimmt, siehe Text
const float resistorFactor = 1023.0 / (R2/(R1 + R2));
const int batteryPin = 0;      // +V der Battere ist mit Analogpin 0 verbunden

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

void loop()
{
  int val = analogRead(batteryPin); // Wert vom Sensor einlesen
  float volts = (val / resistorFactor) * referenceVolts ; // Verhältnis berechnen
  Serial.println(volts); // Wert in Volt ausgeben
}
```

Diskussion

Wie bei den vorangegangenen Analog-Rezepten nutzt dieses die Tatsache, dass der gemessene `analogRead`-Wert im Verhältnis zur Referenzspannung steht. Da die tatsächliche Spannung aber durch die beiden Widerstände geteilt wurde, muss der Wert von `analogRead` multipliziert werden, um die tatsächliche Spannung zu ermitteln. Der Code ähnelt dem aus Rezept 5.7, doch der Wert für `resistorFactor` wird basierend auf den Spannungsteiler-Widerständen aus Tabelle 5-4 gewählt:

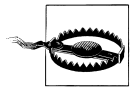
```
const int resistorFactor = 511;   // Durch Spannungsteiler-Widerstände bestimmt, siehe Tabelle
5-3
```

Der vom Analogpin eingelesene Wert wird nicht durch 1023 dividiert, sondern durch die Werte der Vorschaltwiderstände:

```
float volts = (val / resistorFactor) * referenceVolts ;  // Verhältnis berechnen
```

Die zum Aufbau der Tabelle genutzte Berechnung basiert auf der folgenden Formel: Die Ausgangsspannung entspricht der Eingangsspannung mal R2 geteilt durch die Summe von R1 und R2. Im Beispiel mit zwei gleichen Widerstandswerten, bei dem die Spannung einer 9-V-Batterie halbiert wird, hat `resistorFactor` den Wert 511 (die Hälfte von 1023), d.h., der Wert der `volts`-Variablen ist doppelt so hoch wie die am Eingangspin anliegende

Spannung. Mit den Widerständen für 10 Volt liegt der Analogwert der 9-V-Batterie bei ungefähr 920.



Liegen mehr als 5 Volt an einem Pin an, kann das den Pin beschädigen oder sogar den Chip zerstören. Prüfen Sie also genau, ob Sie die richtigen Widerstandswerte gewählt und sie richtig angeschlossen haben. Wenn Sie ein Multimeter besitzen, messen Sie die Spannung, bevor Sie etwas mit dem Arduino-Pin verbinden.

KAPITEL 6

Werte von Sensoren einlesen

6.0 Einführung

Werte über Sensoren einzulesen und zu verarbeiten, ermöglicht es dem Arduino, auf die Welt um ihn herum zu reagieren oder über sie zu informieren. Dieses Kapitel enthält einfache und praktische Beispiele, wie man die beliebtesten Eingabegeräte und Sensoren nutzt. Die Schaltdiagramme zeigen, wie man die Bauelemente anbindet und mit Strom versorgt, während die Code-Beispiele zeigen, wie man die Sensordaten verarbeitet.

Sensoren reagieren auf Ereignisse der physikalischen Welt und wandeln sie in ein elektrisches Signal um, dass der Arduino über einen Eingangspin einlesen kann. Die Natur des elektrischen Signals, die der Sensor zur Verfügung stellt, hängt von der Art des Sensors ab und davon, wie viele Daten er übertragen muss. Einige Sensoren (wie Photowiderstände und piezoelektrische Sensoren) bestehen aus einem Material, das seine elektrischen Eigenschaften als Reaktion auf physikalische Änderungen verändert. Andere sind ausgefeilte elektronische Module mit eigenem Mikrocontroller, die Informationen verarbeiten, bevor sie Daten an den Arduino übergeben.

Sensoren nutzen die folgenden Methoden, um Informationen bereitzustellen:

Digital AN/AUS
: Einige Bauelemente, wie der Tilt-Sensor in Rezept 6.1 und der Bewegungssensor in Rezept 6.3, schalten einfach eine Spannung ein oder aus. Sie können diese Sensoren so verarbeiten, wie in den Taster-Rezepten in Kapitel 5.

Analog
: Andere Sensoren liefern ein Analogsignal zurück (eine Spannung proportional zum abgerufenen Wert, wie etwa Temperatur oder Lichtstärke). Die Beispiele zur Bestimmung von Lichtstärke (Rezept 6.2), Bewegung (Rezepte 6.1 und 6.3), Vibration (Rezept 6.6), Sound (Rezept 6.7) und Beschleunigung (Rezept 6.18) zeigen, wie Analogsensoren verwendet werden können. Alle nutzen die Funktion `analogRead`, die in Kapitel 5 erläutert wurde.

Impulsbreite
: Abstandssensoren wie der PING in Rezept 6.4 stellen die Daten in Form eines Impulses zur Verfügung, dessen Länge proportional zum Abstand ist. Solche Sensoren nutzende Anwendungen messen die Dauer des Impulses mit Hilfe der `pulseIn`-Funktion.

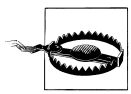
Seriell
: Einige Sensoren liefern Werte über ein serielles Protokoll zurück. Der RFID-Leser in Rezept 6.9 und das GPS in Rezept 6.14, kommunizieren beispielsweise über den seriellen Port mit dem Arduino (mehr über den seriellen Port erfahren Sie in Kapitel 4). Die meisten Arduinos besitzen nur einen seriellen Hardware-Port. Rezept 6.14 zeigt, wie Sie zusätzliche Software-Ports einrichten können, wenn Sie mit mehreren seriellen Sensoren arbeiten oder der Hardware-Port für eine andere Aufgabe genutzt wird.

Synchrone Protokolle: I2C und SPI
: Die digitalen Standards I2C und SPI wurden für Mikrocontroller wie den Arduino entwickelt, um sich mit externen Sensoren und Modulen unterhalten zu können. Rezept 6.16 zeigt, wie man ein Kompass-Modul über synchrone Digitalsignale anbindet. Diese Protokolle werden ausgiebig von Sensoren, Aktuatoren und Peripheriegeräten genutzt und werden detailliert in Kapitel 13 behandelt.

Es gibt andere Eingabegeräte, die Sie nutzen können. Dabei handelt es sich um Sensoren enthaltende Konsumartikel, die als eigenständige Geräte verkauft werden. Zu den Beispielen aus diesem Kapitel gehört eine PS/2-Maus und ein Playstation-Controller. Diese Geräte können sehr praktisch sein. Sie integrieren Sensoren in einem robusten und ergonomischen Gehäuse. Darüber hinaus sind sie billig (oft billiger, als die Kosten für die darin enthaltenen Sensoren), weil es sich um Massenprodukte handelt. Wahrscheinlich liegen einige dieser Geräte bei Ihnen herum.

Wenn Sie mit einem Gerät arbeiten, das von keinem Rezept behandelt wird, können Sie möglicherweise ein anderes Rezept wiederverwerten, das ähnliche Daten verarbeitet. Informationen zu den Ausgangssignalen des Sensors erhalten Sie üblicherweise von dem Unternehmen, bei dem Sie ihn gekauft haben, oder alternativ auf dem Datenblatt (googeln Sie einfach nach der Artikelnummer oder Beschreibung).

Datenblätter richten sich an Ingenieure, die Produkte entwickeln, die später auch produziert werden sollen. Üblicherweise enthalten Sie viel mehr Informationen, als Sie brauchen, um Ihr Projekt ans Laufen zu bekommen. Die Informationen zu den Ausgangssignalen finden Sie üblicherweise in einem Abschnitt über Datenformate, Interfaces, Ausgangssignale oder so ähnlich. Vergessen Sie die maximale Spannung nicht (üblicherweise in einem Abschnitt namens »Absolute Maximum Ratings« oder »Absolute Grenzdaten«), um die Komponenten nicht zu beschädigen.



Für maximale Spannungen von 3,3V gedachte Sensoren können zerstört werden, wenn man sie mit 5 Volt verbindet. Überprüfen Sie also die maximale Spannung, bevor Sie das Bauelement anschließen.

Das Einlesen von Sensordaten in einer chaotischen, analogen Welt ist eine Mischung aus Wissenschaft, Kunst und Beharrlichkeit. Möglicherweise brauchen Sie einigen Einfallsreichtum und viele Fehlversuche, bis Sie ein korrektes Ergebnis erhalten. Ein typisches Problem besteht darin, dass der Sensor Ihnen nur mitteilt, dass eine physikalische Bedingung eingetreten ist, aber nicht, wer sie verursacht hat. Den Sensor in den richtigen Kontext zu bringen (Lage, Distanz, Orientierung) und sich dabei auf die Dinge zu beschränken, die man wirklich braucht, kommt erst mit der Erfahrung.

Ein weiterer Aspekt ist die Trennung des gewünschten Signals von Hintergrundgeräuschen ; Rezept 6.6 zeigt, wie man einen Schwellwert nutzt, um zu erkennen, ob ein Signal einen bestimmten Pegel überschritten hat. Rezept 6.7 zeigt, wie Sie den Durchschnitt einer Messwertreihe nutzen können, um Geräuschspitzen herauszufiltern.

Siehe auch

Informationen zum Anschluss elektrischer Komponenten finden Sie in *Make: Electronics* von Charles Platt (Make).

Weiterführende Informationen zum Einlesen von Analogwerten über Sensoren finden Sie in der Einführung zu Kapitel 5 und Rezept 5.6.

6.1 Movement erkennen

Problem

Sie wollen erkennen, ob etwas bewegt, geneigt oder geschüttelt wird.

Lösung

Der nachfolgende Sketch nutzt einen Schalter, der einen Kreis schließt, wenn er gekippt wird. Einen solchen Schalter bezeichnet man als *Neigungssensor* (engl. tilt sensor). Die Schalter-Rezepte in Kapitel 5 (Die Rezepte 5.1 und 5.2) funktionieren auch, wenn man den Schalter/Taster durch einen Neigungssensor ersetzt.

Der Sketch (die Schaltung sehen Sie in Abbildung 6-1) schaltet eine LED an Pin 11 ein, wenn der Neigungssensor in die eine Richtung, und die LED an Pin 12, wenn er in die andere Richtung gekippt wird:

```
/*
tilt Sketch

 Ein Neigungssensor an Pin 2 schaltet eine der
 LEDs an den Pins 11 und 12 ein, je nachdem,
 in welche Richtung der Sensor geneigt wird.
*/

const int tiltSensorPin = 2;       //Pin für Neigungssensor
const int firstLEDPin = 11;       //Pin für erste LED
const int secondLEDPin = 12;       //Pin für zweite LED
```

```
void setup()
{
 pinMode (tiltSensorPin, INPUT);     //Der Code liest diesen Pin
 digitalWrite (tiltSensorPin, HIGH);   //und nutzt einen Pullup-Widerstand

 pinMode (firstLEDPin, OUTPUT);      //Der Code schreibt an diesen
 pinMode (secondLEDPin, OUTPUT);     //und diesen Pin
}

void loop()
{
 if (digitalRead(tiltSensorPin)){    //Ist der Pin "HIGH",
  digitalWrite(firstLEDPin, HIGH);   //erste LED ein-
  digitalWrite(secondLEDPin, LOW);   //und zweite ausschalten
 }
 else{                   //Wenn nicht, kehren
  digitalWrite(firstLEDPin, LOW);    //wir die Sache um
  digitalWrite(secondLEDPin, HIGH);
 }
}
```

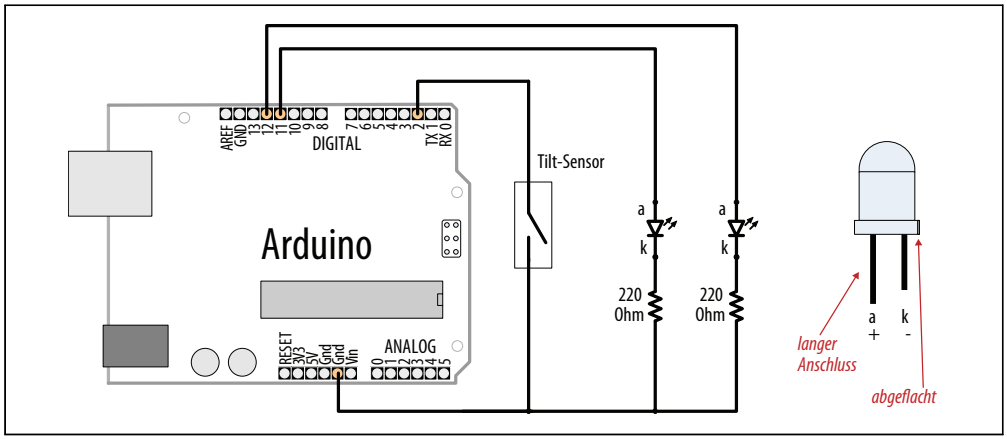


Abbildung 6-1: Neigungssensor und LEDs

Diskussion

Die am weitesten verbreiteten Neigungssensoren bestehen aus einer Kugel in einem Gehäuse und Kontakten auf einer Seite. Wird das Gehäuse geneigt, rollt die Kugel von den Kontakten weg, und die Verbindung wird unterbrochen. Wird es in die andere Richtung geneigt, berührt die Kugel die Kontakte, und der Kreis wird geschlossen. Markierungen zeigen, wie der Sensor auszurichten ist. Neigungssensoren können kleine Bewegungen von 5 bis 10 Grad erkennen. Wenn Sie den Sensor so ausrichten, dass die Kugel direkt über (oder unter) den Kontakten liegt, ändert sich der LED-Zustand nur dann, wenn man ihn umdreht. Auf diese Weise können Sie erkennen, ob etwas auf der Ober- oder Unterseite liegt.

Wenn Sie wissen wollen, ob etwas geschüttelt wird, müssen Sie prüfen, wann sich der Zustand des Sensors zuletzt geändert hat (in unserem Beispiel prüfen wir nur, ob der Schalter offen oder geschlossen war). Ändert sich der Zustand innerhalb einer von Ihnen als signifikant festgelegten Zeitspanne nicht, wird das Objekt auch nicht geschüttelt. Die Veränderung der Ausrichtung des Neigungssensors bestimmt auch, wie heftig Sie ihn schütteln müssen, um ihn auszulösen. Der folgende Code schaltet eine LED ein, wenn der Sensor geschüttelt wird:

```
/*
 shaken Sketch
 Neigungssensor an Pin 2
 LED an Pin 13
*/

const int tiltSensorPin = 2;
const int ledPin = 13;
int tiltSensorPreviousValue = 0;
int tiltSensorCurrentValue = 0;
long lastTimeMoved = 0;
int shakeTime=50;

void setup()
{
 pinMode (tiltSensorPin, INPUT);
 digitalWrite (tiltSensorPin, HIGH);
 pinMode (ledPin, OUTPUT);
}

void loop()
{
 tiltSensorCurrentValue=digitalRead(tiltSensorPin);
 if (tiltSensorPreviousValue != tiltSensorCurrentValue){
  lastTimeMoved = millis();
  tiltSensorPreviousValue = tiltSensorCurrentValue;
 }

 if (millis() - lastTimeMoved < shakeTime){
  digitalWrite(ledPin, HIGH);
 }
 else{
  digitalWrite(ledPin, LOW);
 }
}
```

Viele mechanische Sensorschalter können auf ähnliche Weise genutzt werden. Ein Schwimmerschalter schaltet sich ein, wenn der Wasserpegel in einem Behälter eine gewisse Höhe erreicht (ähnlich dem Schwimmerhahn in einer Wasserspülung). Mit einer Druckleiste, wie man sie an den Eingängen von Geschäften findet, können Sie erkennen, ob jemand auf ihr steht. Wenn Ihr Sensor ein digitales Signal ein- oder ausschaltet, sollte eine Lösung wie in diesen Sketches geeignet sein.

Siehe auch

Hintergrundinformationen zur Verwendung von Schaltern mit dem Arduino finden Sie in Kapitel 5.

Rezept 12.2 enthält weitere Informationen zur Verwendung von `millis` zur Bestimmung von Laufzeiten.

6.2 Licht messen

Problem

Sie wollen Veränderungen der Lichtstärke messen. Sie könnten erkennen wollen, ob etwas vor einem Lichtsensor vorbeigeht oder ob ein Raum zu dunkel ist.

Lösung

Die einfachste Möglichkeit, die Lichtstärke zu bestimmen, bieten lichtempfindliche Widerstände. Ein solcher Widerstand verändert seinen Widerstandswert, wenn sich die Lichtstärke ändert. Schließt man ihn wie in Abbildung 6-2 zu sehen an, erzeugt er eine veränderliche Spannung, die der Arduino über einen Analogpin abgreifen kann.

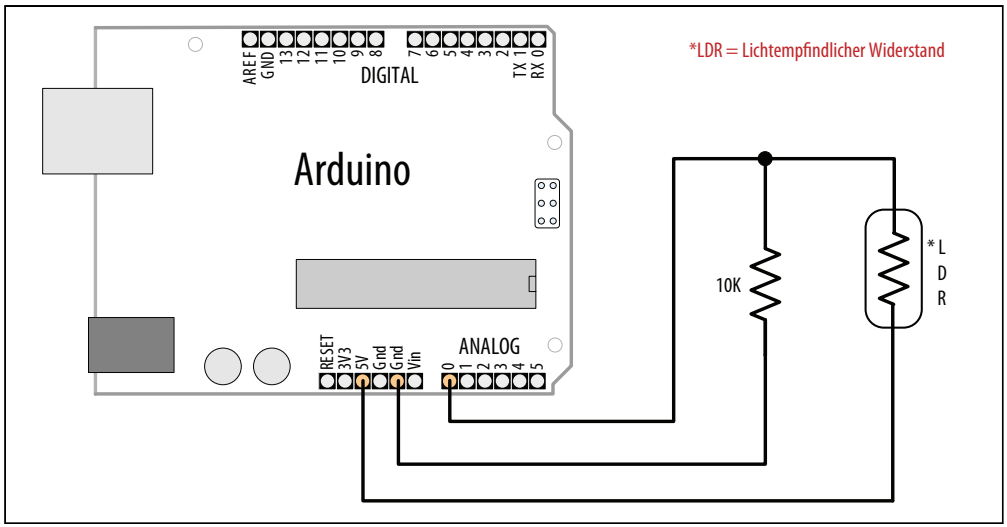


Abbildung 6-2: Anschluss eines lichtempfindlichen Widerstands

Der Sketch für dieses Rezept ist einfach:

```
const int ledPin =  13;   // Mit Pin 13 verbundene LED
const int sensorPin = 0;  // Mit Analogeingang 0 verbundener Sensor

void setup()
{
```

```
 pinMode(ledPin, OUTPUT); // LED-Pin als Ausgang schalten
}

void loop()
{
 int rate = analogRead(sensorPin);   // Analogeingang einlesen
 digitalWrite(ledPin, HIGH);  // LED einschalten
 delay(rate);           // Dauer abhängig von Lichtstärke
 digitalWrite(ledPin, LOW);   // LED ausschalten?
 delay(rate);
}
```

Diskussion

Die Schaltung dieses Rezepts entspricht der Standard-Lösung für jeden Sensor, der seinen Widerstand basierend auf irgendeiner physikalischen Größe verändert (Hintergrundinformationen zur Handhabung von Analogsignalen finden Sie in Kapitel 5). Bei der Schaltung in Abbildung 6-2 verändert sich die Spannung an Analogpin 0, wenn sich der Widerstand des lichtempfindlichen Widerstands aufgrund variierender Lichtstärke ändert.

Eine Schaltung wie diese nutzt nicht den gesamten möglichen Wertebereich des Analogeingangs (0 bis 1023), da die Spannungswerte nicht zwischen den vollen 0 bis 5 Volt liegen. Das liegt daran, dass es bei jedem Widerstand einen Spannungsabfall gibt, so dass die Spannung an dieser Stelle nie die Grenzen der Spannungsversorgung erreicht. Wenn Sie solche Sensoren nutzen, müssen Sie also die tatsächlichen Werte ermitteln, die der Sensor in Ihrem Fall zurückliefert. Dann müssen Sie bestimmen, wie Sie diese Werte in die Werte umwandeln, die Sie für Ihre Steuerung benötigen (was auch immer Sie steuern). Details zur Anpassung von Wertebereichen finden Sie in Rezept 5.7.

Ein lichtempfindlicher Widerstand ist ein einfacher sog. *ohmscher Sensor*. Eine Reihe ohmscher Sensoren reagieren auf Änderungen verschiedener physikalischer Eigenschaften. Ähnliche Schaltungen werden auch mit anderen Arten einfacher ohmscher Sensoren funktionieren, auch wenn Sie möglicherweise den Widerstand an den jeweiligen Sensor anpassen müssen.

Die Wahl des besten Widerstandswerts hängt vom verwendeten lichtempfindlichen Widerstand ab und davon, mit welchen Lichtstärken gearbeitet wird. Ingenieure würden ein Lichtmessgerät verwenden und das Datenblatt des lichtempfindlichen Widerstands konsultieren, doch wenn Sie ein Multimeter besitzen, können Sie den Widerstand des lichtempfindlichen Widerstands bei einer Lichtstärke messen, die ungefähr in der Mitte des Helligkeitsbereichs liegt, mit dem Sie arbeiten wollen. Notieren Sie den Wert und wählen Sie den für diesen Wert am besten geeigneten Widerstand.

Siehe auch

Dieser Sketch wurde in Rezept 1.6 vorgestellt. Dort finden Sie weitere Informationen und Variationen dieses Sketches.

6.3 Motion erkennen (Passive Infrarot-Detektoren integrieren)

Problem

Sie wollen erkennen, wenn sich jemand in der Nähe eines Sensors bewegt.

Lösung

Nutzen Sie einen Bewegungssensor, z.B. einen passiven Infrarot-Sensor (PIR), der Werte an einem Digitalpin ändert, wenn sich jemand in der Nähe bewegt.

Sensoren wie der SparkFun PIR Motion Sensor (SEN-08630) und der Parallax PIR Sensor (555-28027) können einfach an Arduino-Pins angeschlossen werden, wie in Abbildung 6-3 zu sehen.

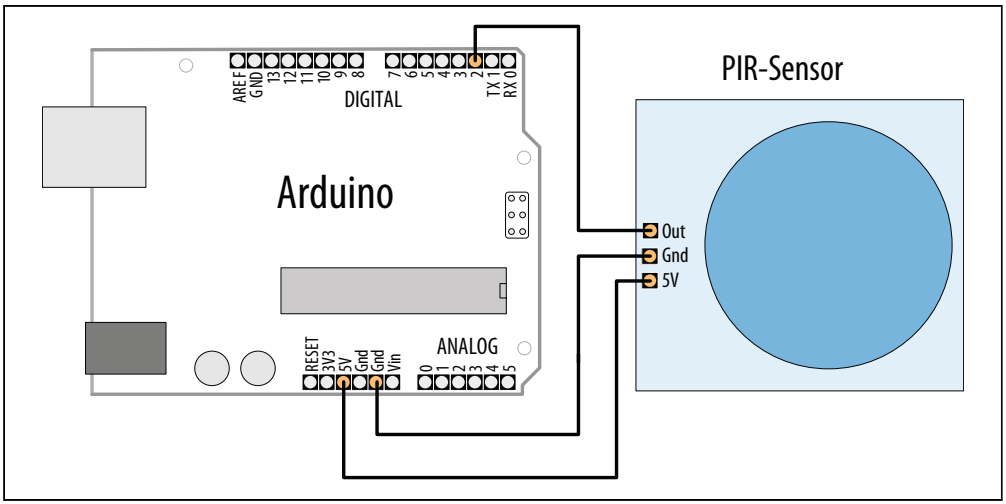


Abbildung 6-3: Anschluss eines PIR-Bewegungssensors

Suchen Sie sich die richtigen Pins aus dem Datenblatt für Ihren Sensor heraus. Beim Parallax-Sensor sind die Pins mit »OUT« »-« und »+« (für Ausgang, Masse und +5V) gekennzeichnet. Der SparkFun-Sensor verwendet die Bezeichnungen »Alarm«, »GND« und »DC« (für Ausgang, Masse und +5V).

Der folgende Sketch schaltet die LED an Arduino-Pin 13 ein, wenn der Sensor eine Bewegung erkennt:

```
/*
  PIR Sketch
  Passiver Infrarot-Bewegungssensor an Pin 2
  schaltet die LED an Pin 13 ein
*/
```

```
const int ledPin = 13;       // Pin für LED
const int inputPin = 2;       // Pin für PIR-Sensor

void setup() {
 pinMode(ledPin, OUTPUT);     // LED ist Ausgang
 pinMode(inputPin, INPUT);     // Sensor ist Eingang
}

void loop(){
 int val = digitalRead(inputPin); // Eingangswert einlesen
 if (val == HIGH)           // Ist der Eingang HIGH
 {
  digitalWrite(ledPin, HIGH);   // Bewegung erkannt, LED einschalten
  delay(500);
  digitalWrite(ledPin, LOW);    // LED ausschalten
 }
}
```

Diskussion

Der Code ähnelt den Taster-Beispielen aus Kapitel 5. Das liegt daran, dass sich der Sensor wie ein Schalter verhält, wenn eine Bewegung erkannt wird. Es gibt verschiedene Arten von PIR-Sensoren, und Sie sollten sich die Daten zu dem von Ihnen angeschlossenen ansehen.

Einige Sensoren, wie der Parallax, besitzen Steckbrücken (Jumper), die bestimmen, wie sich der Ausgang verhält, wenn eine Bewegung erkannt wird. In einem Modus bleibt der Ausgang `HIGH`, wenn Bewegung erkannt wird, oder er kann so gesetzt werden, dass er beim Auslösen kurz auf `HIGH` und dann wieder auf `LOW` geht. Der Beispiel-Sketch in diesem Rezept funktioniert mit beiden Modi.

Andere Sensoren können `LOW` zurückliefern, wenn sie eine Bewegung erkennen. Geht der Ausgangspin Ihres Sensors auf `LOW`, wenn eine Bewegung erkannt wird, ändern Sie die Zeile, die den Eingangswert überprüft, wie folgt ab:

```
if (val == LOW)            // Bewegung, wenn Eingang LOW
```

PIR-Sensoren gibt es in unterschiedlichen Arten und für verschiedene Distanzen und Winkel. Durch sorgfältige Auswahl und Positionierung können Sie die Bewegungserkennung auf einen Teil des Raums einschränken.

PIR-Sensoren reagieren auf Wärme und können nicht nur auf Menschen, sondern auch auf Tiere (Katzen, Hunde etc.) und andere Wärmequellen reagieren.

6.4 Abstände messen

Problem

Sie möchten einen Abstand ermitteln, etwa zu einer Wand oder zu jemandem, der sich zum Arduino hin bewegt.

Lösung

Dieses Rezept nutzt den beliebten Parallax PING))) Ultraschall-Sensor. Er kann den Abstand eines Objekts in einem Bereich von 2 Zentimetern bis zu 3 Metern messen. Der Sketch gibt den Abstand über den seriellen Monitor aus und lässt eine LED schneller blinken, je näher das Objekt kommt. In Abbildung 6-4 sehen Sie das Schaltdiagramm:

```
/* Ping))) Sensor
 * Gibt den Abstand eines Ping)))-Sensors aus und ändert
 * die Blinkgeschwindigkeit einer LED in Abhängigkeit vom
 * Abstand.
 */


const int pingPin = 5;
const int ledPin  = 13; // LED-Pin

void setup()
{
 Serial.begin(9600);
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 int cm = ping(pingPin) ;
 Serial.println(cm);
 digitalWrite(ledPin, HIGH);
 delay(cm * 10 ); // Für jeden Zentimeter 10 Millisekunden hinzufügen
 digitalWrite(ledPin, LOW);
 delay( cm * 10);
}

// Folgender Code basiert auf http://www.arduino.cc/en/Tutorial/Ping
// Gibt den Abstand in cm zurück
int ping(int pingPin)
{
 // Variablen für ping-Dauer,
 // und Abstand in cm:
 long duration, cm;

 // Der PING))) wird durch einen HIGH-Impuls von 2 oder mehr Mikrosekunden angestoßen.
 // Wir setzen vorher einen kurzen LOW-Impuls an, um einen sauberen HIGH-Impuls sicherzustellen:
 pinMode(pingPin, OUTPUT);
 digitalWrite(pingPin, LOW);
 delayMicroseconds(2);
 digitalWrite(pingPin, HIGH);
```

```
 delayMicroseconds(5);
 digitalWrite(pingPin, LOW);

 pinMode(pingPin, INPUT);
 duration = pulseIn(pingPin, HIGH);

 // Zeit in Abstand umwandeln
 cm = microsecondsToCentimeters(duration);
 return cm ;
}

long microsecondsToCentimeters(long microseconds)
{
 // Die Schallgeschwindigkeit liegt bei 340 m/s oder 29 Mikrosekunden pro Zentimeter.
 // Der Ping läuft hin und zurück, d.h., um den Abstand des Objekts zu
 // ermitteln, verwenden wir die Hälfte der zurückgelegten Strecke.
 return microseconds / 29 / 2;
}
```

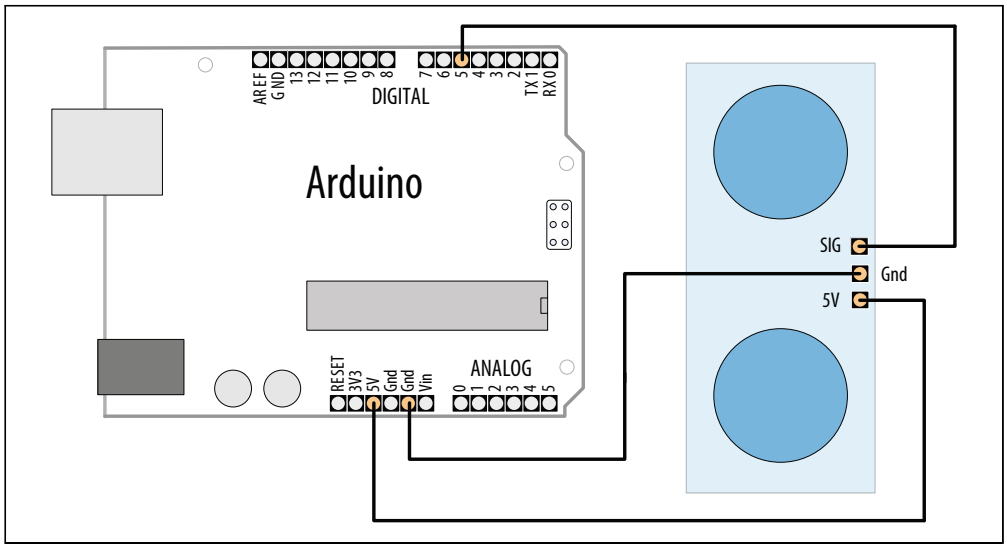


Abbildung 6-4: Anschluss des Ping)))-Sensors

Diskussion

Ultraschallsensoren messen die Zeit, die der Schall braucht, um von einem Objekt abzuprallen und zum Sensor zurückzukehren.

Der »ping«-Impuls wird erzeugt, wenn der `pingPin` für zwei Mikrosekunden auf `HIGH` geht. Der Sensor generiert dann einen Puls, der endet, wenn der Schall zurückkehrt. Die Länge dieses Impulses ist proportional zur Strecke, die der Schall zurückgelegt hat, und der Sketch verwendet die `pulseIn`-Funktion, um diese Dauer zu messen. Die Schallgeschwindigkeit beträgt 340 Meter pro Sekunde oder 29 Mikrosekunden pro Zentimeter. Die Formel für den Hin- und Rückweg lautet: Hin-und Rückweg = Mikrosekunden / 29

Die Formel für den eigentlichen Abstand in Zentimetern lautet also: Mikrosekunden / 29 / 2

Der MaxBotix EZ1 ist ein weiterer Ultraschallsensor, der zur Abstandsmessung genutzt werden kann. Er ist einfacher zu integrieren als der Ping))), weil er nicht »angepingt« werden muss. Er kann kontinuierlich Abstandsdaten liefern, entweder in Form einer Analogspannung oder proportional als Impulsbreite. Abbildung 6-5 zeigt, wie man ihn anschließt.

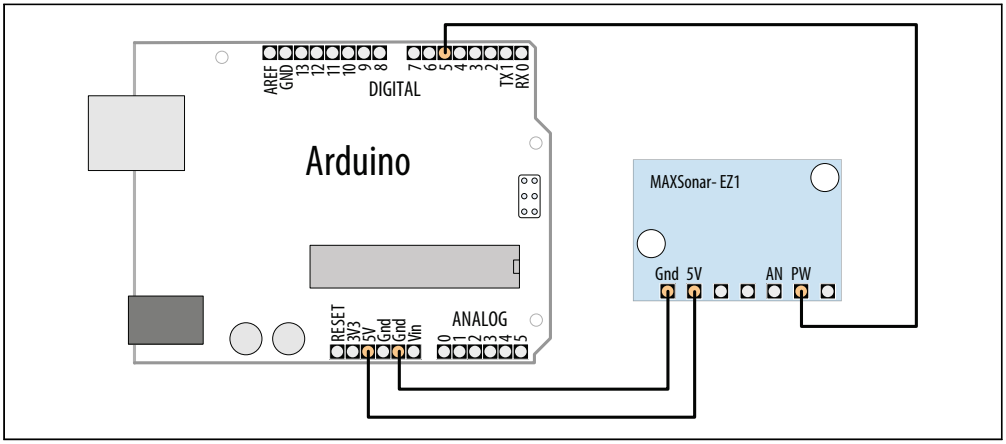


Abbildung 6-5: Anschluss des EZ1 PW-Ausgangs mit digitalem Eingangspin

Der nachfolgende Sketch nutzt den EZ1-PW-Ausgang (Impulsbreite), um eine ähnliche Ausgabe zu erzeugen wie im obigen Sketch:

```
/*
 * EZ1Rangefinder Distance Sensor
 * Gibt den Abstand eines EZ1-Sensors aus und ändert die
 * Blinkgeschwindigkeit einer LED in Abhängigkeit vom
 * Abstand.
 */

const int sensorPin = 5;
const int ledPin  = 13; // LED-Pin

long value = 0;
int cm = 0;
int inches = 0;

void setup()
{
 Serial.begin(9600);
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 value = pulseIn(sensorPin, HIGH) ;
```

```
  cm = value / 58;     // Impulsbreite ist 58 Mikrosekunden pro Zentimeter
  inches = value / 147;  // oder 147 Mikrosekunden pro Zoll
  Serial.print(cm);
  Serial.print(',');
  Serial.println(inches);

  digitalWrite(ledPin, HIGH);
  delay(cm * 10 ); // Für jeden Zentimeter 10 Millisekunden hinzufügen
  digitalWrite(ledPin, LOW);
  delay( cm * 10);

  delay(20);
 }
```

Der EZ1 wird über die entsprechenden Arduino-Pins mit +5V und Masse versorgt. Verbinden Sie den EZ1 PW-Pin mit dem Arduino-Digitalpin 5. Der Sketch misst die Breite des Impulses mit der `pulseIn`-Funktion. Die Breite des Impulses liegt bei 58 Mikrosekunden pro Zentimeter bzw. bei 147 Mikrosekunden pro Zoll.

Bei langen Anschlussleitungen müssen Sie am Sensor möglicherweise einen Kondensator zwischen +5V und Masse schalten, um die Spannungsversorgung zu stabilisieren. Bei fehlerhaften Messwerten verbinden Sie einfach einen 10-uF-Kondensator mit dem Sensor (weitere Informationen zu Entstörkondensatoren finden Sie in Anhang C).

Sie können den Abstand vom EZ1 auch über dessen Analogausgang messen. Verbinden Sie den AN-Pin mit einem Analogeingang und lesen Sie den Wert mit `analogRead` ein. Der folgende Code gibt den Wert des Analogeingangs, umgewandelt in Zoll, aus:

```
value = analogRead(0);
inches = value / 2;  // jede Einheit von analogRead liegt bei etwa 5mv
Serial.println(inches);
```

Der Analogausgang liefert etwa 9,8mV pro Zoll zurück. Der Wert von `analogRead` liegt bei etwa 4,8mV pro Einheit (in Rezept 5.6 erfahren Sie mehr über `analogRead`). Der obige Code fasst jeweils zwei Einheiten zu einem Zoll zusammen. Der Rundungsfehler ist im Vergleich zur Genauigkeit des Sensors gering, doch wenn Sie eine genauere Berechnung wünschen, können Sie mit Fließkommazahlen arbeiten:

```
value = analogRead(0);
float mv = (value /1024.0) * 5000 ;
float inches = mv / 9.8; // 9,8mv pro Zoll
Serial.println(inches) ;
```

Siehe auch

Rezept 5.6 erläutert, wie man Werte von `analogInput` in Spannungswerte umwandelt.

Die Arduino-Referenz zu `pulseIn`: *http://www.arduino.cc/en/Reference/PulseIn*

6.5 Abstände genauer messen

Problem

Sie wollen messen, wie weit Objekte vom Arduino entfernt sind. Die Werte sollen aber genauer sein als in Rezept 6.4.

Lösung

Infrarotsensoren (IR) besitzen üblicherweise einen Analogausgang, der mit `analogRead`. eingelesen werden kann. Sie sind genauer als Ultraschallsensoren, haben aber eine geringere Reichweite (typischerweise zwischen 10 Zentimetern und 1 oder 2 Metern). Der folgende Sketch bietet die gleiche Funktionalität wie in Rezept 6.4, verwendet aber den Infrarotsensor Sharp GP2Y0A02YK0F (in Abbildung 6-6 sehen Sie die Verschaltung):

```
/* ir-distance Sketch
 * Gibt den Abstand eines IR-Sensors aus und ändert die
 * Blinkgeschwindigkeit einer LED in Abhängigkeit vom
 * Abstand.
 */

const int ledPin   = 13; // LED-Pin
const int sensorPin = 0;  // analoger Sensor-Pin

const long referenceMv = 5000; // long int, um Überlauf bei Multiplikation zu verhindern

void setup()
{
 Serial.begin(9600);
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 int val = analogRead(sensorPin);
 int mV = (val * referenceMv) / 1023;

 Serial.print(mV);
 Serial.print(",");
 int cm = getDistance(mV);
 Serial.println(cm);

 digitalWrite(ledPin, HIGH);
 delay(cm * 10 ); // Für jeden Zentimeter 10 Millisekunden hinzufügen
 digitalWrite(ledPin, LOW);
 delay( cm * 10);

 delay(100);
}

// Der Abstand wird aus einer Tabelle interpoliert
// Tabelleneinträge sind Abstände in Schritten von 250 Millivolt
const int TABLE_ENTRIES = 12;
```

```
const int firstElement = 250; // Erster Eintrag ist 250 mV
const int INTERVAL  = 250; // Millivolt zwischen den Elementen
static int distance[TABLE_ENTRIES] = {150,140,130,100,60,50,40,35,30,25,20,15};

int getDistance(int mV)
{
  if( mV >  INTERVAL * TABLE_ENTRIES-1 )
    return distance[TABLE_ENTRIES-1];
  else
  {
    int index = mV / INTERVAL;
    float frac = (mV % 250) / (float)INTERVAL;
    return distance[index] - ((distance[index] - distance[index+1]) * frac);
  }
}
```

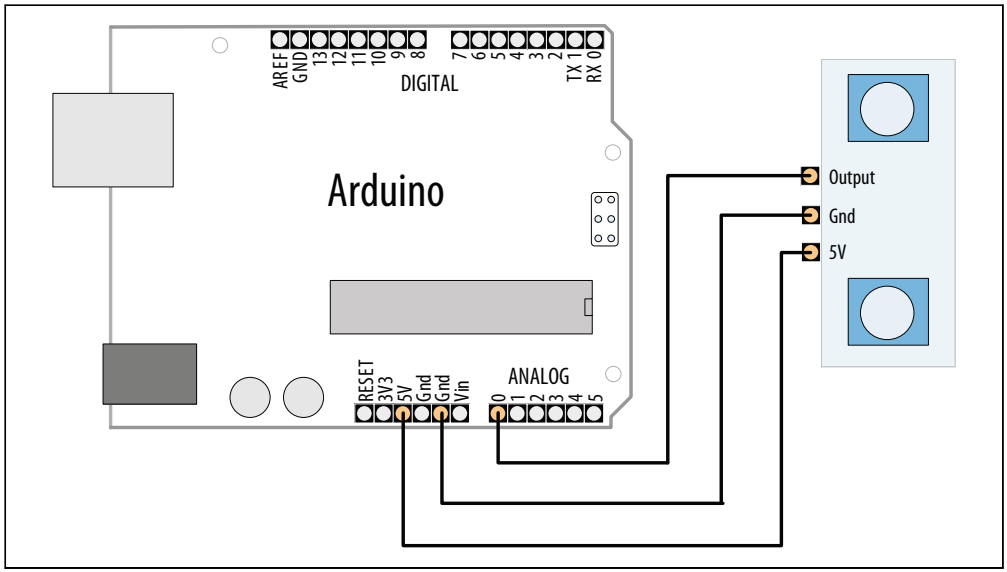


Abbildung 6-6: Anschluss des Sharp IR-Abstandssensors

Diskussion

Die vom IR-Sensor gelieferten Werte sind nicht linear, d.h., der von `analogRead` gelesene Wert ist nicht proportional zum Abstand. Die Berechnung ist daher etwas komplizierter als in Rezept 6.4. Der Sketch in diesem Rezept nutzt eine Tabelle, um den Abstand zu bestimmen. Dazu sucht er sich den nächstgelegenen Eintrag in der Tabelle, basierend auf dem Verhältnis von gemessenem Wert zum nächstgelegenen Tabelleneintrag (diese Technik nennt man *Interpolation*). Die Werte der Tabelle müssen für Ihren Sensor angepasst werden – nutzen Sie dazu das entsprechende Datenblatt oder probieren Sie es einfach aus.

Da die Werte für die Tabelle durch Ausprobieren ermittelt werden können (man misst die Spannung, bis sie sich um die nötige Menge erhöht hat, und misst dann den Abstand), können Sie diese Technik auch nutzen, wenn Ihnen keine Gleichung zur Interpretation der Werte zur Verfügung steht, z.B. wenn Sie kein Datenblatt besitzen.

Die Umwandlung der Spannung in den Abstand erfolgt in der Funktion:

```
int getDistance(int mV)
```

Die Funktion überprüft zuerst, ob der Wert innerhalb des Bereichs der Tabelle liegt. Der kleinste gültige Abstand wird zurückgeliefert, wenn der Wert nicht im Wertebereich liegt:

```
if( mV > INTERVAL * TABLE_ENTRIES )
  return distance[TABLE_ENTRIES-1]; //TABLE_ENTRIES-1 ist letzter gültiger Eintrag
```

Liegt der Wert innerhalb des Tabellenbereichs, ermittelt eine ganzzahlige Division, welcher Eintrag am nächsten liegt, aber kleiner als der Messwert ist:

```
int index = mV / INTERVAL ;
```

Der Modulo-Operator (siehe Kapitel 3) wird verwendet, um den Rest zu ermitteln, wenn ein Messwert zwischen zwei Einträgen liegt:

```
float frac = (mV % 250) / (float)INTERVAL;

return distance[index] + (distance[index]* (frac / interval));
```

Die letzte Zeile der `getDistance`-Funktion nutzt den Index und den Rest, um den Abstand zu berechnen und zurückzugeben. Sie liest den Wert aus der Tabelle und fügt, basierend auf dem `frac`-Wert, noch einen proportionalen Anteil hinzu. Das letzte Element ist eine Näherung, doch da es nur einen kleinen Teil des Ergebnisses ausmacht, ist es akzeptabel. Wenn Ihnen das Ergebnis nicht genau genug ist, müssen Sie eine Tabelle aufbauen, in der mehr Werte enger beieinander liegen.

Eine Tabelle kann auch die Performance verbessern, wenn die Berechnung lange dauert oder wenn immer wieder mit einer beschränkten Anzahl von Werten gerechnet wird. Berechnungen, insbesondere Fließkommaberechnungen, können sehr langsam sein. Diese Berechnungen durch Tabellen zu ersetzen, kann die Dinge deutlich beschleunigen.

Die Werte können (wie bei diesem Sketch) fest kodiert sein, oder in `setup()` berechnet werden. Der Sketch braucht dann zwar etwas länger zum Starten, aber da das nur einmal beim Einschalten des Arduino passiert, profitieren Sie danach beim jedem `loop()` vom Geschwindigkeitszuwachs. Den Geschwindigkeitszuwachs erkaufen Sie sich allerdings durch einen erhöhten Speicherbedarf – je größer die Tabelle, desto mehr RAM wird benötigt. Kapitel 17 zeigt, wie Sie `Progmem` nutzen können, um Daten im Programmspeicher abzulegen.

Bei langen Anschlussleitungen müssen Sie am Sensor möglicherweise einen Kondensator zwischen +5V und Masse schalten, um die Spannungsversorgung zu stabilisieren. Bei fehlerhaften Messwerten verbinden Sie einfach einen 10-uF-Kondensator mit dem Sensor (weitere Informationen zu Entstörkondensatoren finden Sie in Anhang C).

Siehe auch

Eine detaillierte Beschreibung des Sharp IR-Sensors finden Sie unter *http://www.societyofrobots.com/sensors_sharpirrange.shtml*.

6.6 Vibration messen

Problem

Sie wollen auf Vibration reagieren, z.B. wenn an eine Tür geklopft wird.

Lösung

Ein Piezo-Sensor reagiert auf Vibration. Er funktioniert am besten, wenn er mit einer größeren vibrierenden Oberfläche verbunden ist. Den Anschluss sehen Sie in Abbildung 6-7:

```
/* piezo Sketch
 * Schaltet eine LED ein, wenn geklopft wird
 */

const int sensorPin = 0;  // Analogpin für Sensor
const int ledPin  = 13;   // Pin für LED
const int THRESHOLD = 100;


void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
 int val = analogRead(sensorPin);
 if (val >= THRESHOLD)
 {
  digitalWrite(ledPin, HIGH);
  delay(100);  // Damit man die LED sieht
 }
 else
  digitalWrite(ledPin, LOW);
}
```

Diskussion

Ein Piezo-Sensor, auch Klopfsensor genannt, erzeugt eine Spannung als Reaktion auf eine physikalische Belastung. Je höher die Belastung, desto höher die Spannung. Das Piezo-Element ist gepolt, und die positive Seite (üblicherweise mit einem roten Draht oder mit »+« gekennzeichnet) wird mit dem analogen Eingang verbunden. Die negative Seite

(schwarz oder mit »–« gekennzeichnet) wird mit Masse verbunden. Ein hochohmiger Widerstand (1 Megaohm) wird parallel zum Sensor geschaltet.

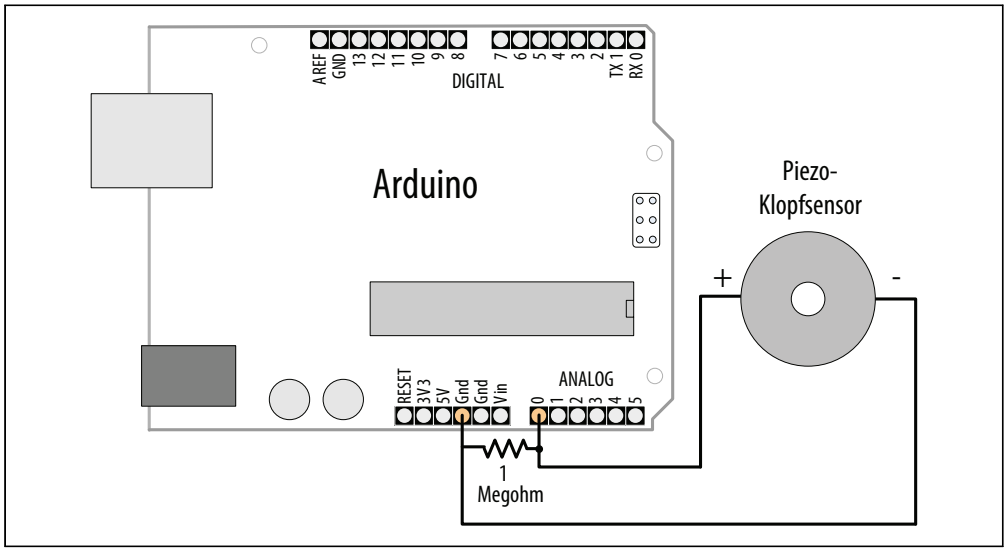


Abbildung 6-7: Anschluss eines Klopfsensors

Die Spannung wird über Arduinos `analogRead` eingelesen, um eine LED einzuschalten (in Kapitel 5 erfahren Sie mehr über die `analogRead`-Funktion). `THRESHOLD` definiert den Schwellwert, bei dem die LED eingeschaltet wird. Sie können diesen Wert erhöhen oder verkleinern und so die Empfindlichkeit des Sketches regeln.

Piezo-Sensoren kann man in Kunststoffgehäusen kaufen oder als einfache Metallplättchen mit zwei Kabeln dran. Die Komponenten selbst sind gleich, d.h., Sie können die Variante nutzen, die für Ihr Projekt am besten geeignet ist.

Manche Sensoren, wie das Piezo-Element, können auch vom Arduino angesteuert werden, um das zu erzeugen, was sie auch messen. Kapitel 9 zeigt, wie man mit einem Piezo-Element Töne erzeugen kann.

6.7 Geräusche erkennen

Problem

Sie wollen Geräusche wie Klatschen, Sprechen oder Schreien erkennen.

Lösung

Dieses Rezept verwendet das Breakout-Board BOB-08669 für das Electret-Mikrofon (SparkFun). Verbinden Sie das Board wie in Abbildung 6-8 zu sehen und laden Sie den Code auf den Arduino hoch.

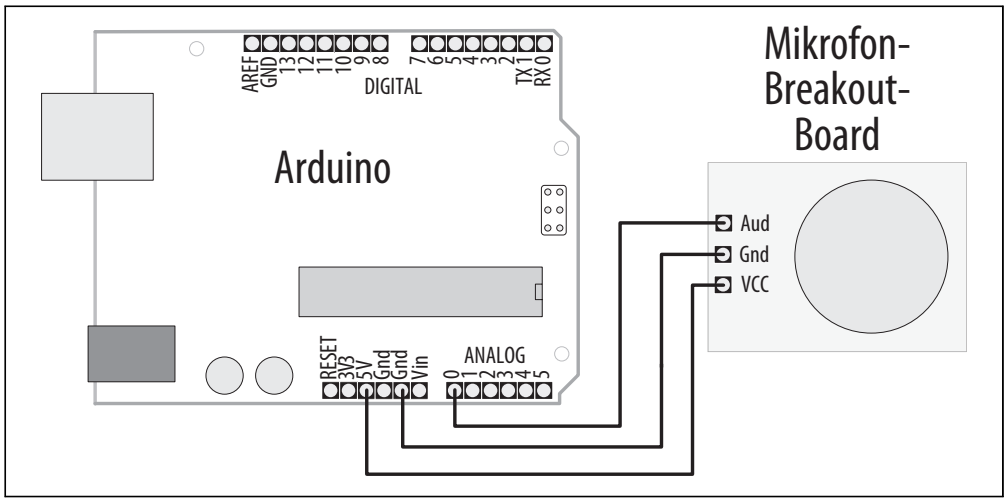


Abbildung 6-8: Anschluss des Mikrofon-Boards

Die eingebaute LED an Arduino-Pin 13 geht an, wenn Sie in der Nähe des Mikrofons klatschen, schreien oder laute Musik spielen. Möglicherweise müssen Sie den Schwellwert korrigieren – verwenden Sie den seriellen Monitor, um sich die hohen und niedrigen Werte anzusehen, und ändern Sie den Schwellwert so ab, dass er zwischen den hohen Werten (wenn Geräusche vorhanden sind) und den niedrigen Werten (wenn keine oder nur wenige Geräusche zu hören sind) liegt. Laden Sie den angepassten Code auf den Arduino hoch und probieren Sie es erneut:

```
/*
microphone Sketch

SparkFun Breakout-Board für Electret-Mikrofon an Analogpin 0
*/

const int ledPin = 13;       //LED an Pin 13
const int middleValue = 512;    //Mitte des analogen Wertebereichs
const int numberOfSamples = 128; //Messwerte pro Durchgang

int sample;             //Vom Mikro gelieferter Wert
long signal;             //Messwert ohne DC-Offset
long averageReading;        //Durchschnitt des Messdurchgangs

long runningAverage=0;      //Laufender Durchschnitt der berechneten Werte
const int averagedOver= 16;   //Wie schnell wirken sich neue Werte auf den laufenden Durchschnitt
aus?
                //Größere Zahlen bedeuten langsamer

const int threshold=400;     //Bei welchem Schwellwert schaltet sich die LED ein?
void setup() {
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
}
```

```
void loop() {
 long sumOfSquares = 0;
 for (int i=0; i<numberOfSamples; i++) { //Viele Messwerte einlesen und Durchschnitt bilden
  sample = analogRead(0);          //Wert einlesen
  signal = (sample - middleValue);    //Offset von der Mitte abziehen
  signal *= signal;             //Quadrieren, damit alle Werte positiv sind
  sumOfSquares += signal;          //Zum Gesamtwert hinzuaddieren
 }
 averageReading = sumOfSquares/numberOfSamples;    //Laufenden Durchschnitt berechnen
 runningAverage=(((averagedOver-1)*runningAverage)+averageReading)/averagedOver;

 if (runningAverage>threshold){      //Durchschnitt über Schwellwert?
  digitalWrite(ledPin, HIGH);       //Ja, LED einschalten
 }else{
  digitalWrite(ledPin, LOW);        //Nein, LED ausschalten
 }
 Serial.println(runningAverage);     //Wert zu Testzwecken ausgeben
}
```

Diskussion

Ein Mikrofon erzeugt sehr schwache elektrische Signale. Würden Sie es direkt mit einem Pin des Arduino verbinden, wäre keine Änderung messbar. Das Signal muss zuerst verstärkt werden, damit es vom Arduino genutzt werden kann. Das SparkFun-Board besitzt ein Mikrofon und eine integrierte Verstärkerschaltung, die das Signal so aufbereitet, dass es vom Arduino verarbeitet werden kann.

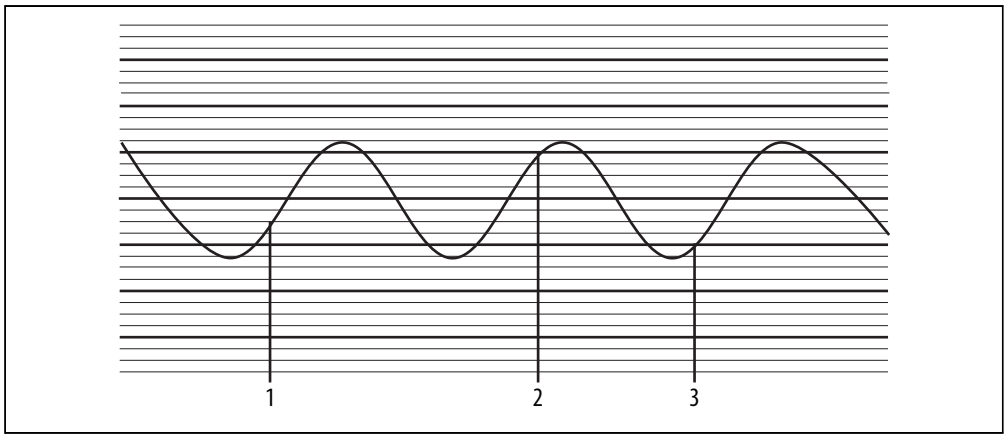


Abbildung 6-9: Messung eines Audiosignals an drei Stellen

Da wir in diesem Rezept ein Audiosignal messen, müssen wir einige zusätzliche Berechnungen vornehmen, um an nützliche Informationen zu gelangen. Ein Audiosignal verändert sich recht schnell, und der von `analogRead` zurückgelieferte Wert hängt davon ab, an welchem Punkt des wellenförmigen Signals Sie einen Messwert einlesen. Wenn Sie mit `analogRead` nicht vertraut sind, sehen Sie sich Kapitel 5 und Rezept 6.2 an. Ein Beispiel für die Wellenform eines Audiosignals ist in Abbildung 6-9 zu sehen. Mit der Zeit steigt und sinkt die Spannung in einem regelmäßigen Muster. Wenn Sie zu drei verschiedenen

Zeitpunkten messen, erhalten Sie auch drei unterschiedliche Messwerte. Würden Sie das nutzen, um eine Entscheidung zu treffen, könnten Sie fälschlicherweise schließen, dass das Signal in der Mitte lauter wird.

Eine genaue Messung verlangt daher mehrere, zeitlich nah beeinanderliegende, Messungen. Die Spitzen und Täler nehmen zu, wenn das Signal größer wird. Die Differenz zwischen dem Tiefpunkt eines Tals und dem obersten Punkt einer Spitze nennt man *Amplitude*, und sie steigt an, wenn das Signal lauter wird.

Um die Größe der Spitzen und Täler zu bestimmen, messen Sie die Differenz zwischen Spannungsmitte und den Größen der Spitzen und Täler. Sie können sich diesen Mittelpunkt als Linie vorstellen, die genau in der Mitte zwischen der höchsten Spitze und dem tiefsten Tal verläuft (siehe Abbildung 6-10). Die Linie repräsentiert den Spannungs-Offset des Signals (die Spannung ohne Spitzen oder Täler). Wenn Sie diesen Offset von den `analogRead`-Werten abziehen, erhalten Sie den korrekten Wert der Signalamplitude.

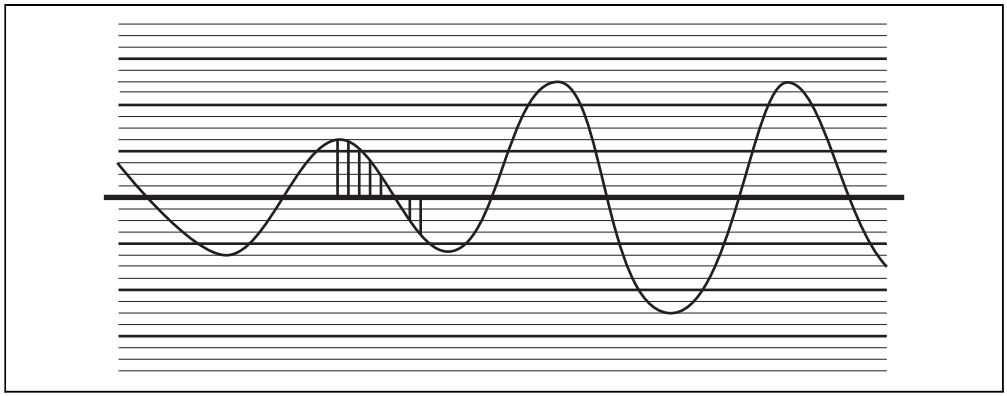


Abbildung 6-10: Audiosignal mit Offset (Signal-Mittelpunkt)

Wird das Signal lauter, steigt die Durchschnittsgröße dieser Werte an, doch da einige Werte negativ sind (wenn das Signal unter den Offset fällt), heben sie sich gegenseitig auf, und der Durchschnitt tendiert gegen 0. Um das zu korrigieren, quadrieren wir jeden Wert (nehmen ihn mit sich selbst mal). Das macht alle Werte positiv und erhöht den Unterschied bei kleinen Änderungen, was für die Untersuchung ebenfalls hilfreich ist. Der Durchschnittswert steigt und sinkt nun mit der Signalamplitude.

Für diese Berechnung müssen wir den Wert des Offsets kennen. Um ein sauberes Signal zu erhalten, wird der Mikrofonverstärker einen Offset verwenden, der so mittig wie möglich im erlaubten Spannungbereich liegt, damit das Signal so groß wie möglich werden kann, ohne zu verzerren. Unser Code geht genau davon aus und verwendet den Wert 512 (genau in der Mitte des analogen Wertebereichs von 0 bis 1023).

Die Variablenwerte am Anfang des Sketches können angepasst werden, wenn der Sketch auf die von Ihnen benötigten Tonlagen nicht gut reagiert.

numberOfSamples ist auf 128 gesetzt – setzt man diesen Wert zu klein an, deckt der Durchschnitt die kompletten Zyklen der Wellenform möglicherweise nicht adäquat ab, und Sie erhalten falsche Ergebnisse. Setzen Sie den Wert hingegen zu hoch an, bilden Sie den Durchschnitt über einen zu langen Zeitraum und sehr kurze Töne können in der großen Datenmenge verloren gehen. Es kann auch zu einer deutlichen Verzögerung zwischen Ton und aufleuchtendes LED kommen. Für die Berechnung verwendete Konstanten wie numberOfSamples und averagedOver verwenden Zweierpotenzen (128 bzw. 16). Versuchen Sie, Werte zu benutzen, die durch 2 teilbar sind, um die größtmögliche Performance zu erreichen (mehr zu mathematischen Funktionen erfahren Sie in Kapitel 3).

6.8 Temperatur messen

Problem

Sie wollen die Temperatur ausgeben, oder den Wert zur Steuerung eines Gerätes nutzen, etwa um etwas zu schalten, wenn die Temperatur einen Schwellwert erreicht.

Lösung

Das Rezept gibt die Temperatur in Fahrenheit und Celsius aus. Es verwendet den beliebten Temperatursensor LM35. Der Sensor sieht aus wie ein Transistor, und die Verschaltung ist in Abbildung 6-11 zu sehen:

```
/*
 lm35 Sketch
 Gibt die Temperatur über den seriellen Monitor aus
 */

const int inPin = 0; // Analogpin


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


void loop()
{
 int value = analogRead(inPin);
 Serial.print(value); Serial.print(" > ");
 float millivolts = (value / 1024.0) * 5000;
 float celsius = millivolts / 10;  // 10mV pro Grad Celsius
 Serial.print(celsius);
 Serial.print(" Grad Celsius, ");

 Serial.print( (celsius * 9)/ 5 + 32 );  // Umwandlung in Fahrenheit
 Serial.println(" Grad Fahrenheit");

 delay(1000); // Eine Sekunde warten

}
```

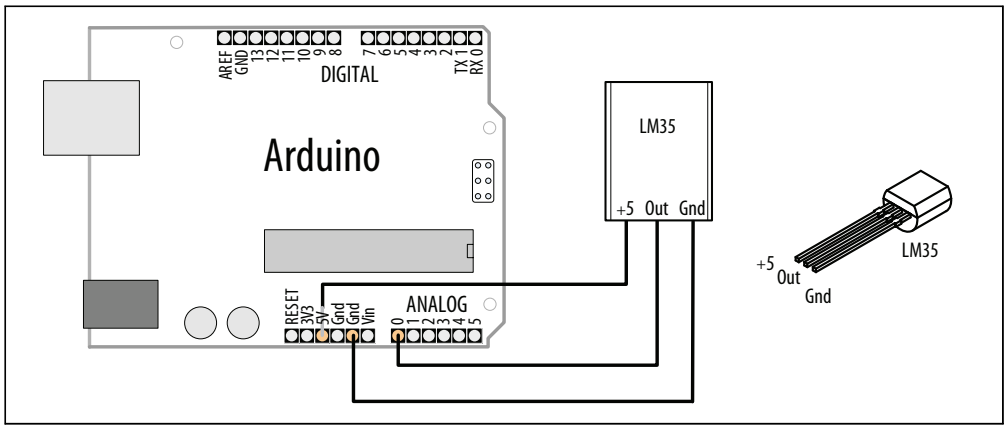


Abbildung 6-11: Anschluss des Temperatursensors LM35

Diskussion

Der Temperatursensor LM35 erzeugt eine Analogspannung, die zur Temperatur direkt proportional ist. Sie beträgt am Ausgang 1 Millivolt je 0,1°C (10mV pro Grad).

Der Sketch wandelt die `analogRead`-Werte in Millivolt um (siehe Kapitel 5) und teilt sie dann durch 10, um die Temperatur in Grad zu ermitteln.

Die Genauigkeit des Sensors liegt bei etwa 0,5°C, und in vielen Fällen können Sie auf Fließkommazahlen verzichten und mit ganzen Zahlen arbeiten.

Der folgende Sketch aktiviert Pin 13, wenn die Temperatur einen Schwellwert überschreitet:

```
const int inPin = 0; // Analogpin für Senso
const int outPin = 13; // Digitalpin für LED

const int threshold = 25; // Schwellwert, der den Ausgangspin anstößt


void setup()
{
 Serial.begin(9600);
 pinMode(outPin, OUTPUT);
}


void loop()
{
 int value = analogRead(inPin);
 long celsius = (value * 500L) /1024;   // 10 mV je Grad C, siehe Text
 Serial.print(celsius);
 Serial.print(" Grad Celsius: ");
 if(celsius > threshold)
 {
```

```
  digitalWrite(outPin, HIGH);
  Serial.println("Pin ist an");
 }
 else
 {
  digitalWrite(outPin, LOW);
  Serial.println("Pin ist aus");
 }
 delay(1000); // Eine Sekunde warten
}
```

Der Sketch nutzt nur long-Werte (32-Bit) zur Berechnung. Der Buchstabe *L* hinter einer Zahl sorgt dafür, dass die Berechnung in long-Arithmetik erfolgt, damit die Multiplikation der Maximaltemperatur (500 bei einem 5-V-Arduino) mit dem eingelesenen Analogwert nicht zu einem Überlauf führt. Weitere Informationen zur Umwandlung von Analogpegeln in Spannungen finden Sie in Kapitel 5.

Wenn Sie Werte in Fahrenheit benötigen, können Sie den LM34-Sensor verwenden, der Ausgaben in Fahrenheit erzeugt, oder Sie können die Werte mit der folgenden Formel umrechnen:

```
float f = (celsius * 9)/ 5 + 32 );
```

Eine Alternative zur Temperaturmessung ist der LM335. Er sieht aus wie der LM35, wird aber anders verschaltet und genutzt.

Die LM335-Ausgabe entspricht 10mV pro Grad Kelvin, also 2,731 Volt bei 0 Grad Celsius. Ein Vorwiderstand wird für die Betriebsspannung benötigt. Oft wird ein 2K-Ohm-Widerstand verwendet, es können aber auch 2,2K-Ohm sein. Hier ein Sketch, der die Temperatur mit Hilfe des LM335 ausgibt (die Verschaltung sehen Sie in Abbildung 6-12):

```
/*
 lm335 Sketch
 Gibt die Temperatur über den seriellen Monitor aus
 */

const int inPin = 0; // Analogpin


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


void loop()
{
 int value = analogRead(inPin);
 Serial.print(value); Serial.print(" > ");
 float millivolts = (value / 1024.0) * 5000;
 // 10mV pro Grad Kelvin, 0 Grad Celsius ist 273,15
 float celsius = (millivolts / 10) - 273.15 ;

 Serial.print(celsius);
 Serial.print(" Grad Celsius, ");
```

```
 Serial.print( (celsius * 9)/ 5 + 32 ); // In Fahrenheit umwandeln
 Serial.println(" Grad Fahrenheit");

 delay(1000); // Eine Sekunde warten
}
```

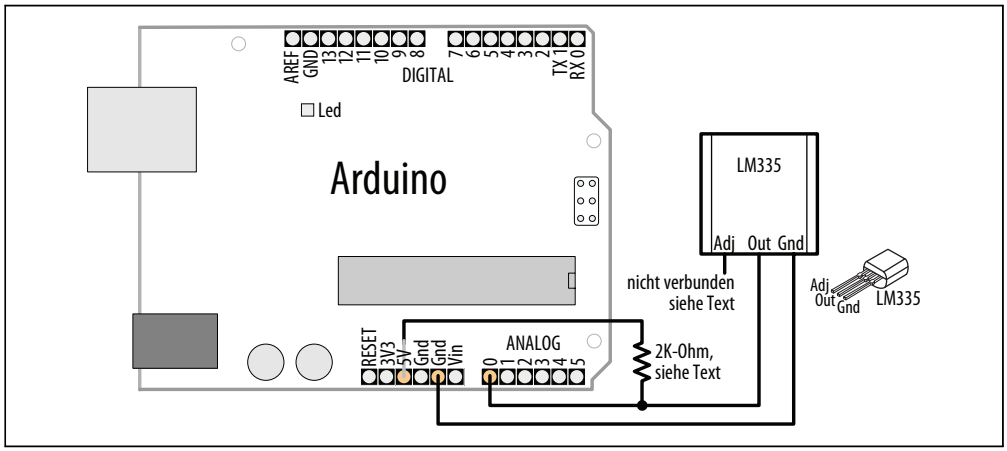


Abbildung 6-12: Anschluss des Temperatursensors LM335

Sie können die Genauigkeit verbessern, indem Sie den nicht angeschlossenen *adj*-Pin mit dem Schleifer eines 10-K-Potis (und die beiden anderen Anschlüsse mit +5V und Masse) verbinden. Gleichen Sie die Einstellung dann mit dem Poti über ein bekanntermaßen genaues Thermometer ab.

Siehe auch

LM35-Datenblatt: *http://www.national.com/ds/LM/LM35.pdf*

LM335-Datenblatt: *http://www.national.com/ds/LM/LM135.pdf*

6.9 RFID-Tags lesen

Problem

Sie wollen einen RFID-Tag lesen und auf bestimmte IDs reagieren.

Lösung

Abbildung 6-13 zeigt die Anbindung eines Parallax RFID- (Radio Frequency Identification)Lesers an den seriellen Port des Arduino. (Möglicherweise müssen Sie den Leser abklemmen, um den Sketch hochladen zu können)

Dieser Leser arbeitet mit 125-kHz-Tags. Wenn Sie einen anderen Leser nutzen, müssen Sie den korrekten Anschluss und die richtige Nutzung in der Dokumentation nachschlagen.

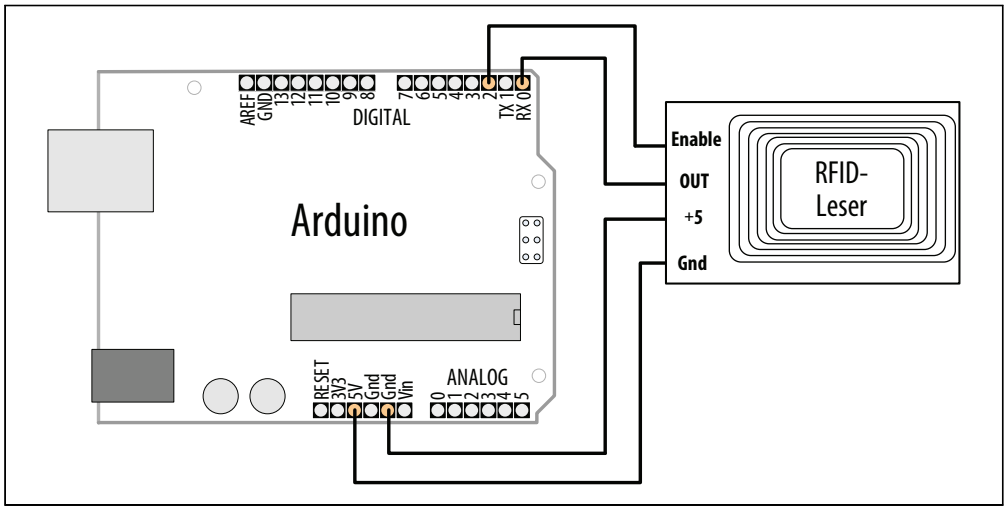


Abbildung 6-13: Serieller RFID-Leser am Arduino

Der Sketch liest den Wert eines RFID-Tags ein und gibt ihn aus:

```
/*
 RFID Sketch
 Gibt den von einem RFID-Tag eingelesenen Wert aus
 */

const int startByte  = 10; // ASCII-Linefeed vor jedem Tag
const int endByte    = 13; // ASCII Carriage Return schließt jeden Tag ab
const int tagLength  = 10; // Anzahl der Ziffern im Tag
const int totalLength = tagLength + 2; //Tag-Länge + Start- und End-Byte

char tag[tagLength + 1]; // Enthält den Tag und eine abschließende 0

int bytesread = 0;

void setup()
{
 Serial.begin(2400);   // Baudrate Ihres RFID-Lesers verwenden
 pinMode(2,OUTPUT);    // Mit RFID ENABLE-Pin verbunden
 digitalWrite(2, LOW); // RFID-Leser aktivieren
}

void loop()
{
 if(Serial.available() >= totalLength) // Ausreichend Daten da?
 {
  if(Serial.read() == startByte)
  {
```

```
    bytesread = 0;          // Am Anfang des Tags setzen wir den Zähler auf 0 zurück
    while(bytesread < tagLength) // 10-Ziffern-Code einlesen
    {
     int val = Serial.read();
     if((val == startByte)||(val == endByte)) // Auf Code-Ende prüfen
      break;
     tag[bytesread] = val;
     bytesread = bytesread + 1; // Bereit für nächste Ziffer
    }
    if( Serial.read() == endByte) // Auf korrektes End-Zeichen prüfen
    {
     tag[bytesread] = 0; // String abschließen
     Serial.print("RFID-Tag ist: ");
     Serial.println(tag);
    }
   }
  }
}
```

Diskussion

Ein Tag besteht aus einem Startzeichen, gefolgt vom eigentlichen 10-Ziffern-Code und einem End-Zeichen. Der Sketch wartet, bis ein vollständiger Tag verfügbar ist, und gibt ihn dann aus, wenn er gültig ist. Der Tag wird in Form von ASCII-Ziffern empfangen (mehr zum Empfang von ASCII-Ziffern finden Sie in Rezept 4.4). Sie können den empfangenen String auch in eine Zahl umwandeln, wenn Sie ihn speichern oder mit anderen Werten vergleichen wollen. Dazu ändern Sie die letzten Codezeilen wie folgt ab:

```
    if( Serial.read() == endByte) // Auf korrektes End-Zeichen prüfen
    {
     tag[bytesread] = 0; // String abschließen
     long tagValue = atol(tag); // ASCII-Tag in long-Wert umwandeln
     Serial.print("RFID-Tag ist: ");
     Serial.println(tagValue);
    }
```

RFID steht für »Radio Frequency Identification«, und wie es der Name andeutet, reagiert es auf Radiofrequenzen empfindlich und ist entsprechend störanfällig. Der Code in diesem Rezept verwendet nur Codes mit der richtigen Länge und den richtigen Start- und Stop-Bits, was die meisten Fehler eliminieren sollte. Doch Sie können den Code auch stabiler machen, indem Sie den Tag wiederholt einlesen und die Daten nur nutzen, wenn sie übereinstimmen. (RFID-Leser wiederholen den Code, solange sich ein gültiger Tag in der Nähe befindet.) Zu diesem Zweck erweitern Sie die letzten Zeilen im obigen Code-Fragment wie folgt:

```
if( Serial.read() == endByte) // Auf korrektes End-Zeichen prüfen
{
 tag[bytesread] = 0; // String abschließen
 long tagValue = atol(tag); // ASCII-Tag in long-Wert umwandeln
 if (tagValue == lastTagValue)
 {
  Serial.print("RFID-Tag ist: ");
  Serial.println(tagValue);
```

```
    lastTagValue = tagValue;
  }
}
```

Sie müssen zu Beginn des Sketches noch eine Deklaration für `lastTagValue` einfügen:

```
long lastTagValue=0;
```

Dieser Ansatz ähnelt dem Code aus Rezept 5.3. Eine Bestätigung erhalten Sie nur, wenn der Tag lange genug verfügbar ist, um zweimal gelesen zu werden, doch Fehler nehmen auf diese Weise deutlich ab. Sie können ein versehentliches Einlesen verhindern, indem Sie dafür sorgen, dass der Tag eine gewisse Zeit verfügbar sein muss, bevor die Nummer ausgegeben wird.

6.10 Drehbewegungen messen

Problem

Sie wollen die Drehbewegung eines Objekts messen und ausgeben, um seine Geschwindigkeit und/oder Richtung verfolgen zu können.

Lösung

Um eine Drehbewegung zu messen, können Sie einen Drehwinkelgeber nutzen, der mit dem Objekt verbunden ist, das Sie nachhalten wollen. Schließen Sie den Drehwinkelgeber wie in Abbildung 6-14 an:

```
/*
 Drehwinkelgeber einlesen
 Diese einfache Version fragt nur die Encoder-Pins ab
 Die Position wird über den seriellen Monitor ausgegeben
 */


const int encoderPinA = 4;
const int encoderPinB = 2;
const int encoderStepsPerRevolution=16;
int angle = 0;

int val;

int encoderPos = 0;
boolean encoderALast = LOW;  // Vorigen Pin-Zustand merken


void setup()
{
 pinMode(encoderPinA, INPUT);
 pinMode(encoderPinB, INPUT);
 digitalWrite(encoderPinA, HIGH);
 digitalWrite(encoderPinB, HIGH);
 Serial.begin (9600);
}
```

```
void loop()
{
 boolean encoderA = digitalRead(encoderPinA);

 if ((encoderALast == HIGH) && (encoderA == LOW))
 {
  if (digitalRead(encoderPinB) == LOW)
  {
   encoderPos--;
  }
  else
  {
   encoderPos++;
  }
  angle=(encoderPos % encoderStepsPerRevolution)*360/encoderStepsPerRevolution;
  Serial.print (encoderPos);
  Serial.print (" ");
  Serial.println (angle);
 }

 encoderALast = encoderA;
}
```

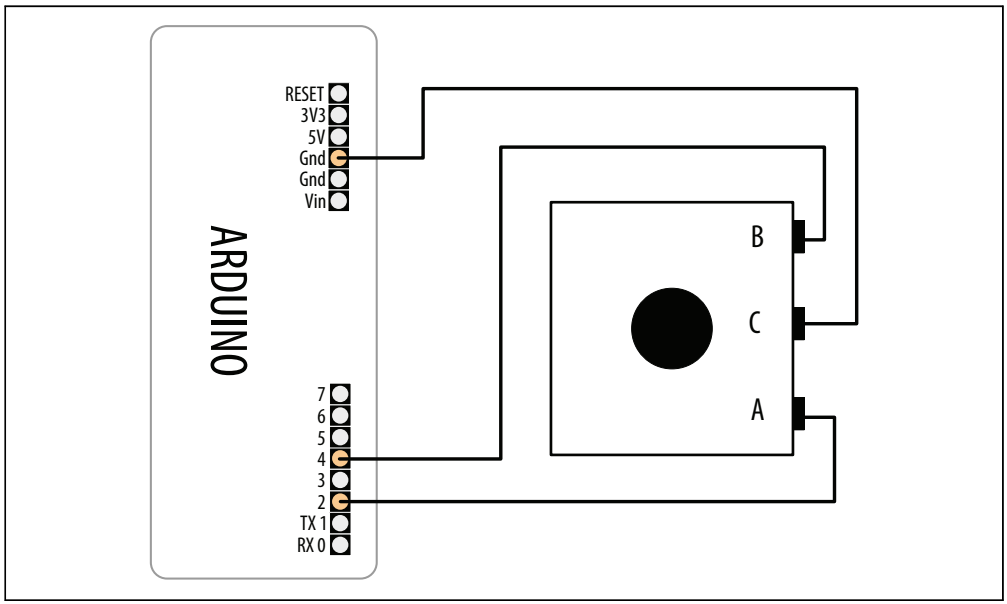


Abbildung 6-14: Drehwinkelgeber

Diskussion

Ein Drehwinkelgeber erzeugt zwei Signale, während er bewegt wird. Beide Signale wechseln zwischen `HIGH` und `LOW`, während sich die Achse dreht, doch die Signale sind zueinander leicht verschoben. Wenn Sie den Punkt erkennen, an dem eines der Signale

von HIGH zu LOW wechselt, sagt Ihnen der Zustand des anderen Pins (HIGH oder LOW), in welche Richtung sich die Achse dreht.

Die erste Zeile des Codes in der loop-Funktion liest also einen der Encoder-Pins ein:

```
int encoderA = digitalRead(encoderPinA);
```

Dann wird dieser Wert mit dem vorigen verglichen, um zu sehen, ob er gerade auf LOW gegangen ist:

```
if ((encoderALast == HIGH) && (encoderA == LOW))
```

Ist das nicht der Fall, wird der nachfolgende Code-Block nicht ausgeführt, sondern zum Ende von loop gesprungen, wo der gerade gelesene Wert in encoderALast gespeichert wird. Dann geht es mit einem neuen Messwert wieder von vorne los.

Gibt der folgende Ausdruck

```
if ((encoderALast == HIGH) && (encoderA == LOW))
```

true zurück, liest der Code den anderen Encoder-Pin ein und inkrementiert bzw. dekrementiert encoderPos in Abhängigkeit vom zurückgelieferten Wert. Er berechnet den Winkel der Achse (0 ist dabei der Punkt, an dem sich die Achse befand, als der Code gestartet wurde). Der Wert wird dann über den seriellen Port gesendet, und Sie können ihn auf dem seriellen Monitor sehen.

Drehwinkelgeber gibt es mit verschiedenen Auflösungen (*Schritten pro Umdrehung*). Sie gibt an, wie oft das Signal bei einer Umdrehung der Achse zwischen HIGH und LOW wechselt. Die Werte reichen dabei von 16 bis 1000. Bei höheren Werten können kleinere Bewegungen erkannt werden, aber dafür kosten die Encoder auch deutlich mehr Geld. Die Auflösung des Encoders ist im Code fest eingetragen:

```
const int encoderStepsPerRevolution=16;
```

Bei einem anderen Encoder müssen Sie den Wert entsprechend korrigieren.

Wenn Sie keine auf- und absteigenden Werte erhalten, egal in welcher Richtung Sie den Geber drehen, sollten Sie den Test umkehren und nach einer steigenden statt einer fallenden Flanke Ausschau halten. Vertauschen Sie die LOW- und HIGH-Werte in der Zeile, in der die Werte überprüft werden, wie folgt:

```
if ((encoderALast == LOW) && (encoderA == HIGH))
```

Drehwinkelgeber erzeugen nur ein Inkrement bzw. Dekrement. Sie können Ihnen nicht direkt sagen, in welchem Winkel sich die Achse gerade befindet. Der Code berechnet das, doch immer relativ zur Achsstellung beim Programmstart. Der Code überwacht die Pins, indem er sie kontinuierlich abfragt (engl. *Polling*). Es gibt keine Garantie, dass sich die Pinwerte nicht geändert haben, seit sie zuletzt abgefragt wurden, d.h., wenn sich der Code um viele andere Dinge kümmern muss und der Encoder sehr schnell gedreht wird, dann ist es möglich, dass einige Schritte verloren gehen. Das ist bei hochauflösenden Encodern wahrscheinlicher, da sie beim Drehen viel mehr Signale senden.

Um die Geschwindigkeit herauszufinden, müssen Sie zählen, wie viele Schritte in eine Richtung in einer festgelegten Zeit registriert werden.

6.11 Mehrere Drehbewegungen messen

Problem

Sie arbeiten mit zwei oder mehr Drehwinkelgebern und wollen deren Drehbewegungen messen und ausgeben.

Lösung

Die Schaltung arbeitet mit zwei Encodern, die wie in Abbildung 6-15 angeschlossen sind. Weitere Informationen zu Drehwinkelgebern finden Sie in Rezept 6.10:

```
/*
 RotaryEncoderMultiPoll
 Der Sketch verwendet zwei Encoder.
 Einer ist mit den Pins 2 und 3,
 der andere mit den Pins 4 und 5 verbunden
 */

const int  ENCODERS = 2; // Anzahl Encoder

const int encoderPinA[ENCODERS] = {2,4};    // encoderA, Pins an 2 und 4
const int encoderPinB[ENCODERS] = {3,5};    // encoderB, Pins an 3 und 5
int encoderPos[ ENCODERS] = { 0,0};       // Positionen auf 0 setzen
boolean encoderALast[ENCODERS] = { LOW,LOW}; // Letzter Zustand des encoderA-Pins


void setup()
{
 for (int i=2; i<6; i++){
  pinMode(i, HIGH);
  digitalWrite(i, HIGH);
 }
 Serial.begin (9600);
}

int updatePosition( int encoderIndex)
{
 boolean encoderA = digitalRead(encoderPinA[encoderIndex]);
 if ((encoderALast[encoderIndex] == HIGH) && (encoderA == LOW))
 {
  if (digitalRead(encoderPinB[encoderIndex]) == LOW)
  {
   encoderPos[encoderIndex]--;
  }
  else
  {
   encoderPos[encoderIndex]++;
```

```
  }
  Serial.print("Encoder ");
  Serial.print(encoderIndex,DEC);
  Serial.print("=");
  Serial.print (encoderPos[encoderIndex]);
  Serial.println ("/");
 }
 encoderALast[encoderIndex] = encoderA;
}


void loop()
{
 for(int i=0; i < ENCODERS;i++)
 {
  updatePosition(i);
 }
}
```

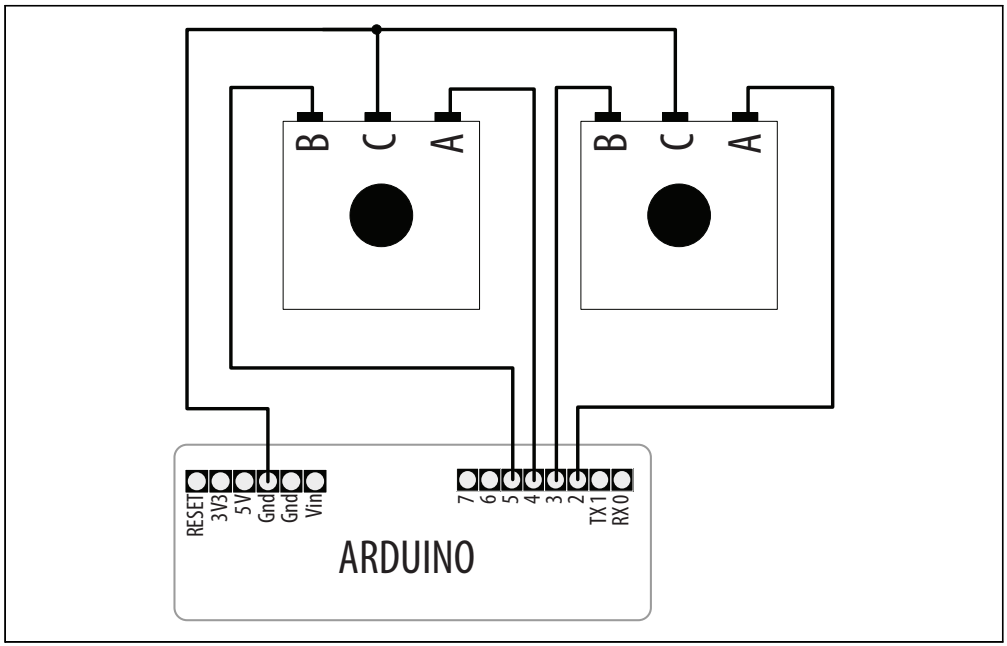


Abbildung 6-15: Anschluss zwei Drehwinkelgeber

Diskussion

Dieses Rezept nutzt die gleiche Logik wie Rezept 6.10, in dem ein Encoder gelesen wurde. Hier wird aber ein Array für alle Variablen verwendet, die für jeden Encoder separat vorgehalten werden müssen. Wir können dann eine `for`-Schleife nutzen, um alle Encoder einzulesen und ihre Rotation zu berechnen. Um mehr Encoder zu nutzen, müssen Sie den `ENCODERS`-Wert entsprechend erhöhen, die Arrays erweitern und definieren, an welchen Pins sie angeschlossen sind.

Wenn Sie keine auf- und absteigenden Werte erhalten, egal in welcher Richtung Sie den Geber drehen, sollten Sie den Test umkehren und nach einer steigenden statt einer fallenden Flanke Ausschau halten. Vertauschen Sie die LOW- und HIGH-Werte in der Zeile

```
if ((encoderALast[encoderIndex] == HIGH) && (encoderA == LOW))
```

wie folgt:

```
if ((encoderALast[encoderIndex] == LOW) && (encoderA == HIGH))
```

Wenn ein Encoder funktioniert, der andere aber nur hochzählt, tauschen Sie einfach die A- und B-Verbindungen des betreffenden Encoders.

6.12 Drehbewegungen in einem viel beschäftigten Sketch messen

Problem

Nachdem Ihr Code gewachsen ist und neben dem Einlesen der Encoderwerte noch andere Dinge erledigen muss, wird der Encoder immer unzuverlässiger. Das Problem ist besonders groß, wenn sich die Achse schnell dreht.

Lösung

Die Schaltung entspricht der aus Rezept 6.11. Wir verwenden einen Interrupt, um sicherzustellen, dass der Code auf jeden Schritt reagiert:

```
/*
 RotaryEncoderInterrupt Sketch
 */

const int encoderPinA = 2;
const int encoderPinB = 4;
int Pos, oldPos;
volatile int encoderPos = 0; // Bei Interrupts geänderte Variablen sind volatil


void setup()
{
 pinMode(encoderPinA, INPUT);
 pinMode(encoderPinB, INPUT);
 digitalWrite(encoderPinA, HIGH);
 digitalWrite(encoderPinB, HIGH);
 Serial.begin(9600);

 attachInterrupt(0, doEncoder, FALLING); // Encoder-Pin an Interrupt 0 (Pin 2)
}

void loop()
{
 uint8_t oldSREG = SREG;
```

```
 cli();
 Pos = encoderPos;
 SREG = oldSREG;
 if(Pos != oldPos)
 {
  Serial.println(Pos,DEC);
  oldPos = Pos;
 }
 delay(1000);
}


void doEncoder()
{
 if (digitalRead(encoderPinA) == digitalRead(encoderPinB))
  encoderPos++;   // Hochzählen, wenn Encoder-Pins gleich sind
 else
  encoderPos--;   // Runterzählen, wenn Pins nicht gleich sind
}
```

Der Code gibt den `Pos`-Wert höchstens einmal in der Sekunde über den seriellen Port aus (aufgrund der Pause). Der ausgegebene Wert berücksichtigt aber alle Bewegungen, die während dieser Pause ausgeführt wurden.

Diskussion

Wenn Ihr Code mehr Aufgaben erledigen muss, werden die Encoder-Pins seltener abgefragt. Durchlaufen die Pins eine Schrittänderung, bevor die Daten gelesen werden, kann der Arduino den Schritt nicht erkennen. Wird die Achse schnell bewegt, passiert das öfter, weil die Schritte schneller kommen.

Um sicherzustellen, dass der Code bei jedem Schritt reagiert, müssen Sie Interrupts (»Unterbrechungen«) verwenden. Tritt ein Interrupt ein, unterbricht der Code, was er gerade macht, führt den Interrupt-Code aus und macht dann da weiter, wo er unterbrochen wurde.

Bei einem Standard-Arduino-Board können die beiden Pins 2 und 3 als Interrupts verwendet werden. Der Interrupt wird durch die folgende Zeile aktiviert:

```
attachInterrupt(0, doEncoder, FALLING);
```

Die drei benötigten Parameter sind die Interrupt-Pin-ID (0 für Pin 2, 1 für Pin 3), die Funktion, die bei einem Interrupt ausgeführt werden soll (in diesem Fall `doEncoder`) und schließlich das Verhalten des Pins, bei dem der Interrupt ausgelöst wird (in diesem Fall das Abfallen der Spannung von 5 auf 0 Volt). Die anderen Optionen sind `RISING` (die Spannung steigt von 0 auf 5 Volt) und `CHANGE` (Spannung steigt oder fällt).

Die Funktion `doEncoder` prüft die Encoder-Pins, um zu sehen, in welche Richtung sich die Achse gedreht hat, und setzt `encoderPos` entsprechend.

Wenn sich der Wert unabhängig von der Drehrichtung nur erhöht, ändern Sie den Interrupt so ab, dass er auf die ansteigende Flanke (`RISING`) statt auf die fallende (`FALLING`) reagiert.

Da `encoderPos` in der Funktion geändert wird, die beim Interrupt ausgeführt wird, muss sie als `volatile` deklariert werden. Das teilt dem Compiler mit, dass sie jederzeit verändert werden kann. Optimieren Sie den Code nicht aufgrund der Annahme, dass er sich nicht verändert hat, da der Interrupt jederzeit eintreten kann.

Der Arduino Build-Prozess optimiert den Code, indem er Code und Variablen entfernt, die von Ihrem Sketch nicht genutzt werden. Nur in Interrupt-Handlern veränderte Variablen müssen daher als `volatile` deklariert werden, damit der Compiler weiß, dass er diese Variablen nicht entfernen soll.

Um diese Variable in der Hauptschleife zu lesen, müssen Sie besondere Vorkehrungen treffen, um sicherzustellen, dass der Interrupt nicht eintritt, wenn Sie sie gerade lesen. Diese Aufgabe übernimmt das folgende Code-Fragment:

```
uint8_t oldSREG = SREG;

cli();
Pos = encoderPos;
SREG = oldSREG;
```

Zuerst sichern Sie den aktuellen Status von `SREG` (den Interrupt-Registern) und schalten die Interrupts dann mit `cli` aus. Der Wert wird gelesen und das Wiederherstellen von `SREG` schaltet die Interrupts wieder ein und alles ist so, wie es war. Tritt ein Interrupt ein, während die Interrupts ausgeschaltet sind, dann wartet er, bis sie wieder eingeschaltet werden. Dieser Zeitraum ist so kurz, dass kein Interrupt verloren geht (solange Sie den Code im Interrupt-Handler so kurz wie möglich halten).

6.13 Eine Maus nutzen

Problem

Sie wollen die Bewegungen einer PS/2-kompatiblen Maus verarbeiten und auf Änderungen der *x*- und *y*-Koordinaten reagieren.

Lösung

Diese Lösung nutzt LEDs, um Mausbewegungen anzuzeigen. Die Helligkeit der LEDs ändert sich in Reaktion auf die Mausbewegung in *x*- (links und rechts) und *y*- (vor und zurück) Richtung. Das Anklicken der Maustasten legt die aktuelle Position als Referenzpunkt fest (Abbildung 6-16 zeigt den Anschlussplan):

```
/*
  Mouse
  Dieser Arduino-Sketch nutzt die PS2-Maus-Bibliothek
  siehe: http://www.arduino.cc/playground/ComponentLib/Ps2mouse
 */

// PS2-Maus-Bibliothek von:  http://www.arduino.cc/playground/ComponentLib/Ps2mouse
#define WProgram.h Arduino.h
#include <ps2.h>

const int dataPin =  5;
const int clockPin = 6;

const int xLedPin  = 9;
const int yLedPin  = 11;

const int mouseRange = 255;  // Maximaler Bereich der x/y-Werte

char x;           // Von der Maus eingelesene Werte
char y;
byte status;

int xPosition = 0;    // Bei Mausbewegung inkrementierte/dekrementierte Werte
int yPosition = 0;
int xBrightness = 128;  // Basierend auf Mausposition erhöhte/verringerte Werte
int yBrightness = 128;

const byte REQUEST_DATA = 0xeb; // Befehl zum Abruf der Mausdaten

PS2 mouse(clockPin, dataPin);

void setup()
{
 mouseBegin();
}

void loop()
{
 // Daten von Maus einlesen
 mouse.write(REQUEST_DATA); // Daten von Maus anfordern
 mouse.read();    // Ack ignorieren
 status = mouse.read(); // Mausbuttons einlesen
 if(status & 1) // Dieses Bit ist gesetzt, wenn die linke Maustaste gedrückt ist
  xPosition = 0; // x-Position neu ausrichten
 if(status & 2) // Dieses Bit ist gesetzt, wenn die rechte Maustaste gedrückt ist
  yPosition = 0; // y-Position neu ausrichten

 x = mouse.read();
 y = mouse.read();
 if( x != 0 || y != 0)
 {
  // Wenn die Maus bewegt wurde

  xPosition = xPosition + x; // Position akkumulieren
  xPosition = constrain(xPosition,-mouseRange,mouseRange);

  xBrightness = map(xPosition, -mouseRange, mouseRange, 0,255);
```

```
  analogWrite(xLedPin, xBrightness);

  yPosition = constrain(yPosition + y, -mouseRange,mouseRange);
  yBrightness = map(yPosition, -mouseRange, mouseRange, 0,255);
  analogWrite(yLedPin, yBrightness);
 }
}

void mouseBegin()
{
 // Maus zurücksetzen und initialisieren
 mouse.write(0xff);     // Reset
 delayMicroseconds(100);
 mouse.read();        // Ack-Byte
 mouse.read();        // Blank
 mouse.read();        // Blank
 mouse.write(0xf0);     // Modus
 mouse.read();        // Ack
 delayMicroseconds(100);
}
```

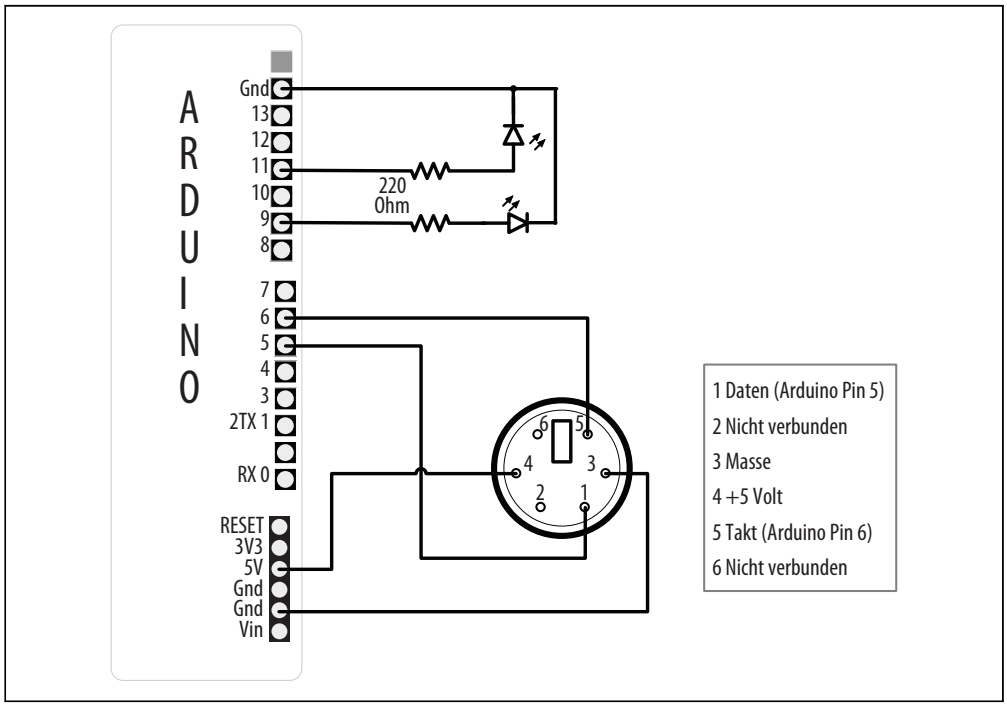


Abbildung 6-16: Anschluss einer Maus und zweier LEDs

Abbildung 6-16 zeigt den weiblichen PS/2-Anschluss von vorne. Wenn Sie keinen solchen Stecker besitzen und sich nicht scheuen, das Ende des Mauskabel abzuschneiden, können Sie sich notieren, welche Kabel mit welchen Pins verbunden sind, und die Kabel mit Steckern verlöten, die Sie direkt auf die richtigen Arduino-Pins aufstecken können.

Diskussion

Verbinden Sie das Maussignal (clock und data) und die Versorgungsanschlüsse mit dem Arduino (siehe Abbildung 6-16). Diese Lösung funktioniert nur mit PS/2-kompatiblen Geräten, d.h., Sie müssen möglicherweise eine ältere Maus auftreiben. Die meisten Mäuse mit dem runden PS/2-Anschluss sollten funktionieren.

Die Funktion `mouseBegin` initialisiert die Maus, damit sie auf Anforderungen von Bewegungsdaten und Button-Status reagiert. Die PS/2-Bibliothek von *http://www.arduino.cc/playground/ComponentLib/Ps2mouse* übernimmt die Low-Level-Kommunikation. Der Befehl `mouse.write` teilt der Maus mit, dass Daten angefordert werden. Der erste Aufruf von `mouse.read` enthält eine Bestätigung (engl. Acknowledgment, kurz Ack), die in diesem Beispiel ignoriert wird. Der nächste Aufruf von `mouse.read` liest den Status der Maustasten ein, und die beiden letzten `mouse.read`-Aufrufe lesen die *x*- und *y*-Bewegung seit der letzten Abfrage ein.

Der Sketch prüft, welche Bits im `status`-Wert gesetzt sind, um zu bestimmen, ob die linke oder rechte Maustaste gedrückt wurde. Die beiden niederwertigsten (rechten) Bits sind `HIGH`, wenn die linke und die rechte Taste gedrückt werden, und das wird mit den folgenden Zeilen überprüft:

```
status = mouse.read(); // Mausbuttons einlesen
if(status & 1) // Dieses Bit ist gesetzt, wenn die linke Maustaste gedrückt ist
 xPosition = 0; // x-Position neu ausrichten
if(status & 2) // Dieses Bit ist gesetzt, wenn die rechte Maustaste gedrückt ist
 yPosition = 0; // y-Position neu ausrichten
```

Die von der Maus eingelesenen `x`- und `y`-Werte enthalten die Bewegung seit der letzten Abfrage und werden zu den Variablen `xPosition` und `yPosition` aufaddiert.

Die Werte von `x` und `y` sind positiv, wenn die Maus nach rechts oder nach vorne bewegt wird. Sie sind negativ, wenn sie nach links oder zurück bewegt wird.

Der Sketch stellt sicher, dass die akkumulierten Werte den definierten Wertebereich (`mouseRange`) nicht überschreiten. Das geschieht mit Hilfe der `constrain`-Funktion:

```
xPosition = xPosition + x; // Position akkumulieren
xPosition = constrain(xPosition,-mouseRange,mouseRange);
```

Die Berechnung der `yPosition` ist eine Kurzform dieser Berechnung. Die Berechnung des `y`-Werts erfolgt hier innerhalb des `constrain`-Aufrufs:

```
yPosition = constrain(yPosition + y,-mouseRange,mouseRange);
```

Die Variablen `xPosition` und `yPosition` werden auf 0 zurückgesetzt, wenn die linke bzw. die rechte Maustaste gedrückt wird.

Die LEDs leuchten entsprechend der Mausposition. Die Helligkeit wird mit `analogWrite` festgelegt – mittlere Helligkeit in der Mitte und zu- oder abnehmende Helligkeit, wenn sich die Mausposition erhöht bzw. verringert.

Die Position kann über den seriellen Monitor ausgegeben werden, indem Sie die folgende Zeile hinter den zweiten Aufruf von `analogWrite()` anhängen:

```
printValues(); // Tasten und x/y-Werte über seriellen Monitor ausgeben
```

Sie müssen außerdem die folgende Zeile in `setup()` einfügen:

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

Zum Schluss müssen Sie den Sketch um die folgende Funktion ergänzen, die die von der Maus empfangenen Werte ausgibt:

```
void printValues()
{
  Serial.println(status, BIN);

  Serial.print("X=");
  Serial.print(x,DEC);
  Serial.print(", Position= ");
  Serial.print(xPosition);
  Serial.print(", Helligkeit= ");
  Serial.println(xBrightness);

  Serial.print("Y=");
  Serial.print(y,DEC);
  Serial.print(", Position= ");
  Serial.print(yPosition);
  Serial.print(", Helligkeit= ");
  Serial.println(yBrightness);
  Serial.println();
}
```

Siehe auch

Geeignete PS/2-Stecker und Breakout-Boards sind *http://www.sparkfun.com/products/8509* und *http://www.sparkfun.com/products/8651*.

6.14 Die Position per GPS bestimmen

Problem

Sie wollen Ihre Position mit Hilfe eines GPS-Moduls bestimmen.

Lösung

Heutzutage stehen eine ganze Reihe Arduino-kompatibler GPS-Module zur Verfügung. Sie nutzen ein vertrautes serielles Interface zur Kommunikation mit dem Host-Mikrocontroller und verwenden ein Protokoll namens *NMEA 0183*. Dieser Industriestandard liefert die GPS-Daten an »Listener«-Einheiten wie den Arduino in für Menschen lesbaren ASCII-»Sätzen« aus. Hier ein Beispiel für einen solchen NMEA-Satz:

```
$GPGLL,4916.45,N,12311.12,W,225444,A,*1D
```

Er beschreibt, unter anderem, eine Position an 49 16.45' nördlicher Breite und 123 11.12' westlicher Länge.

Um die Position zu bestimmen, muss Ihr Arduino-Sketch diese Strings verarbeiten und die relevanten Textstellen in eine numerische Form umwandeln. Code zu entwickeln, der Daten aus NMEA-Sätzen extrahiert, wird (mit Arduinos beschränktem Adressraum) sehr schnell kniffelig und sperrig. Glücklicherweise gibt es eine nützliche Bibliothek, die uns diese Arbeit abnimmt: Mikal Harts TinyGPS. Laden Sie sie von *http://arduiniana.org/* herunter und installieren Sie sie. (Wie man Bibliotheken von Drittanbietern installiert, beschreibt .)

Die allgemeine Strategie für den Einsatz von GPS sieht wie folgt aus:

1. Verbinden Sie das GPS-Gerät physikalisch mit dem Arduino.
2. Lesen Sie die seriellen NMEA-Daten vom GPS-Gerät aus.
3. Verarbeiten Sie die Daten, um die Position zu ermitteln.

Mit TinyGPS machen Sie Folgendes:

1. Verbinden Sie das GPS-Gerät physikalisch mit dem Arduino.
2. Erzeugen Sie ein TinyGPS-Objekt.
3. Lesen Sie die seriellen NMEA-Daten vom GPS-Gerät ein.
4. Verarbeiten Sie jedes Byte mit der TinyGPS-Methode `encode()`.
5. Rufen Sie periodisch die TinyGPS-Methode `get_position()` auf, um die Position zu bestimmen.

Der folgende Sketch zeigt, wie man Daten von einem GPS-Gerät erfasst, das mit der seriellen Schnittstelle des Arduino verbunden ist. Er schaltet die LED an Pin 13 ein, sobald sich das Gerät in der südlichen Hemisphäre befindet:

```
// Einfacher Sketch erkennt die südliche Hemisphäre
// Annahmen: LED an Pin 13, GPS an seriellen Hardware-Pins 0/1
#include "TinyGPS.h"

TinyGPS gps; // TinyGPS-Objekt erzeugen

#define HEMISPHERE_PIN 13

void setup()
{
 Serial.begin(4800); // GPS-Geräte arbeiten oft mit 4800 Baud
 pinMode(HEMISPHERE_PIN, OUTPUT);
 digitalWrite(HEMISPHERE_PIN, LOW); // LED zu Beginn ausschalten
}
void loop()
{
 while (Serial.available())
 {
  int c = Serial.read();
  // encode() Für jedes Byte aurufen
  // Neue Position bestimmen, wenn encode() "wahr" zurückgibt
```

```
    if (gps.encode(c))
    {
     long lat, lon;
     gps.get_position(&lat, &lon);
     if (lat < 0) // Südliche Hemisphäre?
      digitalWrite(HEMISPHERE_PIN, HIGH);
     else
      digitalWrite(HEMISPHERE_PIN, LOW);
    }
   }
  }
```

Wir starten die serielle Kommunikation, indem wir die vom GPS benötigte Geschwindigkeit einstellen. Weitere Informationen zur seriellen Kommunikation mit dem Arduino finden Sie in Kapitel 4.

Eine 4800-Baud-Verbindung wird mit dem GPS hergestellt. Sobald die Bytes eingehen, werden sie von `encode()` verarbeitet, die die NMEA-Daten verarbeitet. Ein `true` von `encode()` zeigt an, dass TinyGPS einen vollständigen »Satz« erfolgreich verarbeitet hat und neue Positionsdaten zur Verfügung stehen könnten. Das ist ein guter Zeitpunkt, um die aktuelle Position mit `get_position()` zu bestimmen.

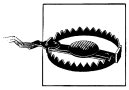
`get_position()` gibt den zuletzt erkannten Breiten- und Längengrad zurück. Das Beispiel untersucht den Breitengrad. Ist er kleiner als 0, also südlich des Äquators, wird die LED eingeschaltet.

Diskussion

Der Anschluss einer GPS-Einheit an einen Arduino ist in der Regel ganz einfach. Dazu müssen üblicherweise nur zwei oder drei Leitungen vom GPS mit den Eingangspins des Arduino verbunden werden. Das beliebte GPS-Modul USGlobalSat EM-406A können Sie zum Beispiel so anschließen, wie in Tabelle 6-1 zu sehen.

Tabelle 6-1: Anschluss eines EM-406A-GPS

EM-406A-Anschluss	Arduino-Pin
GND	Gnd
VIN	+Vcc
RX	TX (Pin 1)
TX	RX (Pin 0)
GND	Gnd



Einige GPS-Module arbeiten mit RS-232-Spannungspegeln, die mit der TTL-Logik des Arduino nicht kompatibel sind und das Board ernsthaft beschädigen können. Wenn Ihr GPS mit RS-232-Pegeln arbeitet, müssen Sie eine Umwandlungslogik zwischenschalten, z.B. einen Chip wie den MAX232.

Der Code in diesem Rezept geht davon aus, dass das GPS direkt mit dem fest eingebauten seriellen Port des Arduino verbunden ist, doch das ist üblicherweise nicht das beste Design. Bei vielen Projekten wird der serielle Hardware-Port zur Kommunikation mit einem PC oder anderen Peripheriegeräten genutzt und kann vom GPS nicht verwendet werden. In solchen Fällen wählen Sie in anderes Paar Digitalpins und nutzen einen seriellen »Software«-Port, um mit dem GPS zu kommunizieren.

SoftwareSerial ist eine Bibliothek, die eine serielle Schnittstelle emuliert und mit der Arduino-IDE mitgeliefert wird. Wenn Sie mit einer Arduino-Version vor 1.0 arbeiten, müssen Sie eine Bibliothek namens NewSoftSerial verwenden, die ebenfalls auf *http://arduiniana.org/* zu finden ist. Detaillierte Informationen zu seriellen Software-Schnittstellen finden Sie in 4.13 und 4.14.

Sie können die TX-Leitung des GPS mit Arduino-Pin 2 und die RX-Leitung mit Pin 3 verbinden und so den seriellen Hardware-Port für Debugging-Zwecke freimachen (siehe Abbildung 4-7). Wenn wir den obigen Sketch so modifizieren, dass er SoftwareSerial nutzt, um das GPS abzufragen, können wir über den seriellen Monitor TinyGPS in Aktion beobachten:

```
// Einfacher Sketch erkennt die südliche Hemisphäre
// Annahmen: LED an Pin 13, GPS an Pins 2/3
// (Optional) Serielle Debugging-Konsole an Hardware-Port 0/1

#include "TinyGPS.h"
#include "SoftwareSerial.h"

#define HEMISPHERE_PIN 13
#define GPS_RX_PIN 2
#define GPS_TX_PIN 3

TinyGPS gps; // create a TinyGPS object
SoftwareSerial ss(GPS_RX_PIN, GPS_TX_PIN); // SoftSerial-Objekt erzeugen

void setup()
{
 Serial.begin(9600); // Für Debugging
 ss.begin(4800); // SoftSerial-Objekt spricht mit GPS
 pinMode(HEMISPHERE_PIN, OUTPUT);
 digitalWrite(HEMISPHERE_PIN, LOW); // LED zu Beginn ausschalten
}
void loop()
{
 while (ss.available())
 {
  int c = ss.read();
  Serial.write(c); // NMEA-Daten zu Debug-Zwecken ausgeben
  // Jedes Byte mit encode() verarbeiten
  // Neue Position bestimmen, wenn encode() "wahr" zurückgibt
  if (gps.encode(c))
  {
   long lat, lon;
   unsigned long fix_age;
  gps.get_position(&lat, &lon, &fix_age);
```

```
    if (fix_age == TinyGPS::GPS_INVALID_AGE )
     Serial.println("Noch keine Daten erkannt!");
    else if (fix_age > 2000)
     Serial.println("Daten sind veraltet!");
    else
     Serial.println("Breiten- und Längengrad gültig!");

     Serial.print("Breite: ");
     Serial.print(lat);
     Serial.print(" Länge: ");
     Serial.println(lon);
     if (lat < 0) // Südliche Hemisphäre?
      digitalWrite(HEMISPHERE_PIN, HIGH);
     else
      digitalWrite(HEMISPHERE_PIN, LOW);
    }
   }
}
```

Beachten Sie, dass der serielle Monitor und das GPS verschiedene Baudraten verwenden können.

Dieser Sketch entspricht genau unserem früheren Beispiel, ist aber deutlich einfacher zu debuggen. Sie können jederzeit einen Monitor an den eingebauten seriellen Port anschließen und sich die NMEA- und TinyGPS-Daten ansehen.

Sobald es eingeschaltet wird, beginnt die GPS-Einheit mit der Übertragung von NMEA-Sätzen. Gültige Positionsdaten enthaltende Datensätze werden aber nur übertragen, nachdem das GPS eine Verbindung hergestellt hat. Dazu muss das Gerät den Himmel »sehen« können, und die ganze Sache kann bis zu zwei Minuten dauern. Schlechtes Wetter, Gebäude und andere Hindernisse können das GPS bei der Positionsbestimmung behindern. Woher weiß der Sketch also, ob TinyGPS gültige Positionsdaten liefert? Die Antwort ist der dritte Parameter von `get_position()`, der optionale `fix_age`.

Wenn Sie einen Zeiger auf eine unsigned `long`-Variable als dritten Parameter an `get_position()` übergeben, füllt TinyGPS sie mit der Zeit in Millisekunden, zu der die letzte gültige Position bestimmt wurde. Siehe auch Rezept 2.11. Der Wert `0xFFFFFFFF` (symbolisch `GPS_INVALID_AGE`) bedeutet, dass TinyGPS noch keine gültigen Positionsdaten erfasst hat. In diesem Fall ist auch der zurückgelieferte Breiten- und Längengrad ungültig (`GPS_INVALID_ANGLE`).

Im normalen Betrieb können Sie recht kleine Werte für `fix_age` erwarten. Moderne GPS-Geräte liefern Positionsdaten ein- bis fünfmal pro Sekunde, d.h., ein `fix_age`-Wert von über 2000 ms deutet auf ein Problem hin. Vielleicht befinden Sie sich gerade in einem Tunnel oder ein defektes Kabel verfälscht den NMEA-Datenstrom, wodurch die Prüfsumme nicht mehr stimmt. (Eine Prüfsummenberechnung stellt sicher, dass die Daten nicht beschädigt sind.) So oder so gibt ein hoher `fix_age`-Wert an, dass die von `get_position()` zurückgelieferten Koordinaten veraltet sind. Das nachfolgende Code-Beispiel zeigt, wie Sie mit `fix_age` sicherstellen können, dass die Positionsdaten aktuell sind:

```
long lat, lon;
unsigned long fix_age;
```

```
gps.get_position(&lat, &lon, &fix_age);
if (fix_age == TinyGPS::GPS_INVALID_AGE)
 Serial.println("Noch keine Daten erkannt!");
else if (fix_age > 2000)
 Serial.println("Daten sind veraltet!");
else
 Serial.println("Breiten- und Längengrad gültig!");
```

Siehe auch

TinyGPS können Sie unter *http://arduiniana.org/libraries/tinygps* herunterladen.

Weiterführende Informationen zum NMEA finden Sie im Wikipedia-Artikel unter *http://en.wikipedia.de/wiki/NMEA*.

Verschiedene Unternehmen verkaufen GPS-Module, die sich gut für TinyGPS und Arduino eignen. Sie unterscheiden sich hauptsächlich im Stromverbrauch, in der Spannung, der Genauigkeit, der physikalischen Schnittstelle und darin, ob sie serielles NMEA unterstützen. SparkFun (*http://www.sparkfun.com*) bietet eine große Auswahl an GPS-Modulen und einen ausgezeichneten Leitfaden für Käufer an.

GPS-Technik hat eine Vielzahl kreativer Arduino-Projekte inspiriert. Ein sehr populäres Beispiel ist der GPS-Datenlogger, bei dem ein bewegliches Gerät seine Positionsdaten in regelmäßigen Intervallen in das Arduino-EEPROM oder anderen Onboard-Speicher schreibt. Ein Beispiel ist das Breadcrumbs-Projekt unter *http://code.google.com/p/breadcrumbs/wiki/UserDocument*. Ladyada stellt ein beliebtes GPS-Datenlogger-Shield her. Siehe *http://www.ladyada.net/make/gpsshield/*.

Zu den weiteren interessanten GPS-Projekten gehören Modellflugzeuge und Helikopter, die sich unter der Kontrolle von Arduino-Software selbst zu vorprogrammierten Zielen bewegen. Mikal Hart hat eine GPS-basierte »Schatztruhe« mit einem internen Schloss entwickelt, die sich erst öffnen lässt, wenn sich die Kiste an einem bestimmten Ort befindet. Siehe *http://arduiniana.org*.

6.15 Bewegungen mit einem Gyroskop erkennen

Problem

Sie wollen auf einen Rotationswinkel reagieren. Auf diese Weise können Sie ein Fahrzeug oder einen Roboter auf gerader Linie halten oder um einen gewünschten Winkel drehen.

Lösung

Gyroskope liefern Daten zum Rotationswinkel (im Gegensatz zu Beschleunigungsmessern, die Änderungen an der Geschwindigkeit messen). Die meisten günstigen Gyroskope verwenden eine Analogspannung proportional zum Rotationswinkel, auch wenn einige ihre Werte über I2C anbieten (mehr über den Einsatz von I2C zur Gerätekommunikation finden Sie in Kapitel 13). Dieses Rezept arbeitet mit einem Gyroskop, dessen Analog-

ausgang proportional zum Rotationswinkel ist. Abbildung 6-17 zeigt die Verschaltung eines LY530AL-Breakout-Boards von SparkFun. Viele kostengünstige Gyroskope (wie das hier verwendete) sind 3,3-V-Elemente und dürfen nicht direkt mit der 5-V-Spannung verbunden werden.

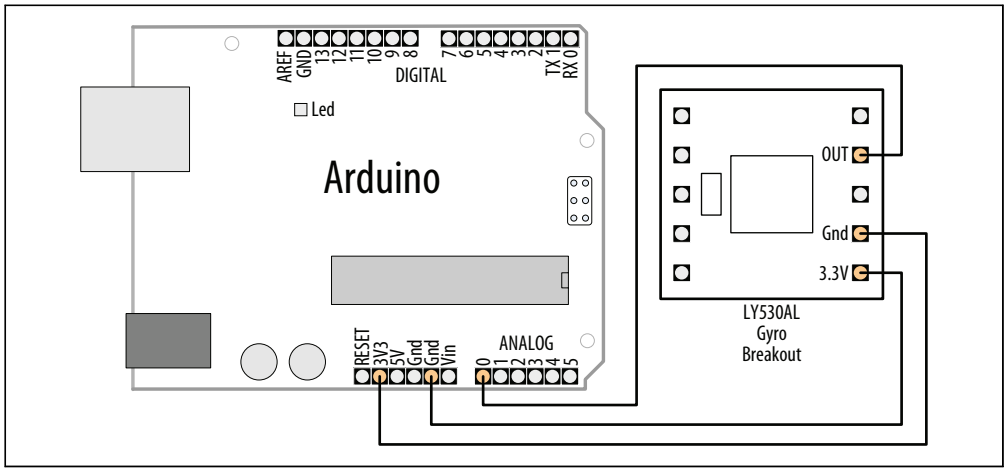
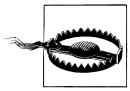


Abbildung 6-17: LY530AL-Gyroskop an 3,3-V-Pin



Überprüfen Sie die maximale Spannung Ihres Gyroskops, bevor Sie es mit Spannung versorgen. Ein 3,3-V-Gyroskop an 5V anzuschließen, kann das Bauteil beschädigen.

Der OUT-Anschluss des Gyroskops ist ein analoger Ausgang und wird mit dem analogen Eingang 0 des Arduino verbunden:

```
/*
 gyro Sketch
 Gibt den Rotationswinkel über den seriellen Monitor aus
 */


const int inputPin = 0;  // Analoger Eingang 0
int rotationRate = 0;

void setup()
{
 Serial.begin(9600);  // Setzt seriellen Port auf 9600 Baud
}

void loop()
{
 rotationRate = analogRead(inputPin); // Gyroskop einlesen
 Serial.print("Gyroskopwert ist ");
 Serial.println(rotationRate);
 delay(100);              // 100ms warten
}
```

Diskussion

Der `loop`-Code liest den Gyroskopwert über den Analogpin 0 ein und gibt ihn über den seriellen Monitor aus.

Verwendung des älteren LISY300AL-Gyroskops

Die alte Auflage hat das LISY300AL-Gyroskop genutzt, das nur noch schwer zu bekommen ist. Sollten Sie noch eins besitzen, können Sie den gleichen Sketch wie oben verwenden, wenn Sie den Power Down-Pin (PD) mit Masse verbinden. Oder, noch besser, Sie schließen den PD-Pin an einen Arduino-Pin an, damit Sie das Gyroskop aus dem Sketch heraus ein- und ausschalten können. Abbildung 6-18 zeigt den Anschluss des LISY3000AL.

Die PD-Verbindung erlaubt es, das Gyroskop in den Stromsparmodus zu schalten, und ist mit Analogpin 1 verbunden (in diesem Sketch wird er als digitaler Ausgang verwendet). Sie können PD an jeden Digitalpin anschließen. Der Pin wurde hier wegen der sauberen Verschaltung gewählt. Der obige Code kann wie folgt modifiziert werden, um den PD-Pin zu kontrollieren:

```
const int inputPin = 0;     // Analoger Eingang 0
const int powerDownPin = 15; // Analoger Eingang 1 ist  digitaler Eingang 15

int rotationRate = 0;

void setup()
{
 Serial.begin(9600);       // Serieller Port auf 9600 Baud
 pinMode(powerDownPin, OUTPUT);
 digitalWrite(powerDownPin, LOW); // Gyroskop nicht im Stromsparmodus
}

// loop-Code wie oben
```

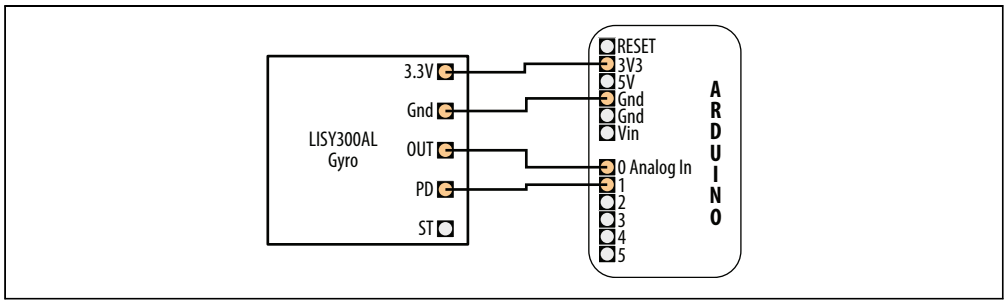


Abbildung 6-18: Anschluss des LISY3000AL-Gyroskops

Wenn Sie das Gyroskop nicht in den Stromsparmodus schalten müssen, verbinden Sie PD einfach mit Masse (PD `LOW` ist an, PD `HIGH` ist Stromsparmodus).

Analoge Eingangspins können auch als Digitalpins verwendet werden (aber nicht andersherum). Der analoge Eingang 0 ist der Digitalpin 14, der analoge Eingang 1 ist der Digitalpin 15 und so weiter. Mit Arduino 1.0 wurden neue Definitionen eingeführt, die es Ihnen erlauben, den analogen Eingang 0 als `A0` anzusprechen, den analogen Eingang 1 als `A1` etc.

Rotation mit dem ITG-3200 in drei Dimensionen messen

Der ITG-3200 ist ein 3-Achs-Gyroskop mit ausgezeichnetem Preis-Leistungs-Verhältnis. Selbst wenn Sie nur zwei Achsen messen müssen, ist er häufig eine bessere Wahl als der LY530ALH, wenn Sie eine hohe Messgenauigkeit benötigen oder es mit hohen Rotationsgeschwindigkeiten (bis zu 2000° pro Sekunde) zu tun haben. Es handelt sich um ein 3,3-V-Bauelement mit I2C-Anschluss. Wenn Sie nicht mit einem 3,3-V-Arduino arbeiten, brauchen Sie einen Pegelwandler, um die SCL- und SDA-Pins des Gyroskops zu schützen. Mehr über I2C und die Verwendung von 3,3-V-Geräten finden Sie in der Einführung von Kapitel 13.

Das Breakout-Board von SparkFun (SEN-09801) macht den Anschluss einfach (siehe Abbildung 6-19), doch Sie dürfen nicht vergessen, den CLK-Jumper auf der Unterseite des Boards zu überbrücken, der das interne Clock-Signal aktiviert.

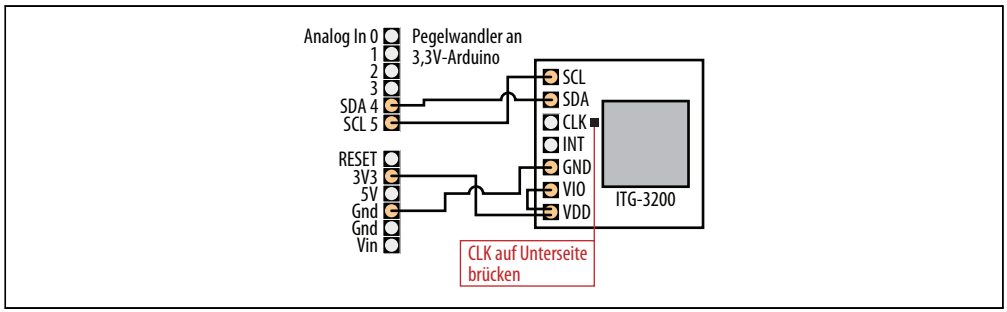


Abbildung 6-19: Anschluss des ITG-3200 an ein 3,3V-Board

Der nachfolgende Sketch gibt die Werte der x-, y- und z-Achsen durch Kommata getrennt aus:

```
/*
 ITG-3200 Beipiel-Sketch
 Basiert auf SparkFun Quick Start Guide: http://www.sparkfun.com/tutorials/265
 */
#include <Wire.h>

const int itgAddress = 0x69;

// ITG-3200-Konstanten - siehe Datenblatt
const byte SMPLRT_DIV= 0x15;
const byte DLPF_FS  = 0x16;
const byte INT_CFG  = 0x17;
```

```
const byte PWR_MGM  = 0x3E;
const byte GYRO_X_ADDRESS = 0x1D; // GYRO_XOUT_H
const byte GYRO_Y_ADDRESS = 0x1F; // GYRO_YOUT_H
const byte GYRO_Z_ADDRESS = 0x21; // GYRO_ZOUT_H

// Konfigurationseinstellungen; Details auf dem Datenblatt
const byte DLPF_CFG_0   = 0x1;
const byte DLPF_CFG_1   = 0x2;
const byte DLPF_CFG_2   = 0x4;
const byte DLPF_FS_SEL_0 = 0x8;
const byte DLPF_FS_SEL_1 = 0x10;

void setup()
{
 Serial.begin(9600);
 Wire.begin();

 //Gyroskop konfigurieren
 //Gyroskop-Bereich der Ausgänge auf +/-2000 Grad pro Sekunde einstellen
 itgWrite(DLPF_FS, (DLPF_FS_SEL_0|DLPF_FS_SEL_1|DLPF_CFG_0));
 //Sample-Rate ist 100 hz
 itgWrite(SMPLRT_DIV, 9);
}

//x-,y- und z-Werte einlesen und über seriellen Monitor ausgeben
void loop()
{
 //Variablen für die Ausgänge.
 int xRate, yRate, zRate;

 //x-,y- und z-Werte vom Gyroskop einlesen.
 xRate = readAxis(GYRO_X_ADDRESS);
 yRate = readAxis(GYRO_Y_ADDRESS);
 zRate = readAxis(GYRO_Z_ADDRESS);

 //Werte über seriellen Monitor ausgeben
 int temperature = 22;
 Serial.print(temperature);
 Serial.print(',');
 Serial.print(xRate);
 Serial.print(',');
 Serial.print(yRate);
 Serial.print(',');
 Serial.println(zRate);

 //10ms warten, bevor die nächsten Werte gelesen werden.
 delay(10);
}

//Die übergebenen Daten in die angegebenen itg-3200-Register schreiben
void itgWrite(char registerAddress, char data)
{

 Wire.beginTransmission(itgAddress); // Sendesequenz initiieren
 Wire.write(registerAddress);     // Zu schreibende Registeradresse
 Wire.write(data);           // Zu schreibende Daten
 Wire.endTransmission();        // Hier werden die Daten dann gesendet
```

```
}

//Daten aus dem angegebnen ITG-3200-Register einlesen und Wert zurückgeben.
unsigned char itgRead(char registerAddress)
{
 //Diese Variable enthält die vom I2C-Gerät eingelesenen Daten.
 unsigned char data=0;

 Wire.beginTransmission(itgAddress);
 Wire.write(registerAddress); //Registeradresse senden
 Wire.endTransmission();    //Ende der Kommunikationssequenz.

 Wire.beginTransmission(itgAddress);
 Wire.requestFrom(itgAddress, 1);  //Gerätedaten abrufen

 if(Wire.available()){  // Auf Antwort des Geräts warten
  data = Wire.read();  // Daten einlesen
 }

 Wire.endTransmission();  //Ende der Kommunikationssequenz
 return data;  //Daten zurückgeben
}

// x-,y- oder z-Wert des Gyroskops einlesen.
// axisRegAddress wählt einzulesende Achse.
int readAxis(byte axisRegAddress)
{
 int data=0;
 data = itgRead(axisRegAddress)<<8;
 data |= itgRead(axisRegAddress + 1);
 return data;
}
```

Siehe auch

Mehr über I2C erfahren Sie in Kapitel 13.

In Rezept 13.1 erfahren Sie mehr zu diesem Thema.

Eine SparkFun-Einführung zum ITG-3200 finden Sie unter *http://www.sparkfun.com/tutorials/265*.

6.16 Richtung bestimmen

Problem

Ihr Sketch soll die Richtung mit Hilfe eines elektronischen Kompasses bestimmen.

Lösung

Dieses Rezept nutzt das HM55B-Kompassmodul von Parallax (#29123); Abbildung 6-20 zeigt die Anschlüsse:

```
/*
 HM55bCompass Sketch
 Implementiert serielles 'Software-SPI' mit Arduinos Bit-Operatoren
 (siehe Rezept 3.13)
 Gibt Kompass-Winkel über seriellen Monitor aus
 */

const int enablePin = 2;
const int clockPin  = 3;
const int dataPin   = 4;

// Befehlscodes (aus HM55B-Datenblatt)
const byte COMMAND_LENGTH = 4;      // Anzahl der Bits in einem Befehl
const byte RESET_COMMAND = B0000;    // Chip zurücksetzen
const byte MEASURE_COMMAND = B1000;  // Messung starten
const byte READ_DATA_COMMAND = B1100; // Daten und Ende-Flag einlesen
const byte MEASUREMENT_READY = B1100; // Wert, der nach Abschluss der Messung zurückgegeben wird


int angle;

void setup()
{
 Serial.begin(9600);
 pinMode(enablePin, OUTPUT);
 pinMode(clockPin, OUTPUT);
 pinMode(dataPin, INPUT);
 reset(); // Kompass-Modul zurücksetzen
}

void loop()
{
 startMeasurement();
 delay(40); // Warten, bis Daten bereit
 if (readStatus()==MEASUREMENT_READY); // Prüfen, ob Daten bereit
 {
  angle = readMeasurement();      //Messwert lesen und Winkel berechnen
  Serial.print("Winkel = ");
  Serial.println(angle); // Winkel ausgeben
 }
 delay(100);
}

void reset()
{
 pinMode(dataPin, OUTPUT);
 digitalWrite(enablePin, LOW);
 serialOut(RESET_COMMAND, COMMAND_LENGTH);
 digitalWrite(enablePin, HIGH);
}

void startMeasurement()
{
 pinMode(dataPin, OUTPUT);
 digitalWrite(enablePin, LOW);
 serialOut(MEASURE_COMMAND, COMMAND_LENGTH);
 digitalWrite(enablePin, HIGH);
```

```
}

int readStatus()
{
 int result = 0;
 pinMode(dataPin, OUTPUT);
 digitalWrite(enablePin, LOW);
 serialOut(READ_DATA_COMMAND, COMMAND_LENGTH);
 result = serialIn(4);
 return result;  // Status zurückgeben
}

int readMeasurement()
{
 int X_Data = 0;
 int Y_Data = 0;
 int calcAngle = 0;
 X_Data = serialIn(11); // Feldstärke in X
 Y_Data = serialIn(11); // und Richtung in Y
 digitalWrite(enablePin, HIGH); // Chip deaktivieren
 calcAngle = atan2(-Y_Data , X_Data) / M_PI * 180; // Winkel ist atan(-y/x)
 if(calcAngle < 0)
  calcAngle = calcAngle + 360; // Winkel von 0 bis 359 statt +/- 180
 return calcAngle;
}

void serialOut(int value, int numberOfBits)
{
 for(int i = numberOfBits; i > 0; i--) // MSB zuerst schreiben
 {
  digitalWrite(clockPin, LOW);
  if(bitRead(value, i-1) == 1)
   digitalWrite(dataPin, HIGH);
  else
   digitalWrite(dataPin, LOW);
  digitalWrite(clockPin, HIGH);
 }
}

int serialIn(int numberOfBits)
{
 int result = 0;

 pinMode(dataPin, INPUT);
 for(int i = numberOfBits; i > 0; i--) // MSB zuerst lesen
 {
  digitalWrite(clockPin, HIGH);
  if (digitalRead(dataPin) == HIGH)
   result = (result << 1) + 1;
  else
   result = (result << 1) + 0;
  digitalWrite(clockPin, LOW);
 }

 // Wandelt das Ergebnis in ein negative Zweierkomplement um,
 // wenn das höchstwertige Bit in den 11-Bit-Daten 1 ist
 if(bitRead(result, 11) == 1)
```

```
  result = (B11111000 << 8) | result; // Zweierkomplement-Negation

 return result;
}
```

Diskussion

Das Kompass-Modul misst die Magnetfeld-Intensität an zwei lotrechten Achsen (x und y). Diese Werte variieren, wenn sich die Richtung im Bezug auf das Erdmagnetfeld (magnetische Nordrichtung) ändert.

Das Datenblatt des Bauelements sagt Ihnen, welche Werte gesendet werden müssen, um den Kompass zurückzusetzen, und wie Sie prüfen können, ob gültige Daten vorliegen (ist dass der Fall, werden sie gesendet).

Der Sketch verwendet die Funktionen `serialIn()` und `serialOut()`, um die Pin-Operationen durchzuführen, die Nachrichten senden und empfangen.

Das Kompass-Modul wird in der `reset()`-Funktion (die von `setup()` aufgerufen wird) in einen definierten Anfangszustand gebracht. Die Funktion `startMeasurement()` leitet die Messung ein, und nach einer kurzen Verzögerung zeigt die Funktion `readStatus()` an, ob die Daten verfügbar sind. Der Wert `0` wird zurückgegeben, wenn die Messung noch nicht bereit ist, bzw. `12` (binär 1100), wenn der Kompass bereit ist, Daten zu übertragen.

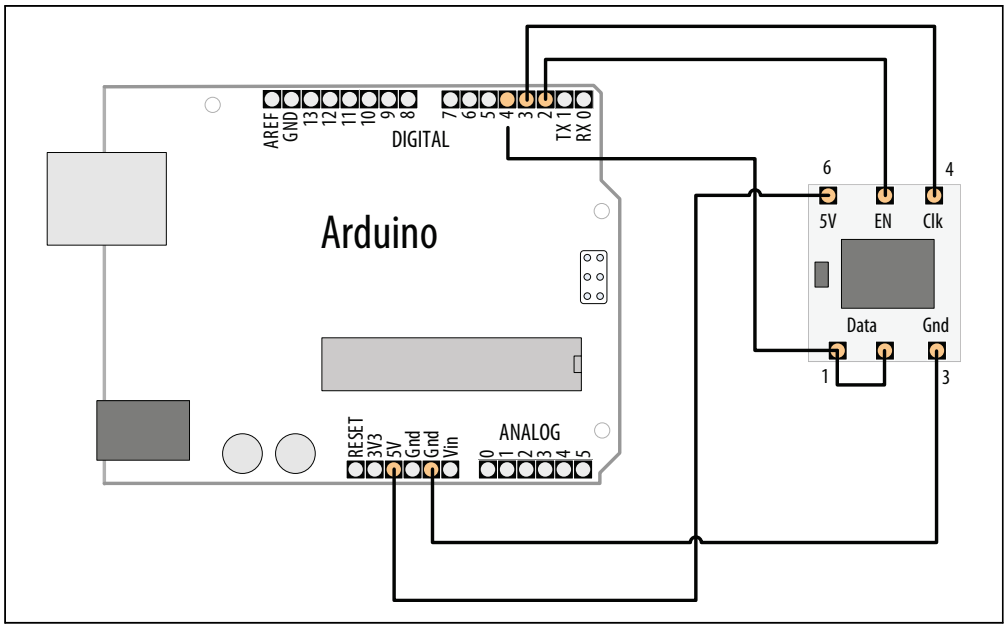


Abbildung 6-20: Anschluss eines HM55B-Kompasses

Elf Datenbits werden in die Variablen `X_Data` und `Y_Data` eingelesen. Wenn Sie ein anderes Bauelement verwenden, müssen Sie auf dem Datenblatt nachsehen, wie viele Bits gesen-

det und in welchem Format sie übertragen werden. X_Data und Y_Data speichern die Magnetfeld-Messwerte, und der Winkel zum magnetischen Nordpol wird wie folgt berechnet: Bogenmaß = arctan(–y/x)

Der Sketch implementiert die Berechnung in der folgenden Zeile:

```
calcAngle = atan2(-Y_Data , X_Data) / M_PI * 180; // Winkel ist atan(-y/x)
```

Damit ein Servo-Motor der Richtung des Kompasses über die ersten 180 Grad folgt, fügen Sie Folgendes hinzu:

```
#include <Servo.h>
Servo myservo;
```

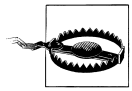
In setup:

```
 myservo.attach(8);
```

Und in loop nach der Berechnung des Winkels:

```
// Servo wird auf 180 Grad beschränkt
  angle = constrain(angle, 0,180);
myservo.write(angle);
```

Richtungssensoren werden verstärkt in Smartphones genutzt. Folgerichtig stehen immer leistungsfähigere und kostengünstigere Bauelemente zur Verfügung. Der folgende Sketch nutzt einen solchen Chip: den 3,3-V-HMC5883L-I2C-Magnetometer. Breakout-Boards sind für dieses Bauteil verfügbar, etwa das SEN-10530 von SparkFun. Verbinden Sie die GND- und VCC-Pins mit Masse und dem 3,3-V-Pin. Die SDA- und SCL- Pins werden mit den Arduino-Pins 4 und 5 verbunden (wie man I2C-Geräte mit dem Arduino nutzt, erfahren Sie in Kapitel 13). Wenn Sie den HMC5883L mit einem 5-V-Arduino nutzen wollen, dann können Sie in Rezept 13.1 nachschauen, wie man einen Pegelwandler nutzt.



Wenn Sie den HMC5883L direkt mit den Arduino-Pins eines normalen 5-V-Boards verbinden, können Sie den HMC5883L-Chip ernsthaft beschädigen.

```
/*
 Verwendet den HMC5883L, um das Erdmagnetfeld an den x-, y- und z-Achsen zu messen
 Gibt die Richtung als Winkel zwischen 0 und 359 Grad an
*/

#include <Wire.h> //I2C-Arduino-Bibliothek

const int hmc5883Address = 0x1E; //0011110b, I2C-7Bit-Adresse des HMC5883
const byte hmc5883ModeRegister    = 0x02;
const byte hmcContinuousMode      = 0x00;
const byte hmcDataOutputXMSBAddress = 0x03;


void setup(){
 //Serielle Schnittstelle und I2C-Kommunikation initialisieren
 Serial.begin(9600);
 Wire.begin();
```

```
 //HMC5883 in den richtigen Betriebsmodus schalten
 Wire.beginTransmission(hmc5883Address); //Kommunikation mit HMC5883 starten
 Wire.write(hmc5883ModeRegister); //Modusregister wählen
 Wire.write(hmcContinuousMode);   //Fortlaufende Messung
 Wire.endTransmission();
}

void loop(){

 int x,y,z; //Daten der drei Achsen

 //HMC5883 anweisen, mit dem Einlesen der Daten zu beginnen
 Wire.beginTransmission(hmc5883Address);
 Wire.write(hmcDataOutputXMSBAddress); //Wähle Register 3, X-MSB-Register
 Wire.endTransmission();

 //Daten aller Achsen einlesen, 2 Register pro Achse
 Wire.requestFrom(hmc5883Address, 6);
 if(6<=Wire.available()){
  x = Wire.read()<<8; //X-msb
  x |= Wire.read(); //X-lsb
  z = Wire.read()<<8; //Z-msb
  z |= Wire.read(); //Z-lsb
  y = Wire.read()<<8; //Y-msb
  y |= Wire.read(); //Y-lsb
 }

 //Werte aller Achsen ausgeben
 Serial.print("x: ");
 Serial.print(x);
 Serial.print(" y: ");
 Serial.print(y);
 Serial.print(" z: ");
 Serial.print(z);

 int angle = atan2(-y , x) / M_PI * 180; // Winkel ist atan(-y/x)
 if(angle < 0)
   angle = angle  + 360; // Winkel von 0 bis 359 statt +/- 180
 Serial.print(" Richtung = ");
 Serial.println(angle);

 delay(250);
}
```

6.17 Daten von einem Spiele-Controller (PlayStation) einlesen

Problem

Sie wollen auf Joystick-Position und Tastendrücke eines Spiele-Controllers reagieren.

Lösung

Das Rezept nutzt einen Controller der Sony PlayStation 2 und die PSX-Bibliothek von *http://www.arduino.cc/playground/Main/PSXLibrary*. Das Schaltdiagramm ist in Abbildung 6-21 zu sehen.

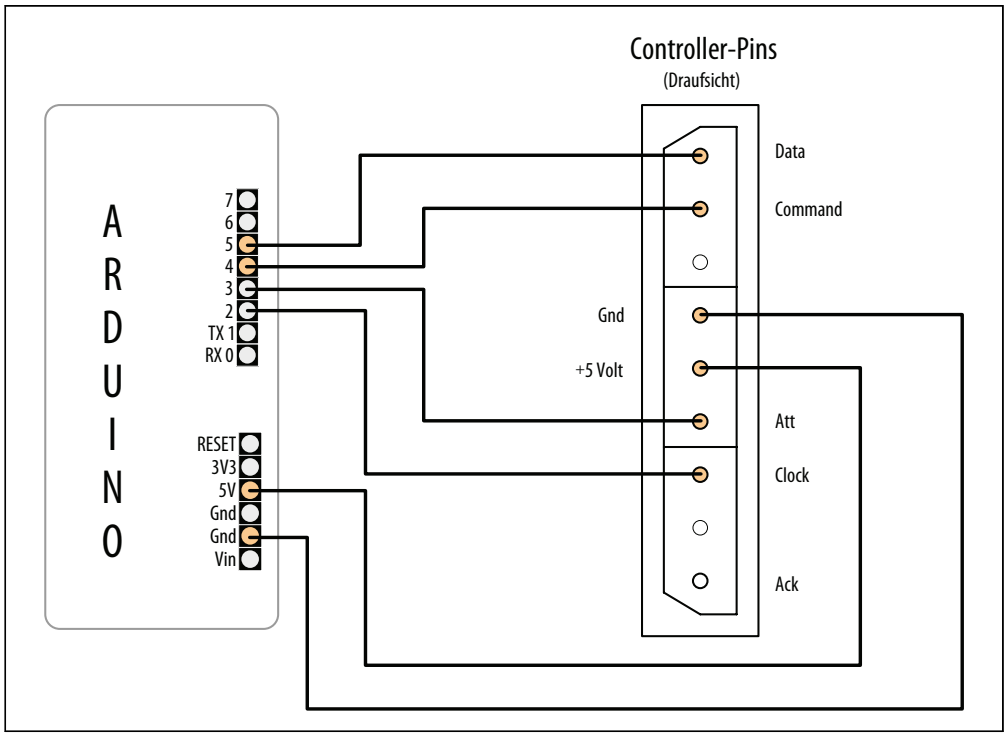


Abbildung 6-21: Anschluss eines PlayStation-Controllers an den Arduino

Der Sketch verwendet den seriellen Monitor, um die gedrückte Tasten auszugeben:

```
/*
 * PSX Sketch
 *
 * Joystick- und Tastenwerte ausgeben.
 * Nutzt die PSX-Bibliothek von Kevin Ahrendt
 * http://www.arduino.cc/playground/Main/PSXLibrary
 */

#include <Psx.h>              // PSX-Bibliothek einbinden

Psx Psx;                  // Instanz der PSX-Bibliothek erzeugen
const int dataPin  = 5;
const int cmndPin  = 4;
const int attPin   = 3;
const int clockPin = 2;
const int psxDelay = 50;      // Verzögerung in Mikrosekunden
```

```
unsigned int data = 0;       // Vom Controller zurückgelieferte Daten

void setup()
{
 // PSX-Bibliothek initialisieren
 Psx.setupPins(dataPin, cmndPin, attPin, clockPin, psxDelay);
 Serial.begin(9600); // Ergebnisse erscheinen auf dem seriellen Monitor
}


void loop()
{
 data = Psx.read();     // Tasten-Daten des PSX-Controllers einlesen

 // Tastenbits prüfen, um Tastendruck zu erkennen
 if(data & psxLeft)
  Serial.println("Links-Taste");
 if(data & psxDown)
  Serial.println("Ab-Taste");
 if(data & psxRight)
  Serial.println("Rechts-Taste");
 if(data & psxUp)
  Serial.println("Auf-Taste");
 if(data & psxStrt)
  Serial.println("Start-Taste");
 if(data & psxSlct)
  Serial.println("Select-Taste");

 delay(100);
}
```

Diskussion

Spiele-Controller stellen Informationen auf unterschiedliche Art und Weise zur Verfügung. Die neuesten Controller enthalten Chips, die die Taster- und Joystick-Werte des Controllers einlesen und diese Informationen über ein Protokoll weitergeben, das je nach Spieleplattform unterschiedlich ist. Ältere Controller greifen eher direkt auf Taster und Joystick zu, und die Stecker haben entsprechend viele Anschlüsse. Die neueste Generation der Spieleplattformen nutzt USB-Verbindungen, und die benötigen eine entsprechende Hardware-Unterstützung, wie etwa ein USB-Host-Shield.

Siehe auch

Rezept 4.1; Rezept 4.11

PlayStation-Controller- Protokoll: *http://www.gamesx.com/controldata/psxcont/psxcont.htm*

6.18 Beschleunigung messen

Problem

Sie wollen auf Beschleunigung reagieren, z.B. um den Anfang oder das Ende einer Bewegung zu erkennen. Oder Sie wollen bestimmen, wie etwas im Bezug auf die Erdoberfläche ausgerichtet ist (Beschleunigungsmessung infolge der Gravitation).

Lösung

Wie bei vielen der in diesem Kapitel behandelten Sensoren haben Sie die Wahl zwischen zahlreichen Geräten und Anschlussarten. Rezept 4.11 zeigt ein Beispiel für einen virtuellen Joystick, bei dem der Beschleunigungsmesser eines Wii Nunchucks genutzt wird, um Handbewegungen zu verfolgen. Rezept 13.2 enthält weitere Informationen zur Verwendung des Beschleunigungsmessers des Wii Nunchucks. Dieses Rezept verwendet analoge Ausgangswerte, die proportional zur Beschleunigung sind. Geeignete Bauelemente sind der ADXL203CE (SF SEN-00844), der ADXL320 (SF SEN 00847) und der MMA7260Q (SF SEN00252) – weitere Information finden Sie in der SparkFun Beschleunigungsmesser-Auswahlhilfe unter (*http://www.sparkfun.com/tutorials/167*) auf der SparkFun-Website.

Abbildung 6-22 zeigt den Anschluss der x- und y-Achsen an den analogen Beschleunigungsmesser.

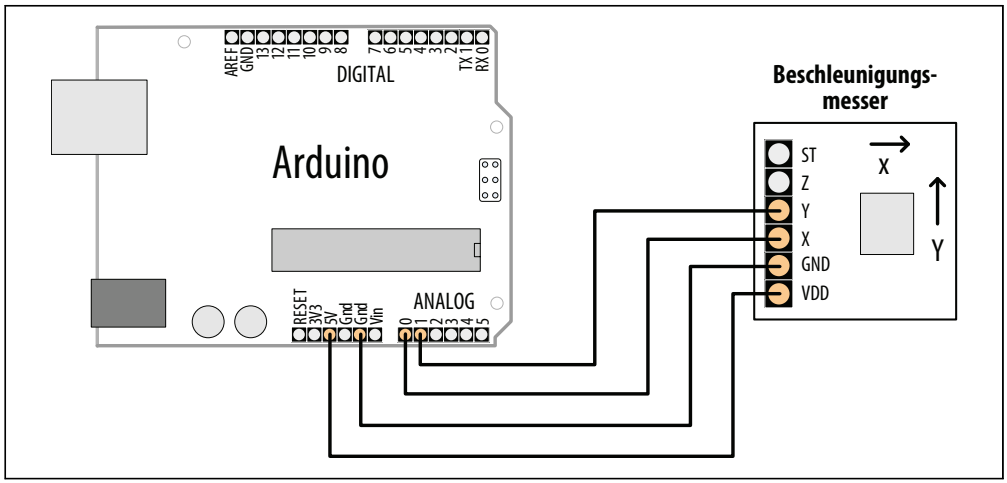
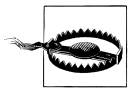


Abbildung 6-22: Anschluss der x- und y-Achsen eines analogen Beschleunigungsmessers



Stellen Sie mit Hilfe des Datenblatts sicher, dass das Bauelement die maximal erlaubte Spannung nicht überschreitet. Viele Beschleunigungsmesser sind für den 3,3-V-Betrieb ausgelegt und können beschädigt werden, wenn Sie mit der 5-V-Spannung eines Arduino-Boards verbunden werden.

Der nachfolgende einfache Sketch verwendet den ADXL320, um die Beschleunigung der x- und y-Achsen auszugeben:

```
/*
 accel Sketch
 Einfacher Sketch zur Ausgabe der Werte für die x- und y-Achsen
 */

const int xPin = 0;  // Analoge Eingangspins
const int yPin = 1;

void setup()
{
 Serial.begin(9600);  // Beachten Sie die höhere Geschwindigkeit
}

void loop()
{
int xValue;  // Werte des Beschleunigungsmessers
int yValue;

  xValue = analogRead(xPin);
  yValue = analogRead(yPin);

  Serial.print("X = ");
  Serial.println(xValue);

  Serial.print("Y = ");
  Serial.println(yValue);
  delay(100);
}
```

Diskussion

Sie können Techniken aus den vorangegangenen Rezepten verwenden, um Informationen aus den Messwerten des Beschleunigungsmessers zu extrahieren. Sie könnten auf einen Schwellwert prüfen wollen, um eine Bewegung zu erkennen (ein Beispiel für eine Schwellwerterkennung finden Sie in Rezept 6.6). Sie könnten Durchschnittswerte wie in Rezept 6.7 verwenden müssen, um nützliche Werte zu erhalten. Liefert der Beschleunigungsmesser horizontale Werte zurück, können Sie sie direkt in Bewegung umrechnen. Bei vertikalen Werten müssen Sie die Auswirkungen der Gravitation berücksichtigen. Das ähnelt dem Gleichspannungs-Offset aus Rezept 6.7, kann aber kompliziert werden, da der Beschleunigungsmesser seine Richtung ändern kann, so dass der Einfluss der Gravitation bei der Messung keine Konstante ist.

Siehe auch

SparkFun Auswahlhilfe: *http://www.sparkfun.com/commerce/tutorial_info.php?tutorials_id=167*

KAPITEL 7

Visuelle Ausgabe

7.0 Einführung

Mit der visuellen Ausgabe kann der Arduino protzen, und entsprechend viele LED-Gerätschaften werden unterstützt. Bevor wir uns den Rezepten dieses Kapitels zuwenden, wollen wir uns aber die digitalen und analogen Ausgänge des Arduino ansehen. Diese Einführung bietet einen guten Einstieg, wenn Sie mit der Verwendung digitaler und analoger Ausgänge (`digitalWrite` und `analogWrite`) nicht vertraut sind.

Digitale Ausgänge

Alle Pins, die als digitale Eingänge genutzt werden können, können auch als digitale Ausgänge verwendet werden. Kapitel 5 enthält eine Übersicht aller Anschlüsse des Arduino. Sie sollten sich die Einführung dieses Kapitels ansehen, wenn Sie nicht wissen, wie man etwas an diese Arduino-Pins anschließt.

Digitale Ausgänge sorgen dafür, dass die Spannung an einem Pin entweder an (HIGH, 5 Volt) oder aus (LOW, 0 Volt) ist. Mit der Funktion `digitalWrite(outputPin, value)` können Sie etwas ein- und ausschalten. Die Funktion verwendet zwei Parameter: `outputPin` ist der zu steuernde Pin und `value` ist entweder `HIGH` (5 Volt) oder `LOW` (0 Volt).

Damit die Spannung am Pin auf diesen Befehl reagiert, muss sich der Pin im *Ausgangs*modus befinden, der mit `pinMode(outputPin, OUTPUT)` gesetzt wird. Der Sketch in Rezept 7.1 zeigt beispielhaft, wie man einen digitalen Ausgang nutzt.

Analoge Ausgänge

Analog bezieht sich auf die Spannungspegel, die schrittweise bis zum Maximum verändert werden können (denken Sie an Helligkeits- und Lautstärkeregler). Arduino besitzt die Funktion `analogWrite`, mit deren Hilfe Sie beispielsweise die Helligkeit einer mit dem Arduino verbundenen LED steuern können.

Die `analogWrite`-Funktion arbeitet in Wahrheit gar nicht analog, auch wenn sie sich so verhält (wie Sie gleich noch sehen werden). `analogWrite` verwendet eine Technik, die man

als Pulsweitenmodulation (Pulse Width Modulation, kurz PWM) bezeichnet. Sie emuliert ein analoges Signal mit Hilfe digitaler Impulse.

PWM verändert dabei die Dauer der An/Aus-Zeiten der Impulse (siehe Abbildung 7-1). Niedrige Ausgangswerte werde dabei durch Impulse erzeugt, die nur für eine kurze Zeitspanne an sind. Bei höheren Ausgangswerten werden diese An-Perioden immer länger. Werden diese Impulse schnell genug wiederholt (beim Arduino etwa 500 mal pro Sekunde), können wir Menschen dieses Pulsieren nicht mehr erkennen, und LEDs sehen so aus, als würde sich ihre Helligkeit sanft verändern, wenn die Impulsbreite verändert wird.

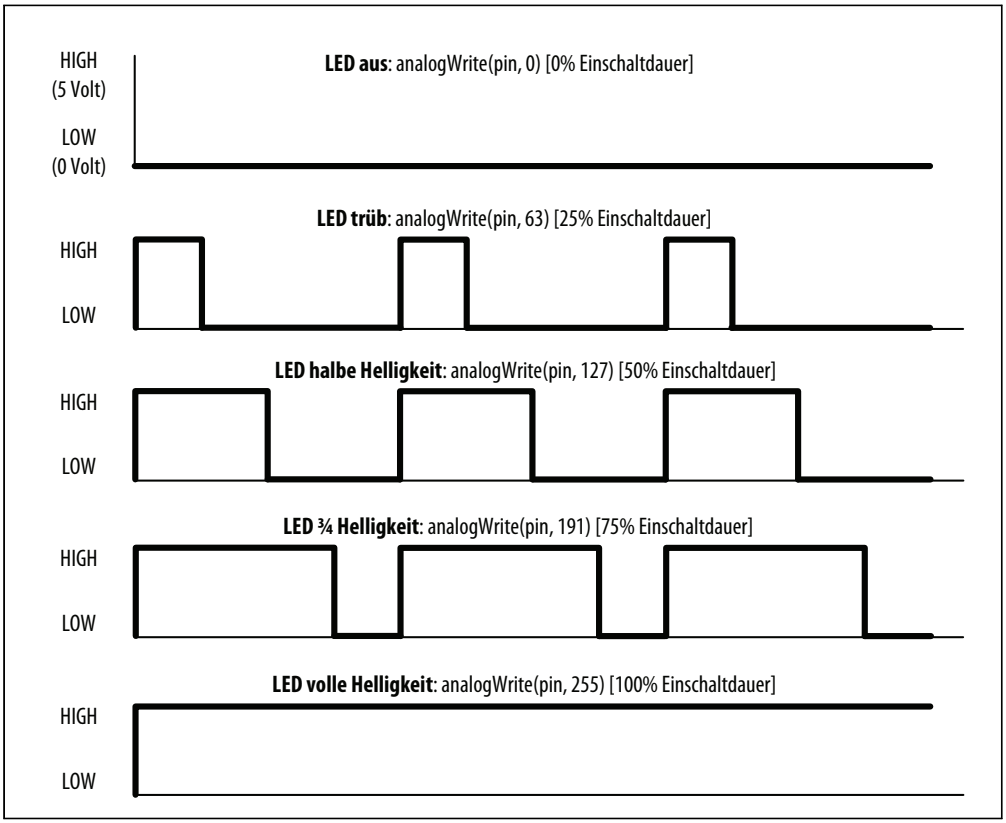


Abbildung 7-1: PWM-Ausgabe für verschiedene analogWrite-Werte

Der Arduino besitzt nur eine beschränkte Anzahl von Pins, die für die analoge Ausgabe verwendet werden können. Bei einem Standard-Board stehen die Pins 3, 5, 6, 9, 10 und 11 zur Verfügung. Bei einem Arduino Mega können Sie die Pins 2 bis 13 für analoge Ausgaben nutzen. Viele der nachfolgenden Rezepte nutzen Pins, die sowohl digital als auch analog genutzt werden können. Sie müssen dann nicht alles neu verdrahten, wenn Sie andere Rezepte ausprobieren wollen. Wenn Sie andere Pins für die analoge Ausgabe wählen, müssen Sie sich für einen der anderen Pins entscheiden, die von `analogWrite` unterstützt werden (alle anderen Pins erzeugen keinen Ausgabewert).

Licht steuern

Die Steuerung der Helligkeit über digitale oder analoge Ausgänge ist eine vielseitige, effektive und weitverbreitete Methode der Interaktion mit dem Benutzer. Einzelne LEDs, Arrays und numerische Displays werden in den Rezepten dieses Kapitels umfassend behandelt. LCD-Text- und -Grafik-Displays verlangen andere Techniken und werden in Kapitel 11 behandelt.

Technische Daten von LEDs

Eine LED ist ein Halbleiter-Bauelement (eine Diode) mit zwei Anschlüssen: einer *Anode* und einer *Kathode*. Ist die Spannung an der Anode »positiver« als an der Kathode (den Unterschied nennt man *Fluss-Spannung*), emittiert das Bauelement Licht (Photonen). Der Anschluss der Anode ist üblicherweise länger, und häufig ist die Seite des Gehäuses mit der Kathode auch abgeflacht (siehe Abbildung 7-2). Die Farbe der LED und der genaue Wert der Fluss-Spannung hängt von der Bauart der Diode ab.

Eine typische rote LED hat eine Fluss-Spannung von etwa 1,8 Volt. Ist die Spannung an der Anode nicht um 1,8 Volt »positiver« als an der Kathode, fließt kein Strom durch die LED, und es wird kein Licht erzeugt. Wird die Spannung an der Anode um 1,8 Volt positiver als an der Kathode, »schaltet« sich die LED ein (sie leitet), und es kommt quasi zu einem Kurzschluss. Sie müssen den Strom mit einem Widerstand beschränken, oder die LED brennt (früher oder später) durch. Rezept 7.1 zeigt, wie man die Werte für die strombeschränkenden Widerstände berechnet.

Möglicherweise müssen Sie ein LED-Datenblatt konsultieren, um die für Ihre Anwendung geeignete LED zu ermitteln, insbesondere um die Werte für die Fluss-Spannung und den Maximalstrom herauszufinden. 7-1 und 7-2 führen die wichtigsten Daten auf, die Sie sich in einem LED-Datenblatt ansehen sollten.

Tabelle 7-1: LED-Schlüsseldaten: absolute Grenzdaten (absolute maximum ratings)

Parameter	Symbol	Nennwert	Einheit	Kommentar
Fluss-Strom	If	25	mA	Maximaler Dauerstrom für diese LED
Spitzen-Fluss-Strom (1/10 duty @ 1 kHz)	If	160	mA	Maximaler Impulsstrom (hier für einen Impuls von 1/10 an und 9/10 aus)

Tabelle 7-2: LED-Schlüsseldaten: elektro-optische Eigenschaften

Parameter	Symbol	Nennwert	Einheit	Kommentar
Lichtstärke	Iv	2	mcd	If = 2 mA – Helligkeit bei 2 mA Strom
	Iv	40	mcd	If = 20 mA – Helligkeit bei 20 mA Strom
Abstrahlwinkel		120	Grad	Abstrahlwinkel des Lichtstrahls
Wellenlänge		620	nm	Dominante oder Spitzen-Wellenlänge (Farbe)
Fluss-Spannung	Vf	1.8	Volt	LED-Spannung, wenn an

Arduino-Pins können bis zu 40 mA Strom liefern. Das ist für eine typische LED mittlerer Helligkeit mehr als genug, reicht aber nicht aus, um LEDs mit höherer Helligkeit oder mehrere an einem Pin angeschlossene LEDs zu betreiben. Rezept 7.3 zeigt, wie man einen Transistor nutzt, um den Strom für die LED zu erhöhen.

Mehrfarbige LEDs bestehen aus zwei oder mehr LEDs in einem physikalischen Gehäuse. Sie können mehr als zwei Anschlüsse aufweisen, um die verschiedenen Farben separat steuern zu können. Da es so viele Verpackungsvarianten gibt, sollten Sie auf dem Datenblatt der LED nachsehen, wie die Anschlüsse zu verschalten sind.

Die Farben sich selbst ändernder mehrfarbiger LEDs mit einem integrierten Chip können nicht angesteuert werden. Da PWM die Spannung sehr schnell ein- und ausschaltet, starten Sie den Chip unter dem Strich immer wieder neu, was diese LEDs für PWM-Anwendungen ungeeignet macht.

Multiplexing

Anwendungen, die viele LEDs ansteuern müssen, können eine als *Multiplexing* bezeichnete Technik verwenden. Multiplexing funktioniert, indem es Gruppen von LEDs (die üblicherweise in Zeilen und Spalten angeordnet sind) nacheinander ein- und ausschaltet. Rezept 7.11 zeigt, wie vier Ziffern aus 32 einzelnen LEDs (acht LEDs pro Ziffer samt Dezimalpunkt) mit nur 12 Pins angesteuert werden können. Acht Pins steuern die Ziffern-Segmente für alle Ziffern an und vier Pins entscheiden, welche Ziffer gerade aktiv ist. Werden die Ziffern sehr schnell durchlaufen (mindestens 25-mal pro Sekunde) verschwindet der Eindruck eines pulsierenden Lichts, d.h., die Anzeige-Elemente sind scheinbar alle gleichzeitig an. Dieses Phänomen nennt man *Phi-Effekt*.

Charlieplexing arbeitet ebenfalls mit Multiplexing, nutzt zusätzlich aber noch die Tatsache aus, dass die LEDs eine *Polarität* besitzen (d.h., sie leuchten nur, wenn die Anode positiver geladen ist als die Kathode). Dabei wird durch Umschalten der Polarität zwischen zwei LEDs hin- und hergeschaltet.

Maximaler Pin-Strom

LEDs können mehr Strom verbrauchen, als der Arduino-Chip liefern kann. Das Datenblatt gibt den Maximalwert für den Arduino-Chip (ATmega328P) mit 40 mA pro Pin an. Der Chip kann insgesamt 200 mA verarbeiten und weitergeben, z.B. fünf Pins `HIGH` und fünf Pins `LOW` mit jeweils 40 mA pro Pin. Damit die Zuverlässigkeit nicht leidet, entwirft man die Anwendungen in der Praxis so, dass sie sich innerhalb der absoluten Grenzwerte bewegen. Also hält man den Strom bei 30 mA (oder weniger), um noch ausreichend Luft zu haben. Bei Hobby-Anwendungen, bei denen ein höherer Strom benötigt wird und eine eingeschränkte Zuverlässigkeit akzeptabel ist, können Sie einen Pin mit bis zu 40 mA betreiben, solange die Gesamt-Obergrenze von 200 mA nicht überschritten wird.

In der Diskussion von Rezept 7.3 finden Sie einen Tipp, wie Sie den Strom ohne externe Transistoren erhöhen können.

Das Datenblatt bezeichnet die 40 mA als absoluten Maximalwert, und einige Ingenieure werden zögern, in der Nähe dieses Wertes zu arbeiten. Allerdings wurde dieser 40 mA-Wert von Atmel noch einmal bekräftigt: Die Pins können diesen Strom ohne Weiteres verarbeiten. Die folgenden Rezepte orientieren sich an diesem 40 mA-Maximum. Wenn Sie allerdings etwas bauen, bei dem es mehr auf Zuverlässigkeit ankommt, ist es vernünftig, diesen Wert auf 30 mA zu senken, damit Sie auf der sicheren Seite sind.

7.1 LEDs anschließen und nutzen

Problem

Sie wollen ein oder mehrere LEDs ansteuern und den richtigen strombegrenzenden Widerstand auswählen, um die LEDs nicht zu beschädigen.

Lösung

Das Ein- und Ausschalten einer LED ist mit dem Arduino eine einfache Sache, und einige Rezepte in den vorangegangenen Kapiteln haben sich diese Fähigkeit auch zunutze gemacht (Rezept 5.1 zeigt beispielsweise, wie man die fest eingebaute LED an Pin 13 ansteuern kann). Dieses Rezept hilft bei der Auswahl und Verwendung externer LEDs. Abbildung 7-2 zeigt den Anschluss von drei LEDs, Sie können den Sketch aber auch mit nur einer oder zwei verwenden.

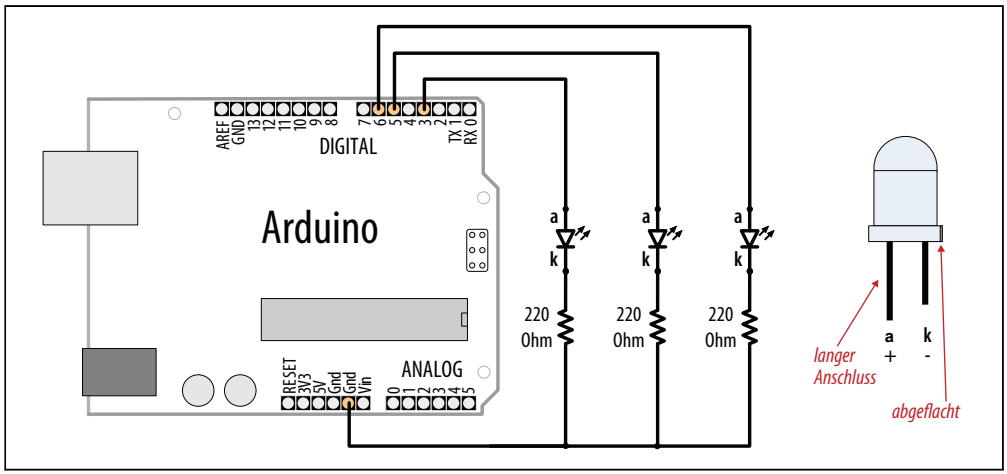


Abbildung 7-2: Anschluss externer LEDs

Das schematische Symbol für die Kathode (der negative Pin) ist *k*, nicht *c*. Das *c* steht für einen Kondensator!

Der folgende Sketch schaltet nacheinander die drei LEDs an den Pins 3, 5 und 6 für jeweils eine Sekunde ein:

```
/*
 LEDs Sketch
 Drei LEDs an unterschiedlichen Digitalpins blinken lassen
 */

const int firstLedPin  = 3;     // Pins für die jeweiligen LEDs wählen
const int secondLedPin = 5;
const int thirdLedPin  = 6;

void setup()
{
 pinMode(firstLedPin, OUTPUT);   // LED-Pin als Ausgang deklarieren
 pinMode(secondLedPin, OUTPUT);   // LED-Pin als Ausgang deklarieren
 pinMode(thirdLedPin, OUTPUT);   // LED-Pin als Ausgang deklarieren
}

void loop()
{
 // Jede LED für 1000 Millisekunden (1 Sekunde) blinken lassen
 blinkLED(firstLedPin, 1000);
 blinkLED(secondLedPin, 1000);
 blinkLED(thirdLedPin, 1000);
}


// LED am angegebenen Pin für die angegebene Dauer in Millisekunden blinken lassen
void blinkLED(int pin, int duration)
{
 digitalWrite(pin, HIGH);   // LED einschalten
 delay(duration);
 digitalWrite(pin, LOW);    // LED ausschalten
 delay(duration);
}
```

Der Sketch legt die mit den LEDs verbundenen Pins in der `setup`-Funktion als Ausgänge fest. Die `loop`-Funktion ruft `blinkLED` auf, um die LED an jedem der drei Pins blinken zu lassen. `blinkLED` schaltet den angegebenen Pin für eine Sekunde (1000 Millisekunden) auf `HIGH`.

Diskussion

Da die Anoden mit den Arduino-Pins und die Kathoden mit Masse verbunden sind, leuchten die LEDs, wenn der Pin auf `HIGH` gesetzt wird, und gehen wieder aus, wenn er `LOW` ist. Sie können die LED auch leuchten lassen, wenn der Pin `LOW` ist und die Kathoden mit den Pins und die Anoden mit Masse verbunden sind (die Widerstände können an beliebiger Stelle dazwischengeschaltet werden).

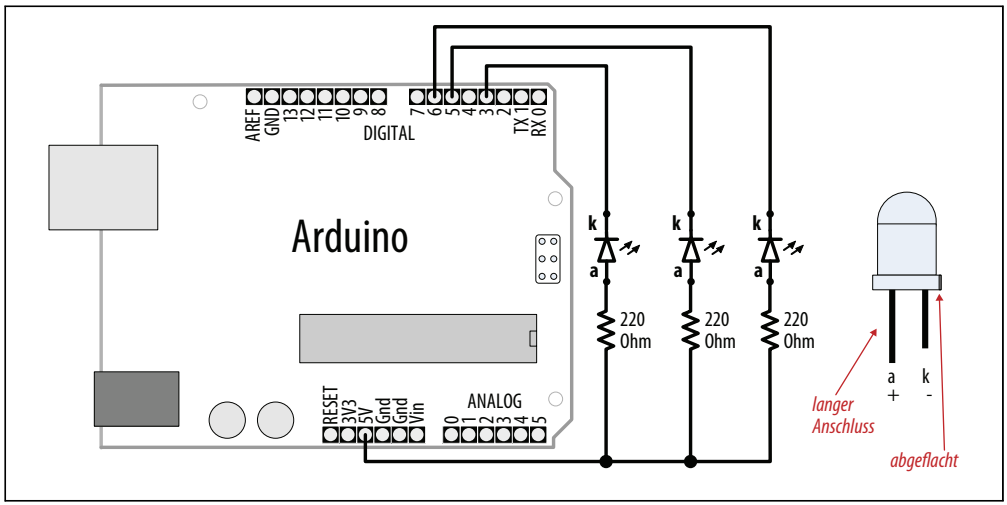


Abbildung 7-3: Anschluss externer LEDs mit Kathode am Pin

Wird eine LED mit der Anode mit +5V verbunden (wie in Abbildung 7-3 zu sehen), leuchtet sie, wenn der Pin auf `LOW` gesetzt wird (der visuelle Effekt kehrt sich um – eine LED geht für eine Sekunde aus, während die beiden anderen an bleiben).

LEDs benötigen einen Vorwiderstand, um den Strom zu kontrollieren, sonst können sie sehr schnell durchbrennen. Für die eingebaute LED an Pin 13 gibt es einen solchen Vorwiderstand auf der Platine. Bei externen LEDs muss ein Vorwiderstand mit der Anode oder Kathode verbunden werden.

Ein bei einer LED in Reihe geschalteter Vorwiderstand kontrolliert den Strom, der fließt, wenn die LED schaltet. Zur Berechnung des Widerstandwerts müssen Sie die Eingangspannung kennen (Vs, üblicherweise 5 Volt), die Fluss-Spannung der LED (Vf) und den Strom (I), der durch die LED fließen soll.

Die Formel für den Widerstand in Ohm (bekannt als Ohmsches Gesetz) lautet

R = (Vs – Vf) / I

Wenn Sie zum Beispiel eine LED bei einer Versorgungsspannung von 5 Volt mit einer Fluss-Spannung von 1,8 Volt und einem Strom von 15 mA ansteuern wollen, setzen Sie die folgenden Werte ein:

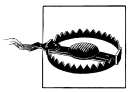
Vs = 5 (für ein 5V-Arduino-Board)
Vf = 1,8 (die Fluss-Spannung der LED)
I = 0,015 (1 Milliampere [mA] ist ein tausendstel Ampere d.h., 15 mA sind 0,015 A)

Die Spannung bei eingeschalteter LED ist (Vs – Vf) also 5 – 1,8, also 3,2 Volt.

Die Berechnung des Vorwiderstands ergibt also 3,2 / 0,015 oder 213 Ohm.

Der Wert von 213 Ohm entspricht keinem Standard-Widerstandswert, weshalb wir ihn auf 220 Ohm aufrunden.

Der Widerstand in Abbildung 7-2 liegt zwischen Kathode und Masse, kann aber ebenso gut auf der anderen Seite der LED (zwischen +5V und Anode) angeschlossen werden.



Arduino-Pins können maximal 40 mA Strom liefern. Wenn Ihre LED mehr Strom benötigt, sehen Sie sich Rezept 7.3 an.

Siehe auch

Rezept 7.3

7.2 Helligkeit einer LED regeln

Problem

Sie wollen die Helligkeit einer oder mehrerer LEDs aus Ihrem Sketch heraus regeln.

Lösung

Verbinden Sie jede LED mit einem PWM-fähigen Analogausgang. Verwenden Sie die Verschaltung aus Abbildung 7-2. Der Sketch lässt die LED langsam (über einen Zeitraum von etwa 5 Sekunden) immer heller werden und dann wieder dunkel.

```
/*
 * LedBrightness Sketch
 * Steuert die Helligkeit von LEDs an analogen Ausgangsports
*/

const int firstLed  = 3;     // Pins für die LEDs festlegen
const int secondLed = 5;
const int thirdLed  = 6;

int brightness = 0;
int increment = 1;

void setup()
{
 // Mit analogWrite angesteuerte Pins müssen nicht als Ausgänge deklariert werden
}

void loop()
{
 if(brightness > 255)
 {
   increment = -1; // Beim Erreichen von 255 herunterzählen
 }
 else if(brightness < 1)
 {
```

```
  increment = 1; // Beim Erreichen von 0 wieder hochzählen
 }
 brightness = brightness + increment; // Inkrementieren (oder bei negativem Vorzeichen
                                      // dekrementieren)

 // Helligkeitswert an die LEDs schreiben
 analogWrite(firstLed, brightness);
 analogWrite(secondLed, brightness);
 analogWrite(thirdLed, brightness );

 delay(10); // 10ms pro Schritt bedeutet 2,55 Sekunden für auf- und abblenden
}
```

Diskussion

Wir verwenden die gleiche Verschaltung wie im vorigen Sketch, steuern die Pins aber über `analogWrite` anstelle von `digitalWrite` an. `analogWrite` nutzt PWM, um die Spannung an der LED zu kontrollieren. In der Einführung zu diesem Kapitel erfahren Sie mehr über analoge Ausgänge.

Der Sketch regelt die Helligkeit, indem er den Wert der `brightness`-Variable bei jedem Schleifendurchlauf erhöht (heller) oder verringert (dunkler). Dieser Wert wird dann über die `analogWrite`-Funktion an die drei angeschlossenen LEDs übergeben. Der Minimalwert für `analogWrite` ist `0` – das entspricht einer Spannung von 0 Volt am Pin. Der Maximalwert ist 255 und hält die Spannung am Pin bei 5 Volt.

Erreicht die `brightness`-Variable ihren Maximalwert, wird sie wieder kleiner, weil sich das Vorzeichen von `increment` von +1 auf –1 ändert (–1 zu einem Wert zu addieren ist das gleiche, wie eine 1 zu subtrahieren).

Siehe auch

Die Einführung zu diesem Kapitel erläutert, wie die Analogausgänge des Arduino funktionieren.

7.3 Hochleistungs-LEDs ansteuern

Problem

Sie müssen LEDs schalten oder ansteuern, die mehr Strom benötigen, als die Arduino-Pins zur Verfügung stellen. Arduino-Chips können nur einen Strom von bis zu 40 mA pro Pin liefern.

Lösung

Verwenden Sie einen Transistor, um den Stromfluss durch die LEDs zu steuern. Schließen Sie die LED wie in Abbildung 7-4 zu sehen an. Sie können den gleichen Code wie in den

obigen Rezepten nutzen, müssen aber sicherstellen, dass die Basis des Transistors mit dem im Sketch verwendeten Pin verbunden ist.

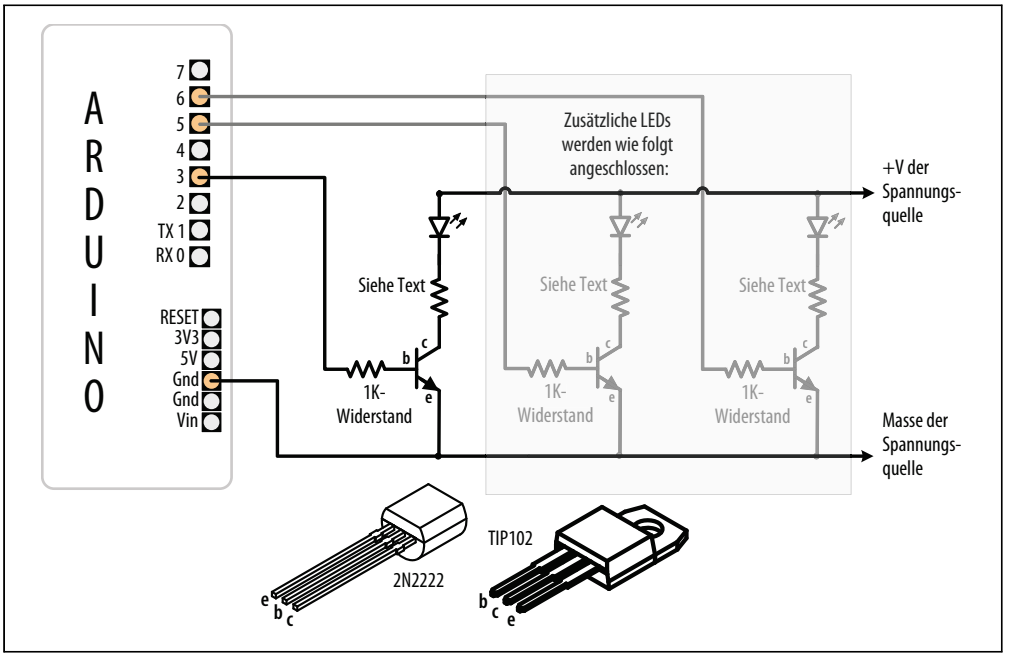


Abbildung 7-4: Transistoren zur Steuerung von Hochleistungs-LEDs nutzen

Diskussion

In Abbildung 7-4 verweist ein Pfeil auf den Pluspol (+V) der Stromversorgung. Das kann der +5V-Pin des Arduino sein, der bis zu 400 mA liefern kann, wenn er über USB versorgt wird. Die bei einer externen Stromversorgung zur Verfügung stehenden Strom- und Spannungswerte hängen vom verwendeten Netzteil ab (der Laderegler leitet überschüssige Spannung als Wärme ab – stellen Sie sicher, dass der Laderegler, ein 3-Pin-Chip in der Nähe des Eingangssteckers, nicht zu heiß wird). Wenn Sie mehr Strom benötigen, als der +5V-Pin des Arduino liefern kann, müssen Sie eine vom Arduino unabhängige Stromquelle verwenden, die die LEDs antreiben kann. Informationen zur Verwendung externer Stromquellen finden Sie in Anhang C.

Wenn Sie eine externe Stromversorgung verwenden, müssen Sie darauf achten, ihre Masse mit der Arduino-Masse zu verbinden.

Wird der Transistor eingeschaltet, fließt Strom vom Kollektor zum Emitter. Ist der Transistor ausgeschaltet, fließt kein signifikanter Strom. Der Arduino kann einen Transistor einschalten, indem er die Spannung an einem Pin mittels `digitalWrite` auf `HIGH` setzt. Ein Widerstand zwischen Pin und Transistor ist nötig, damit nicht zu viel Strom

fließt – 1K-Ohm ist ein typischer Wert (der 5 mA an die Basis des Transistors liefert). Anhang B zeigt, wie man Datenblätter liest und einen Transistor auswählt und einsetzt. Sie können auch spezialisierte ICs wie den ULN2003A verwenden, um mehrere Ausgänge anzutreiben. Er verfügt über sieben hochstromige (0,5 amp) Ausgangstreiber.

Der Widerstand zur Begrenzung des Stromflusses durch die LED wird mit der Technik berechnet, die in Rezept 7.1 beschrieben wurde. Eventuell müssen Sie aber berücksichtigen, dass die Quellspannung durch den kleinen Spannungsverlust am Transistor ein wenig sinkt. Er liegt üblicherweise unter drei Viertel eines Volts (den tatsächlichen Wert können Sie unter der Kollektor/Emitter-Sättigungsspannung nachsehen; siehe Anhang B). Hochleistungs-LEDs (1 Watt oder mehr) werden am besten über eine konstante Stromquelle (einer Schaltung, die den Strom aktiv kontrolliert) angetrieben, um den Stromfluss durch die LED zu steuern.

Wie man die 40 mA pro Pin umgeht

Sie können auch mehrere Pins parallel schalten, um die Grenze von 40 mA pro Pin zu umgehen (siehe Rezept 7.1).

Abbildung 7-5 zeigt, wie man eine LED mit 60 mA über zwei Pins ansteuern kann. Wie Sie sehen, verbindet die LED die Widerstände an den Pins 2 und 7 mit Masse. Beide Pins müssen `LOW` sein, damit die vollen 60 mA durch die LED fließen können. Die separaten Widerstände werden benötigt, versuchen Sie nicht, nur einen Widerstand mit den beiden Pins zu verbinden.

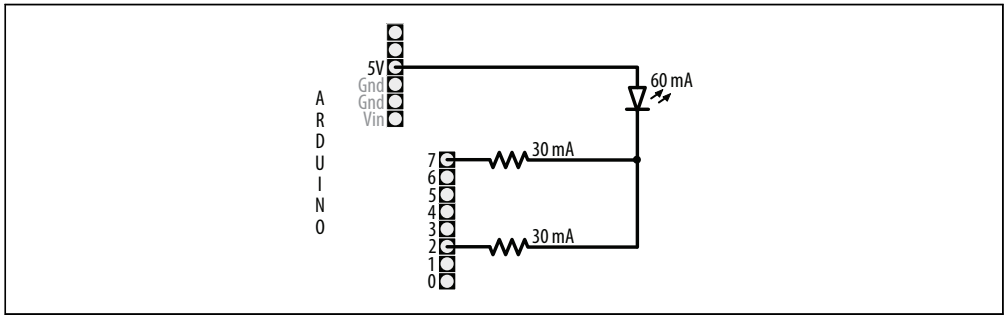


Abbildung 7-5: 40 mA pro Pin umgehen

Diese Technik kann auch genutzt werden, um Strom zu liefern. Wenn Sie die LED beispielsweise umdrehen – also den Anschluss auf Widerstandsseite (Kathode) mit Masse verbinden und den anderen (Anode) mit den Widerständen –, dann schalten Sie die LED ein, indem Sie beide Pins auf `HIGH` setzen.

Es ist am besten, wenn Sie keine benachbarten Pins verwenden, um die Belastung für den Chip zu minimieren. Diese Technik funktioniert für jeden Pin, der `digitalWrite` nutzen kann, funktioniert aber nicht mit `analogWrite`. Wenn Sie für Analogausgänge (PWM) mehr Strom benötigen, müssen Sie wie oben beschrieben mit Transistoren arbeiten.

Siehe auch

Web-Referenz für Konstantstrom-Treiber *http://blog.makezine.com/archive/2009/08/constant_current_led_driver.html*

7.4 Die Farbe einer LED steuern

Problem

Sie wollen die Farbe einer RGB-LED aus einem Programm heraus steuern.

Lösung

Bei RGB-LEDs sind rote, grüne und blaue Elemente in einem einzelnen Gehäuse untergebracht. Dabei sind entweder die Anoden miteinander verbunden (die sog. *gemeinsame Anode*) oder die Kathoden (*gemeinsame Kathode*). Verwenden Sie die Schaltung aus Abbildung 7-6 bei gemeinsamer Anode (die Anoden sind mit +5 V verbunden und die Kathoden mit den Pins.) Nutzen Sie Abbildung 7-2, wenn Ihre RGB-LEDs mit gemeinsamer Kathode arbeiten.

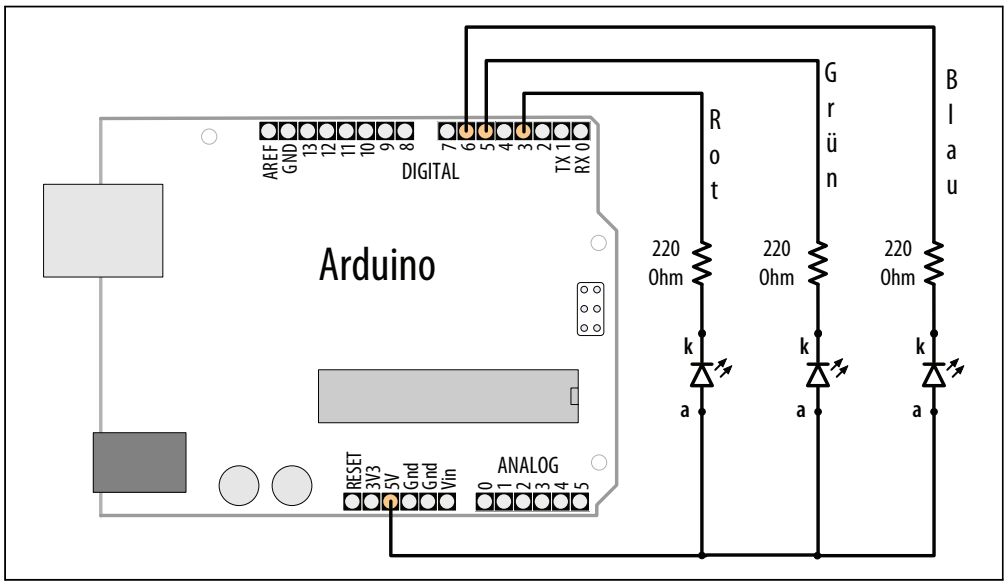


Abbildung 7-6: Anschluss einer RGB-LED (gemeinsame Anode)

Der Sketch bewegt sich kontinuierlich durch das Farbspektrum der LED, indem er die Intensität der roten, grünen und blauen Elemente verändert:

```
/*
 * RGB_LEDs Sketch
 * RGB-LEDs über analoge Ausgangsports ansteuern
```

```
*/

const int redPin  = 3;     // Pins für die LEDs
const int greenPin = 5;
const int bluePin  = 6;
const boolean invert = true; // true bei gemeinsamer Anode, false bei gemeinsamer Kathode

int color = 0; // Ein Wert zwischen 0 und 255 repräsentiert den Farbton
int R, G, B;  // Die Farbkomponenten für Rot, Grün und Blau

void setup()
{
 // Mit analogWrite angesteuerte Pins müssen nicht als Ausgänge deklariert werden
}

void loop()
{
 int brightness = 255; // 255 ist maximale Helligkeit
 hueToRGB( color, brightness);  // Funktion zur Umwandlung von Farbton in RGB
 // RGB-Werte an die Pins schreiben
 analogWrite(redPin, R);
 analogWrite(greenPin, G);
 analogWrite(bluePin, B );

 color++;       // Farbe erhöhen
 if(color > 255)   //
   color = 0;
    delay(10);
}

// Funktion zur Umwandlung einer Farbe in ihre Rot-, Grün- und Blau-Komponenten.

void hueToRGB( int hue, int brightness)
{
  unsigned int scaledHue = (hue * 6);
  // Segment 0 bis 5 um den Farbkreis
  unsigned int segment = scaledHue / 256;
  // Position innerhalb des Segments
  unsigned int segmentOffset = scaledHue - (segment * 256);

  unsigned int complement = 0;
  unsigned int prev = (brightness * ( 255 -  segmentOffset)) / 256;
  unsigned int next = (brightness *  segmentOffset) / 256;
  if(invert)
  {
   brightness = 255-brightness;
   complement = 255;
   prev = 255-prev;
   next = 255-next;
  }

  switch(segment ) {
  case 0:    // Rot
   R = brightness;
     G = next;
   B = complement;
  break;
```

```
    case 1:   // Gelb
    R = prev;
    G = brightness;
    B = complement;
    break;
    case 2:   // Grün
    R = complement;
    G = brightness;
    B = next;
    break;
    case 3:   // Türkis (Cyan)
    R = complement;
    G = prev;
    B = brightness;
    break;
    case 4:   // Blau
    R = next;
    G = complement;
    B = brightness;
    break;
   case 5:    // Violett (Magenta)
    default:
    R = brightness;
    G = complement;
    B = prev;
    break;
    }
}
```

Diskussion

Die Farbe einer RGB-LED wird durch die relative Helligkeit ihrer Rot-, Grün- und Blau-Komponenten bestimmt. Die Kernfunktion des Sketches (`hueToRGB`) übernimmt die Umwandlung eines Farbtons zwischen 0 und 255 in die entsprechende Farbe zwischen Rot und Blau. Das Spektrum sichtbarer Farben wird häufig in einem Farbkreis dargestellt, der aus den Primär- und Sekundärfarben sowie allen Zwischentönen besteht. Die sechs Segmente für die Primär- und Sekundärfarben werden von sechs `case`-Anweisungen verarbeitet. Der Code in einer `case`-Anweisung wird ausgeführt, wenn die segment-Variable der case-Nummer entspricht. Die RGB-Werte werden dann auf die jeweils passenden Werte gesetzt. Segment 0 ist Rot, Segment 1 ist Gelb, Segment 2 ist Grün und so weiter.

Wenn Sie die Helligkeit anpassen wollen, können Sie den Wert der `brightness`-Variable ändern. Das nachfolgende Beispiel zeigt, wie man die Helligkeit mit einem variablen Widerstand oder Sensor korrigiert, der wie in Abbildung 7-13 oder Abbildung 7-17 angeschlossen ist:

```
int brightness = map( analogRead(0),0,1023, 0, 255); // Helligkeit über Sensor bestimmen
```

Die Variable `brightness` bewegt sich im Wertebereich von 0 bis 255. Da der analoge Eingangsbereich zwischen 0 und 1023 liegt, erhöht sich die Helligkeit der LED, wenn dieser Wert steigt.

Siehe auch

Rezept 2.16; Rezept 13.1

7.5 Mehrere LEDs aneinanderreihen: LED-Balkenanzeige

Problem

Sie wünschen sich eine LED-Balkenanzeige, die LEDs proportional zu einem Wert in Ihrem Sketch (oder von einem Sensor) ansteuert.

Lösung

Sie können die LEDs so anschließen wie in Abbildung 7-2. (Verwenden Sie weitere Pins, wenn Sie zusätzliche LEDs anschließen wollen.) Abbildung 7-7 zeigt sechs LEDs, die an benachbarten Pins angeschlossen sind.

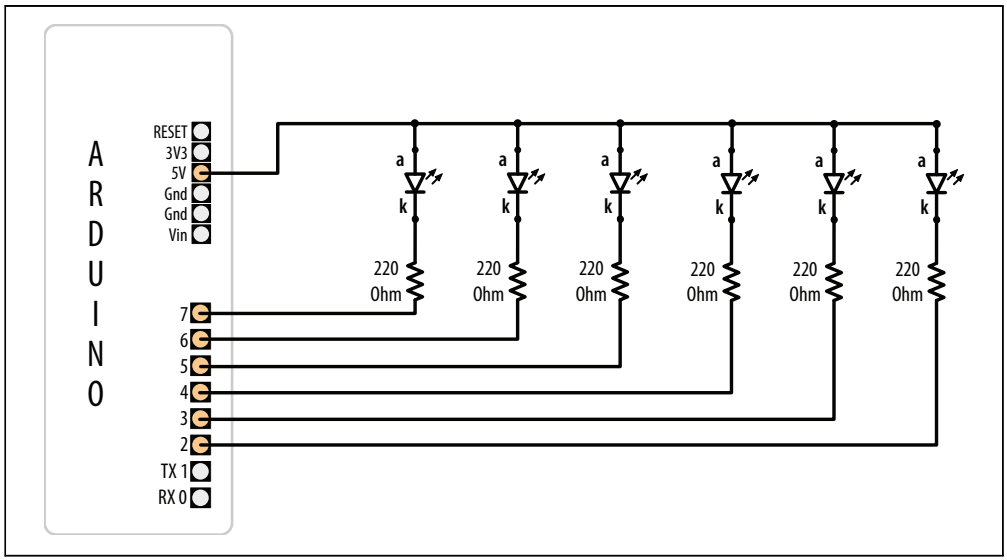


Abbildung 7-7: Sechs LEDs, die mit den Kathoden an den Arduino-Pins angeschlossen sind

Der folgende Sketch schaltet eine Reihe von LEDs ein. Die Anzahl eingeschalteter LEDs ist proportional zum Wert eines Sensors, der mit einem analogen Eingangsport verbunden ist. (Abbildung 7-13 und Abbildung 7-17 zeigen, wie man einen Sensor anschließt.)

```
/*
 Bargraph Sketch

Schaltet eine Reihe von LEDs proportional zum Wert eines analogen Sensors ein.
Sechs LEDs werden angesteuert, aber die Zahl der LEDs kann geändert werden,
indem man den Wert von NbrLEDs anpasst und die Pins in das ledPins-Array einträgt
*/
```

```
const int NbrLEDs = 6;
const int ledPins[] = { 2, 3, 4, 5, 6, 7};
const int analogInPin = 0;  // Analoger Eingangspin ist mit variablem Widerstand verbunden
const int wait = 30;

// Vertauschen Sie die beiden folgenden Konstanten, wenn die Kathoden mit Masse verbunden sind
const boolean LED_ON = LOW;
const boolean LED_OFF = HIGH;

int sensorValue = 0;      // Vom Sensor eingelesener Wert
int ledLevel = 0;       // In 'LED-Balken' umgewandelter Wert

void setup() {
 for (int led = 0; led < NbrLEDs; led++)
 {
  pinMode(ledPins[led], OUTPUT);  // Alle LED-Pins sind Ausgänge
 }
}

void loop() {
 sensorValue = analogRead(analogInPin);        // Sensorwert einlesen
 ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs);  // und auf LEDs abbilden
 for (int led = 0; led < NbrLEDs; led++)
 {
  if (led < ledLevel ) {
   digitalWrite(ledPins[led], LED_ON);    // Pins unter Wert einschalten
  }
  else {
   digitalWrite(ledPins[led], LED_OFF); // Pins über Wert ausschalten
  }
 }
}
```

Diskussion

Die mit den LEDs verbundenen Pins werden im Array `ledPins` vorgehalten. Um die Zahl der LEDs zu ändern, können Sie Elemente in dieses Array einfügen oder entfernen. Dabei müssen Sie aber sicherstellen, dass die Variable `NbrLEDs` der Anzahl der Elemente (also der Anzahl der verwendeten Pins) entspricht. Sie können den Compiler den Wert für `NbrLEDs` berechnen lassen, indem Sie die Zeile:

```
const int NbrLEDs = 6;
```

durch die folgende Zeile ersetzen:

```
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]).
```

Die Funktion `sizeof` gibt die Größe einer Variablen (in Bytes) zurück – in diesem Fall die Zahl der Bytes im `ledPins`-Array. Da es sich um ein Array von Integerwerten handelt (mit zwei Bytes pro Element), wird die Gesamtgröße des Arrays in Bytes durch die Größe eines Elements (`sizeof(ledPins[0])`) dividiert, um die Zahl der Elemente zu berechnen.

Die Arduino-Funktion `map` berechnet die Anzahl der LEDs, die proportional zum Sensorwert eingeschaltet werden sollen. Der Code geht alle LEDs durch und schaltet sie ein,

solange der proportionale Sensorwert größer ist als die LED-Nummer. Ist der Sensorwert beispielsweise 0, wird keine LED eingeschaltet. Liegt der Sensorwert in der Mitte, wird die Hälfte der LEDs eingeschaltet, und wenn der Sensor den Maximalwert zurückgibt, werden alle LEDs eingeschaltet.

Abbildung 7-7 zeigt, dass alle Anoden miteinander verbunden sind (die sog. *gemeinsame Anode*) und die Kathoden mit den jeweiligen Pins. Die Pins müssen auf LOW gesetzt werden, damit die LED leuchtet. Sind die LEDs mit den Anoden an die Pins angeschlossen (wie in Abbildung 7-2 zu sehen), während die Kathoden miteinander verbunden sind (*gemeinsame Kathode*), dann leuchtet die LED, wenn der Pin auf HIGH gesetzt wird. Der Sketch in diesem Rezept nutzt die Konstanten LED_ON und LED_OFF, um einfach zwischen gemeinsamer Anode und gemeinsamer Kathode wechseln zu können. Bei gemeinsamer Kathode vertauschen Sie die Werte der Konstanten wie folgt:

```
const boolean LED_ON = HIGH;  // HIGH schaltet die LED bei gemeinsamer Kathode ein
const boolean LED_OFF = LOW;
```

Sie können auch das *Abschwellen* der LEDs verlangsamen, z.B. um die Bewegung der Anzeige eines Lautstärkereglers zu emulieren. Hier eine Variante des Sketches, die den LED-Balken langsam »abschwellen« lässt, wenn der Pegel sinkt:

```
/*
  Abschwellende LED-Balkenanzeige
*/

const int ledPins[ ] = { 2, 3, 4, 5, 6, 7};
const int NbrLEDs = sizeof(ledPins) / sizeof(ledPins[0]);
const int analogInPin = 0; // Analoger Eingangspin ist mit variablem Widerstand verbunden
const int decay = 10;    // Erhöhung dieses Wertes verringert die "Abschwellgeschwindigkeit" für
storedValue

int sensorValue = 0;     // Vom Sensor eingelesener Wert
int storedValue = 0;     // Gespeicherter (abschwellender) Sensorwert
int ledLevel = 0;       // In 'LED-Balken' umgewandelter Wert

void setup() {
 for (int led = 0; led < NbrLEDs; led++)
 {
  pinMode(ledPins[led], OUTPUT);  // Alle LED-Pins sind Ausgänge
 }
}


void loop() {
 sensorValue = analogRead(analogInPin);     // Sensorwert einlesen
 storedValue = max(sensorValue, storedValue);  // Sensorwert nutzen, wenn größer
 ledLevel = map(storedValue, 0, 1023, 0, NbrLEDs); // Auf Anzahl LEDs abbilden
 for (int led = 0; led < NbrLEDs; led++)
 {
  if (led < ledLevel ) {
   digitalWrite(ledPins[led], HIGH);  // Pins unter Wert einschalten
  }
  else {
   digitalWrite(ledPins[led], LOW);  // Pins über Wert ausschalten
```

```
  }
 }
 storedValue = storedValue - decay;   // Wert "abschwellen" lassen
 delay(10);                 // 10 ms warten
}
```

Das Abschwellen wird in der Zeile verarbeitet, die die `max`-Funktion nutzt. Sie gibt je nachdem, welcher größer ist, den Sensorwert oder den gespeicherten Wert zurück. Ist der Sensorwert höher als der abschwellende Wert, wird er in `storedValue` gespeichert. Anderenfalls wird der Wert von `storedValue` bei jedem Schleifendurchlauf um die Konstante `decay` verkleinert. (Die Schleife selbst wird mit Hilfe der `delay`-Funktion alle 10 Millisekunden durchlaufen.) Erhöht man den Wert von `decay`, reduziert sich die Zeitspanne, in der alle LEDs ausgehen.

Siehe auch

Rezept 3.6 erklärt die `max`-Funktion.

Rezept 5.6 erläutert ausführlicher, wie man einen Sensorwert mit `analogRead` einliest.

Rezept 5.7 beschreibt die `map`- Funktion.

Wenn Sie eine größere Genauigkeit für die »Abschwellzeiten« benötigen, sehen Sie sich 12.1 und 12.2 an. Die Gesamtzeit für einen Schleifendurchgang liegt tatsächlich über 10 Millisekunden, da es eine weitere Millisekunde (oder so) dauert, bis der Rest der Schleife ausgeführt wurde.

7.6 Mehrere LEDs aneinanderreihen: Knight Rider-Lauflicht

Problem

Sie wollen LEDs in einem Lauflicht (wie in der Fernsehserie *Knight Rider*) aufleuchten lassen.

Lösung

Sie können die gleichen Anschlüsse wie in Abbildung 7-7 verwenden:

```
/* KnightRider
 */

const int NbrLEDs = 6;
const int ledPins[] = {2, 3, 4, 5, 6, 7};
const int wait = 30;

void setup(){
 for (int led = 0; led < NbrLEDs; led++)
 {
  pinMode(ledPins[led], OUTPUT);
 }
}
```

```
void loop() {
 for (int led = 0; led < NbrLEDs-1; led++)
 {
  digitalWrite(ledPins[led], HIGH);
 delay(wait);
 digitalWrite(ledPins[led + 1], HIGH);
 delay(wait);
 digitalWrite(ledPins[led], LOW);
 delay(wait*2);
 }
 for (int led = NbrLEDs-1; led > 0; led--) {
 digitalWrite(ledPins[led], HIGH);
 delay(wait);
 digitalWrite(ledPins[led - 1], HIGH);
 delay(wait);
 digitalWrite(ledPins[led], LOW);
 delay(wait*2);
 }
}
```

Diskussion

Dieser Code ähnelt dem aus Rezept 7.5, nur dass die Pins hier nicht von einem Sensorwert abhängen, sondern in einer festen Reihenfolge ein- und ausgeschaltet werden. Es gibt zwei `for`-Schleifen. Die erste erzeugt das Links-nach-rechts-Muster, indem sie die LEDs von links nach rechts einschaltet. Die Schleife beginnt mit der ersten (linken) LED und geht dann alle nachfolgenden LEDs durch, bis auch die letzte (rechte) LED leuchtet. Die zweite `for`-Schleife schaltet die LEDs von rechts nach links ein. Sie beginnt bei der rechten LED und dekrementiert die LED, bis die erste (linke) LED erreicht ist. Die Zeitverzögerng kann in der `wait`-Variablen festgelegt werden. Wählen Sie den Wert, der für den besten visuellen Effekt sorgt.

7.7 Eine LED-Matrix per Multiplexing steuern

Problem

Sie besitzen eine LED-Matrix und wollen die Anzahl der Arduino-Pins minimieren, die zu ihrer Ansteuerung benötigt werden.

Lösung

Dieser Sketch verwendet eine LED-Matrix mit 64 LEDs, bei der die Anoden mit den Zeilen und die Kathoden mit den Spalten verbunden sind (wie bei der Jameco 2132349). Zweifarbige LED-Displays sind möglicherweise leichter zu beschaffen, und Sie können einfach nur eine der Farben ansteuern, wenn Sie die andere nicht brauchen (Abbildung 7-8 zeigt den Anschluss):

```
/*
 matrixMpx Sketch
```

```
 Schaltet die LEDs, von der ersten Zeile und Spalte ausgehend, nacheinander ein, bis alle LEDs
 leuchten
 Das Multiplexing wird verwendet, um 64 LEDs mit 16 Pins anzusteuern
 */

const int columnPins[] = { 2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]  = { 10,11,12,15,16,17,18,19};

int pixel     = 0;      // 0 bis 63 LEDs in der Matrix
int columnLevel = 0;      // In LED-Spalte umgewandelter Pixelwert
int rowLevel   = 0;      // In LED-Zeile umgewandelter Pixelwert

void setup() {
 for (int i = 0; i < 8; i++)
 {
  pinMode(columnPins[i], OUTPUT);  // Alle LED-Pins sind Ausgänge
  pinMode(rowPins[i], OUTPUT);
 }
}

void loop() {
 pixel = pixel + 1;
 if(pixel > 63)
   pixel = 0;

 columnLevel = pixel / 8;             // Auf Anzahl Spalten abbilden
 rowLevel = pixel % 8;               // Rest bestimmen
 for (int column = 0; column < 8; column++)
 {
  digitalWrite(columnPins[column], LOW);    // Diese Spalte an Masse
  for(int row = 0; row < 8; row++)
  {
   if (columnLevel > column)
   {
    digitalWrite(rowPins[row], HIGH);  // Alle LEDs in der Zeile an +5 Volt
   }
   else if (columnLevel == column && rowLevel >= row)
   {
     digitalWrite(rowPins[row], HIGH);
   }
   else
   {
    digitalWrite(columnPins[column], LOW); // Alle LEDs in der Zeile ausschalten
   }
   delayMicroseconds(300);    // Verzögerung liefert eine Framedauer von 20ms für 64 LEDs
   digitalWrite(rowPins[row], LOW);      // LED ausschalten
  }

  // Die Spalte von Masse trennen
  digitalWrite(columnPins[column], HIGH);
 }
}
```

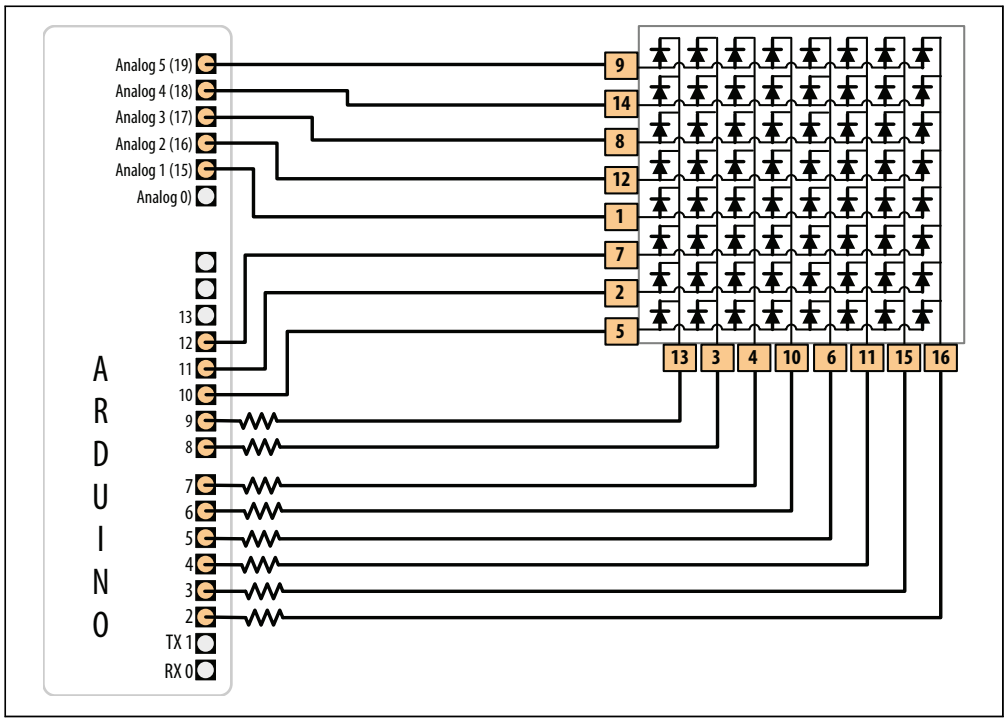
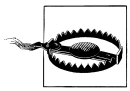


Abbildung 7-8: LED-Matrix an 16 Digitalpins



LED-Matrixanzeigen haben keine Standard-Anschlussbelegung, d.h., Sie müssen sie auf dem Datenblatt der Anzeige nachsehen. Verbinden Sie die Zeilen mit den Anoden und die Spalten mit den Kathoden, wie in Abbildung 7-15 oder Abbildung 7-16 dargestellt, verwenden Sie aber die LED-Pin-Nummern aus Ihrem Datenblatt.

Diskussion

Die Widerstandswerte müssen so gewählt werden, dass der Maximalstrom von 40 mA pro Pin nicht überschritten wird. Da der Strom für bis zu acht LEDs durch jeden Spaltenpin fließen kann, darf der maximale Strom für jede LED nur ein Achtel von 40 mA, also 5 mA, betragen. Jede LED in einer typischen kleinen roten Matrix hat eine Fluss-Spannung von etwa 1,8 Volt. Die Berechnung des Widerstands, der zu einem Strom von 5mA bei einer Fluss-Spannung von 1,8V führt, ergibt einen Wert von 680 Ohm. Suchen Sie sich aus dem Datenblatt die Fluss-Spannung für Ihre Matrix heraus. Jede Spalte der Matrix ist durch einen Vorwiderstand mit einem Digitalpin verbunden. Geht die Spannung am Spaltenpin herunter und die am Zeilenpin hoch, leuchtet die dazugehörige LED. Bei allen LEDs, bei denen die Spannung am Spaltenpin hoch oder am Zeilenpin unten ist, fließt kein Strom durch die LED, und sie leuchtet nicht.

The for-Schleife geht alle Zeilen und Spalten durch und schaltet nacheinander die LEDs ein, bis alle leuchten. Die Schleife beginnt in der ersten Spalte und Zeile und inkrementiert die Zeilenzähler, bis alle LEDs in dieser Zeile leuchten. Sie macht dann mit der nächsten Spalte weiter, und so weiter, bis nacheinander alle LEDs an sind.

Sie können die LEDs auch proportional zu einem Sensorwert leuchten lassen (Rezept 5.6 zeigt, wie man einen Sensor an einen Analogport anschließt), indem Sie die folgenden Änderungen am Sketch vornehmen.

Kommentieren Sie die drei folgenden Zeilen am Anfang der Schleife aus:

```
pixel = pixel + 1;
if(pixel > 63)
  pixel = 0;
```

Ersetzen Sie das durch die folgenden Zeilen, die den Sensorwert an Pin 0 einlesen und auf die Zahl der Pixel zwischen 0 und 63 abbilden:

```
int sensorValue = analogRead(0);         // Sensorwert einlesen
pixel =  map(sensorValue, 0, 1023, 0, 63);   // und auf Pixel (LED) abbilden
```

Sie können das mit einem variablen Widerstand ausprobieren, der wie in Abbildung 5-7 aus Kapitel 5 mit dem Analogpin 0 verbunden ist. Die Anzahl eingeschalteter LEDs ist dann proportional zum Sensorwert.

7.8 Bilder (Images) auf einer LED-Matrix darstellen

Problem

Sie wollen ein oder mehrere Bilder (Images) auf einer LED-Matrix ausgeben, etwa für eine Animation, bei der mehrere Images schnell hintereinander dargestellt werden.

Lösung

Diese Lösung kann die gleiche Verschaltung nutzen wie in Rezept 7.7. Der Sketch erzeugt den Effekt eines schlagenden Herzens, indem er die LEDs in Form eines Herzens aufleuchten lässt. Für jeden Herzschlag wird ein kleines Herz, gefolgt von einem großen Herzen, ausgegeben (die Images sind in Abbildung 7-9 zu sehen):

```
/*
 * matrixMpxAnimation Sketch
 * Animiert die Images zweier Herzen zu einem Herzschlag
 */

// Die Herzen werden als Bitmaps gespeichert - jedes Bit entspricht einer LED
// Bei einer 0 ist die LED aus, bei einer 1 ist sie an
byte bigHeart[] = {
 B01100110,
 B11111111,
```

```
 B11111111,
 B11111111,
 B01111110,
 B00111100,
 B00011000,
 B00000000};

byte smallHeart[] = {
 B00000000,
 B00000000,
 B00010100,
 B00111110,
 B00111110,
 B00011100,
 B00001000,
 B00000000};


const int columnPins[] = {  2, 3, 4, 5, 6, 7, 8, 9};
const int rowPins[]   = { 10,11,12,15,16,17,18,19};

void setup() {
 for (int i = 0; i < 8; i++)
 {
  pinMode(rowPins[i], OUTPUT);     // Alle LED-Pins sind Ausgänge
  pinMode(columnPins[i], OUTPUT);
  digitalWrite(columnPins[i], HIGH); // Spaltenpins von Masse trennen
 }
}

void loop() {
 int pulseDelay = 800 ;      // Wartezeit zwischen Herzschlägen in Millisekunden

 show(smallHeart, 80);       // Zeige kleines Herz für 100 ms
 show(bigHeart, 160);        // Gefolgt vom großen Herz für 200ms
 delay(pulseDelay);          // Dazwischen passiert nichts
}


// Zeigt einen Frame des Images, das im Array abgelegt ist,
// auf das der image-Parameter zeigt.
// Der Frame wird für die angegebene Dauer in Millisekunden wiederholt
void show( byte * image, unsigned long duration)
{
 unsigned long start = millis();        // Timing der Animation starten
 while (start + duration > millis())     // Ausgabe für die gewünschte Dauer
 {
  for(int row = 0; row < 8; row++)
  {
   digitalWrite(rowPins[row], HIGH);      // Zeile mit +5 Volt verbinden
   for(int column = 0; column < 8; column++)
   {
    boolean pixel = bitRead(image[row],column);
    if(pixel == 1)
    {
     digitalWrite(columnPins[column], LOW); // Spalte mit Masse verbinden
    }
```

```
      delayMicroseconds(300);           // Kleine Verzögerung für jede LED
      digitalWrite(columnPins[column], HIGH);  // Spalte von Masse trennen
    }
    digitalWrite(rowPins[row], LOW);       // LEDs trennen
  }
 }
}
```

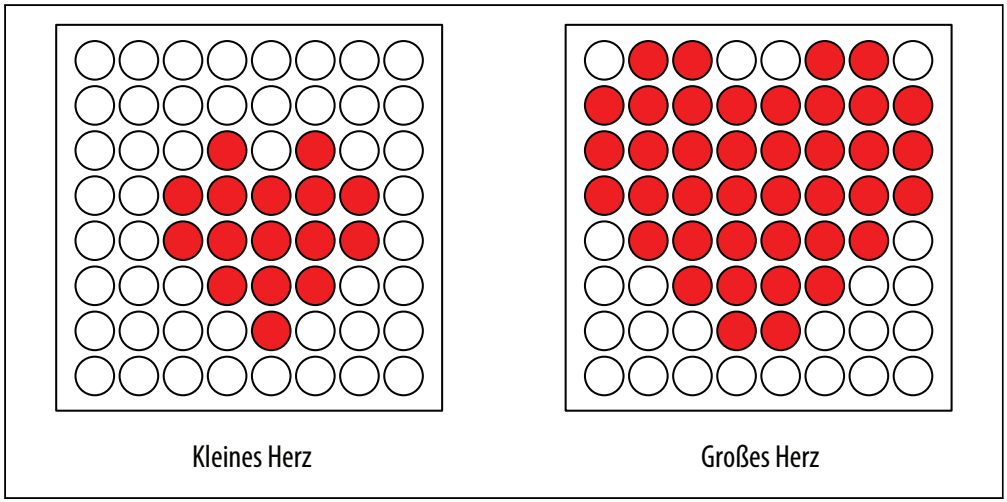


Abbildung 7-9: Die zwei Herzen, die bei jedem Herzschlag ausgegeben werden

Diskussion

Das Multiplexen (Schalten) der Spalten und Zeilen ähnelt Rezept 7.7, doch der an die LEDs geschriebene Wert basiert auf den Images, die in den Arrays `bigHeart` und `smallHeart` abgelegt sind. Jedes Element des Arrays repräsentiert ein *Pixel* (eine einzelne LED), und jede Array-Zeile repräsentiert eine Zeile der Matrix. Eine Zeile besteht aus acht Bits, die im Binärformat (das durch das große *B* am Anfang jeder Zeile festgelegt wird) angegeben werden. Ein Bit mit dem Wert `1` legt fest, dass die entsprechende LED an ist, die `0` bedeutet, sie ist aus. Der Animationseffekt entsteht, indem man zwischen den Arrays schnell hin und her wechselt.

Die `loop`-Funktion wartet kurz (800 Millisekunden) zwischen den Schlägen und ruft dann die `show`-Funktion zuerst mit dem `smallHeart`-Array und dann mit dem `bigHeart`-Array auf. Die `show`-Funktion geht jedes Element in allen Zeilen und Spalten durch und schaltet die LED ein, wenn das entsprechende Bit 1 ist. Die `bitRead`-Funktion (siehe Rezept 2.20) wird genutzt, um den Wert jedes Bits zu ermitteln.

Eine kurze Verzögerung von 300 Mikrosekunden zwischen den Pixeln gibt dem Auge genug Zeit, die LED wahrzunehmen. Das Timing wurde so gewählt, dass jedes Image oft genug wiederholt wird (50 mal pro Sekunde), damit man das Blinken nicht bemerkt.

Hier eine Variante, die die Geschwindigkeit des Herzschlags basierend auf dem Wert eines Sensors ändert. Sie können das mit einem variablen Widerstand am analogen Eingangspin 0 ausprobieren (siehe Rezept 5.6). Verwenden Sie die gleiche Verschaltung und den gleichen Code wie oben, ersetzen Sie nur die `loop`-Funktion durch den folgenden Code:

```
void loop() {
 sensorValue = analogRead(analogInPin);       // Sensorwert einlesen
 int pulseRate =  map(sensorValue,0,1023,40,240); // In Schläge pro Minute umwandeln
 int pulseDelay = (60000 / pulseRate );  // Wartezeit zwischen Herzschlägen in Millisekunden

 show(smallHeart, 80);        // Kleines Herz für 100 ms zeigen
 show(bigHeart, 160);        // Gefolgt vom großen Herzen für 200ms
 delay(pulseDelay);          // Dazwischen passiert nichts
}
```

Diese Version berechnet die Pause zwischen den Herzschlägen über die `map`-Funktion (siehe Rezept 5.7), die den Sensorwert in Schläge pro Minute umwandelt. Die Berechnung berücksichtigt die Zeit nicht, die es dauert, das Herz darzustellen, doch Sie können 240 Millisekunden (80 ms plus 160 ms für die beiden Images) abziehen, wenn Sie ein genaueres Timing wünschen.

Siehe auch

In 7.12 und 7.13 finden Sie Informationen, wie man Schieberegister zur Ansteuerung von LEDs nutzt, wenn man die Anzahl der Arduino-Pins reduzieren will, die zum Ansteuern einer LED-Matrix benötigt werden.

In 12.1 and 12.2 erfahren Sie mehr darüber, wie man die Zeit mit der `millis` -Funktion verwaltet.

7.9 Eine LED-Matrix ansteuern: Charlieplexing

Problem

Sie besitzen eine LED-Matrix und möchten die Anzahl der Pins minimieren, die benötigt werden, um sie ein- und auszuschalten.

Lösung

Charlieplexing ist eine spezielle Form des Multiplexings, die die Zahl der LEDs erhöht, die von einer Gruppe von Pins angesteuert werden können. Der folgende Sketch steuert sechs LEDs mit nur drei Pins (Abbildung 7-10 zeigt den Anschluss):

```
/*
 * Charlieplexing Sketch
 * Sechs LEDs nacheinander einschalten, die über die Pins 2, 3 und 4 angeschlossen sind
 */

byte pins[] = {2,3,4};  // Mit den LEDs verbundene Pins
```

```
// Die nächsten beiden Zeilen ermitteln die Anzahl der Pins und der LEDs aus dem obigen Array
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2} }; // Pins auf LEDs abbbilden


void setup()
{
  // Hier gibt es nichts zu tun
}

void loop(){
   for(int i=0; i < NUMBER_OF_LEDS; i++)
   {
     lightLed(i);  // Alle LEDs nacheinander einschalten
     delay(1000);
   }
}


// Diese Funktion schaltet die angegebene LED ein; die erste LED ist 0
void lightLed(int led)
{
 // Die nachfolgenden vier Zeilen wandeln die LED-Nummer in Pin-Nummern um
 int indexA = pairs[led/2][0];
 int indexB = pairs[led/2][1];
 int pinA = pins[indexA];
 int pinB = pins[indexB];

 // Schaltet alle Pins aus, die nicht mit der LED verbunden sind
 for(int i=0; i < NUMBER_OF_PINS; i++)
  if( pins[i] != pinA && pins[i] != pinB)
  {  // Ist dieser Pin nicht einer unserer Pins,
     pinMode(pins[i], INPUT);  // als Eingang festlegen
     digitalWrite(pins[i],LOW); // und Pullup ausschalten
  }
 // Nun die Pins für die angegebene LED einschalten
 pinMode(pinA, OUTPUT);
 pinMode(pinB, OUTPUT);
 if( led % 2 == 0)
 {
   digitalWrite(pinA,LOW);
   digitalWrite(pinB,HIGH);
 }
 else
 {
   digitalWrite(pinB,LOW);
   digitalWrite(pinA,HIGH);
 }
}
```

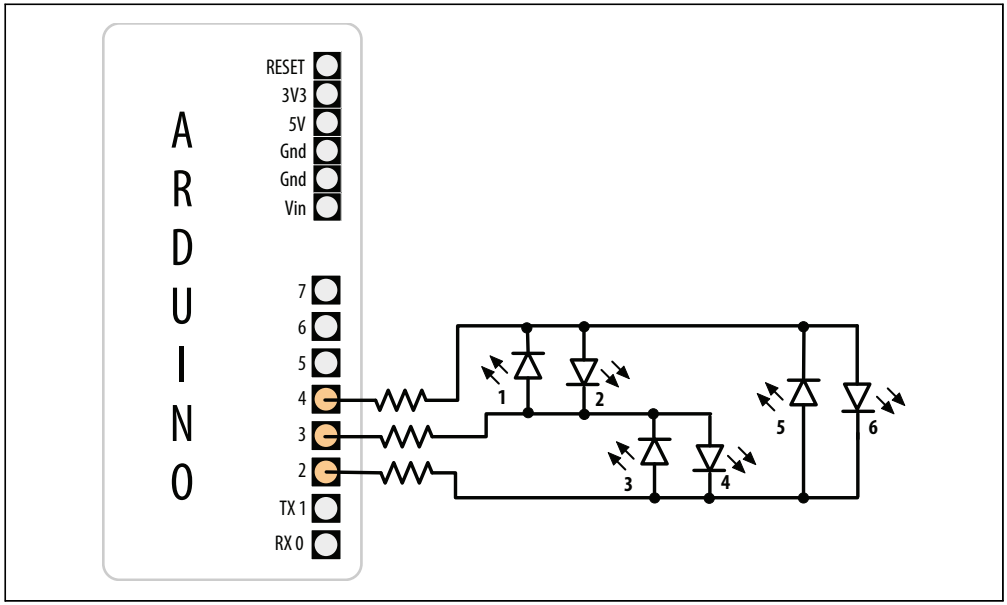


Abbildung 7-10: Ansteuerung von sechs LEDs mit nur drei Pins per Charlieplexing

Diskussion

Der Begriff *Charlieplexing* stammt von Charlie Allen (von Microchip Technology, Inc.), der diese Methode veröffentlicht hat. Die Technik nutzt die Tatsache aus, dass LEDs nur leuchten, wenn sie »richtig herum« angeschlossen sind (d.h., wenn die Anode »positiver« ist als die Kathode). Die nachfolgende Tabelle zeigt die LED-Nummer (siehe Abbildung 7-8), die bei gültigen Kombinationen der drei Pins leuchtet. `L` steht für `LOW`, `H` für `HIGH` und `i` für den `INPUT`-Modus. Schaltet man einen Pin in den `INPUT`-Modus, trennt man ihn von der Schaltung:

```
 Pins    LEDs
4 3 2  1 2 3 4 5 6
L L L  0 0 0 0 0 0
L H i  1 0 0 0 0 0
H L i  0 1 0 0 0 0
i L H  0 0 1 0 0 0
i H L  0 0 0 1 0 0
L i H  0 0 0 0 1 0
H i L  0 0 0 0 0 1
```

Sie können die Anzahl der LEDs mit nur einem zusätzlichen Pin auf 12 verdoppeln. Die ersten sechs LEDs werden dabei wie im obigen Beispiel angeschlossen. Den Anschluss der sechs zusätzlichen LEDs sehen Sie in Abbildung 7-11.

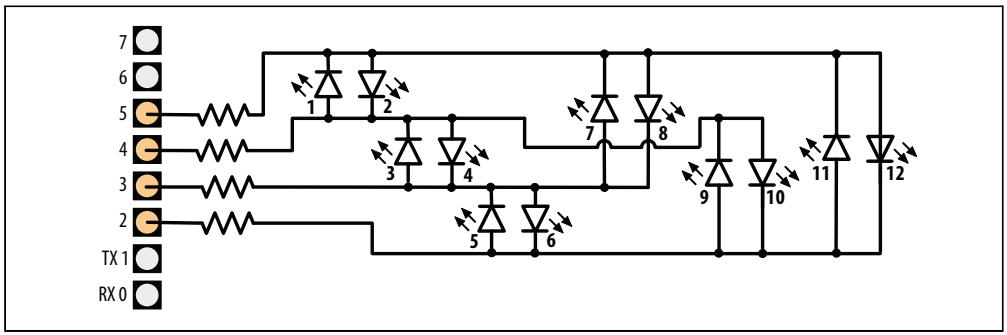


Abbildung 7-11: Charlieplexing mit vier Pins steuert 12 LEDs

Tragen Sie den zusätzlichen Pin im obigen Sketch in das `pins`-Array ein:

```
byte pins[] = {2,3,4,5}; // Mit den LEDs verbundene Pins
```

Tragen Sie die zusätzlichen Einträge in das `pairs`-Array ein:

```
byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2}, {2,3}, {1,3}, {0,3} };
```

Alles andere kann unverändert bleiben und die Schleife geht alle 12 LEDs durch, weil der Code die Anzahl der LEDs aus der Anzahl der Einträge im `pins`-Array bestimmt.

Da das Charlieplexing die Arduino-Pins so ansteuert, dass nur jeweils eine LED eingeschaltet ist, ist es schwieriger, mehrere LEDs quasi gleichzeitig leuchten zu lassen. Man kann das aber erreichen, indem man eine für das Charlieplexing modifizierte Multiplexing-Technik nutzt.

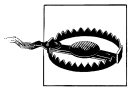
Der folgende Sketch erzeugt eine Balkenanzeige, indem es eine Reihe von LEDs ansteuert. Welche LEDs das sind, hängt vom Sensorwert am Analogpin 0 ab:

```
byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2} };

int ledStates = 0; //Enthält Zustand von bis zu 15 LEDs
int refreshedLed; // "Aufzufrischende" LED

void setup()
{
  // Hier ist nichts zu tun
}

void loop()
{
const int analogInPin = 0; // Analoger Eingangspin ist mit variablem Widerstand verbunden

 // Hier folgt der Code aus dem bargraph-Rezept
 int sensorValue = analogRead(analogInPin);      // Analogwert einlesen
 // Auf Anzahl der LEDs abbilden
 int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
```

```
 for (int led = 0; led < NUMBER_OF_LEDS; led++)
 {
  if (led < ledLevel ) {
   setState(led, HIGH);   // Pins unter Wert einschalten
  }
  else {
   setState(led, LOW);    // Pins über Wert ausschalten
  }
 }
 ledRefresh();

}

void setState( int led, boolean state)
{
 bitWrite(ledStates,led, state);
}

void ledRefresh()
{
 // Bei jedem Aufruf eine andere LED auffrischen.
 if( refreshedLed++ > NUMBER_OF_LEDS) // Zur nächsten LED wechseln
   refreshedLed = 0; // Wieder bei der ersten LED anfangen, wenn alle aufgefrischt wurden

 if( bitRead(ledStates, refreshedLed ) == HIGH)
    lightLed( refreshedLed );
}

// Diese Funktion entspricht der aus dem obigen Sketch
// Sie schaltet die angebene LED ein; die erste LED ist 0
void lightLed(int led)
{
 // Die folgenden vier Zeilen wandeln die LED-Nummer in die Pin-Nummern um
 int indexA = pairs[led/2][0];
 int indexB = pairs[led/2][1];
 int pinA = pins[indexA];
 int pinB = pins[indexB];

 // Alle Pins ausschalten, die nicht mit der angegebenen LED verbunden sind
 for(int i=0; i < NUMBER_OF_PINS; i++)
  if( pins[i] != pinA && pins[i] != pinB)
  { // Ist der Pin nicht einer unserer Pins,
    pinMode(pins[i], INPUT);  // als Eingang festlegen
    digitalWrite(pins[i],LOW); // und Pullup ausschalten
  }
 // Nun die Pins für die angegebene LED einschalten
 pinMode(pinA, OUTPUT);
 pinMode(pinB, OUTPUT);
 if( led % 2 == 0)
 {
  digitalWrite(pinA,LOW);
  digitalWrite(pinB,HIGH);
 }
 else
```

```
 {
   digitalWrite(pinB,LOW);
   digitalWrite(pinA,HIGH);
 }
}
```

Der Sketch nutzt die Bitwerte der Variablen `ledStates` für den Zustand der LEDs (0 für aus und 1 für an). Die `refresh`-Funktion überprüft jedes Bit und schaltet die LEDs ein, wenn es auf 1 gesetzt ist. Die `refresh`-Funktion muss immer wieder schnell aufgerufen werden, da die LEDs sonst flackern würden.



Verzögerungsschleifen im Code können den Phi-Effekt (Trägheit des Auges) behindern, der dafür verantwortlich ist, dass wir das Flackern der LEDs nicht sehen.

Sie können einen Interrupt nutzen, um die `refresh`-Funktion im Hintergrund abzuarbeiten (ohne die Funktion in `loop` explizit aufrufen zu müssen). Timer-Interrupts werden in Kapitel 18 behandelt, aber hier gibt es schon mal einen Vorgeschmack darauf, wie man einen Interrupt nutzt, um das Auffrischen der LEDs zu erledigen. Wir verwenden eine Bibliothek namens FrequencyTimer2 (die im Arduino Playground verfügbar ist), um den Interrupt einzurichten:

```
#include <FrequencyTimer2.h> // Einbinden dieser Bibliothek, um refresh zu ermöglichen


byte pins[] = {2,3,4};
const int NUMBER_OF_PINS = sizeof(pins)/ sizeof(pins[0]);
const int NUMBER_OF_LEDS = NUMBER_OF_PINS * (NUMBER_OF_PINS-1);

byte pairs[NUMBER_OF_LEDS/2][2] = { {0,1}, {1,2}, {0,2} };

int ledStates = 0; // Enthält Zustände von bis zu 15 LEDs
int refreshedLed; // Die aufzufrischende LED
void setup()
{
 FrequencyTimer2::setPeriod(20000/ NUMBER_OF_LEDS); // Zeitintervall festlegen
 // Die nächste Zeile legt fest, welche Funktion FrequencyTimer2 aufrufen soll (ledRefresh)
 FrequencyTimer2::setOnOverflow(ledRefresh);
 FrequencyTimer2::enable();
}

void loop()
{
const int analogInPin = 0; // Analoger Eingangspin ist mit variablem Widerstand verbunden
```

```
 // Hier folgt der Code aus dem bargraph-Rezept
 int sensorValue = analogRead(analogInPin);      // Analogwert einlesen
 // Auf Anzahl der LEDs abbbilden
 int ledLevel = map(sensorValue, 0, 1023, 0, NUMBER_OF_LEDS);
 for (int led = 0; led < NUMBER_OF_LEDS; led++)
 {
  if (led < ledLevel ) {
   setState(led, HIGH);   // Pins unter Wert einschalten
  }
  else {
   setState(led, LOW);    // Pins über Wert ausschalten
  }
 }
  // Die LEDs werden nicht mehr in der Schleife aufgefrischt. Das wird von FrequencyTimer2
übernommen
}

// Der restliche Code entspricht dem aus dem vorigen Beispiel
```

Der Timer für die FrequencyTimer2-Bibliothek wird auf 1,666 Mikrosekunden gesetzt (20 ms durch 12, also die Anzahl der LEDs). Die Methode `FrequencyTimer2setOnOverflow` legt die Funktion fest (`ledRefresh`), die aufgerufen soll, wenn der Timer »auslöst«.

Siehe auch

Das LOL-Board ist ein Arduino-Shield, das eine 9×14-Matrix (126 LEDs) per Charlieplexing ansteuert. Es ist ein schönes Beispiel dafür, was mit dieser Technik möglich ist, wenn man bei der Hard- und Software über die üblichen Design-Beschränkungen hinausgeht: *http://jimmieprodgers.com/kits/lolshield/makelolshield/*.

Kapitel 18 enthält weitere Informationen zu Timer-Interrupts.

7.10 Eine 7-Segment-LED-Anzeige ansteuern

Problem

Sie wollen Zahlen auf einer 7-Segment-Anzeige darstellen.

Lösung

Der folgende Sketch gibt Ziffern zwischen 0 bis 9 auf einer einstelligen 7-Segment-Anzeige aus. Den Anschluss sehen Sie in Abbildung 7-12. Die Ausgabe wird durch das Einschalten einer Kombination von Segmenten erreicht, die die Ziffern repräsentieren:

```
/*
 * SevenSegment Sketch
 * Zeigt Ziffern zwischen 0 bis 9 auf einer einstelligen Anzeige an
 * Das Beispiel zählt die Sekunden von 0 bis 9
 */
```

```
// Die Bits repräsentieren die Segmente A bis G (und den Dezimalpunkt) für die Ziffern 0-9
const byte numeral[10] = {
 //ABCDEFG /dp
 B11111100, // 0
 B01100000, // 1
 B11011010, // 2
 B11110010, // 3
 B01100110, // 4
 B10110110, // 5
 B00111110, // 6
 B11100000, // 7
 B11111110, // 8
 B11100110, // 9
};

// Pins für Dezimalpunkt und alle Segmente
//               dp,G,F,E,D,C,B,A
const int segmentPins[8] = { 5,8,9,7,6,4,3,2};

void setup()
{
 for(int i=0; i < 8; i++)
 {
  pinMode(segmentPins[i], OUTPUT); // Segment- und DP-Pins als Ausgang festlegen
 }
}


void loop()
{
 for(int i=0; i <= 10; i++)
 {
  showDigit(i);
  delay(1000);
 }
 // Der letzte Wert wurde erreicht und das Display wird ausgeschaltet
 delay(2000);  // Zwei Sekunden Pause mit ausgeschaltetem Display
}

// Gibt eine Ziffer zwischen 0 und 9 auf einer 7-Segment-Anzeige aus
// Alle Werte außerhalb von 0-9 schalten die Anzeige aus
void showDigit( int number)
{
 boolean isBitSet;

 for(int segment = 1; segment < 8; segment++)
 {
  if( number < 0 || number > 9){
   isBitSet = 0;  // Alle Segmente ausschalten
  }
  else{
   // isBitSet ist "wahr", wenn das angegebene Bit 1 ist
   isBitSet = bitRead(numeral[number], segment);
  }
```

```
    isBitSet = ! isBitSet; // Diese Zeile bei gemeinsamer Kathode entfernen
    digitalWrite( segmentPins[segment], isBitSet);
  }
}
```

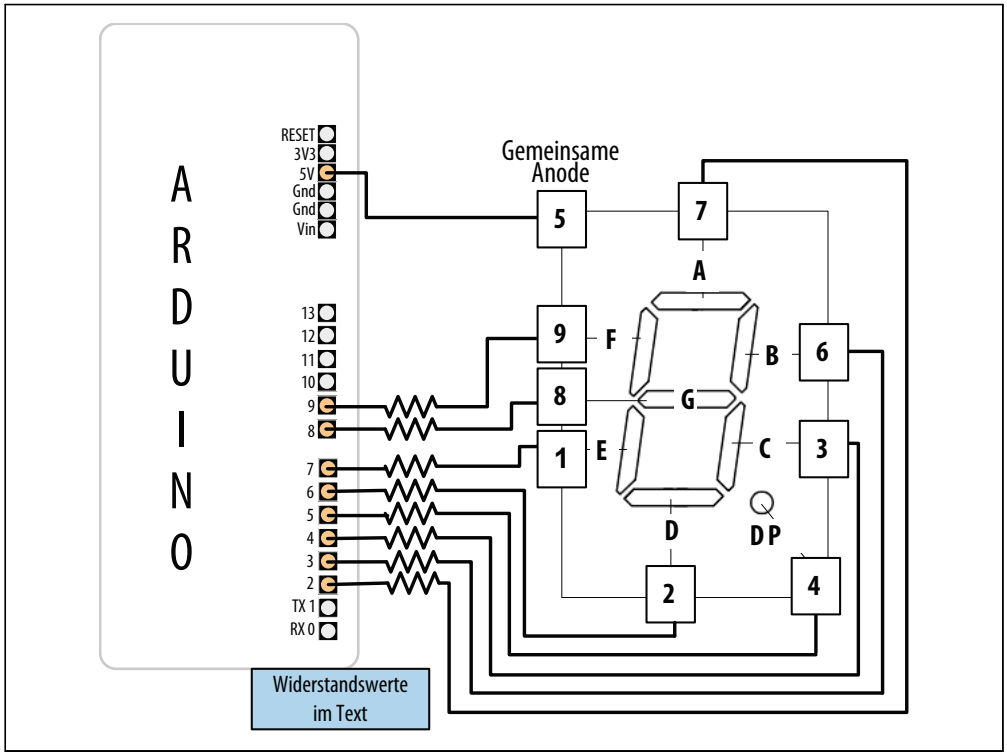


Abbildung 7-12: Anschluss einer 7-Segment-Anzeige

Diskussion

Die für jede Ziffer zu aktivierenden Segmente werden im Array `numeral` vorgehalten. Für jede Ziffer wird ein Byte genutzt, bei dem jedes Bit eines der sieben Segmente (oder den Dezimalpunkt) repräsentiert.

Das Array namens `segmentPins` enthält die Pins, die jedem Segment zugeordnet sind. Die Funktion `showDigit` prüft, ob die Zahl im Bereich von 0 bis 9 liegt. Handelt es sich um eine gültige Zahl, wird dann jedes Segment-Bit untersucht und eingeschaltet, wenn es gesetzt (1) ist. In Rezept 3.12 erfahren Sie mehr über die `bitRead`-Funktion.

Wie in Rezept 7.4 erwähnt, ist ein Pin `HIGH`, wenn man ein Segment bei einer Anzeige mit gemeinsamer Kathode einschaltet. Bei einer Anzeige mit gemeinsamer Anode ist er `LOW`. Unser Code ist für eine Anzeige mit gemeinsamer Anode gedacht, weshalb er den Wert wir folgt invertiert (d.h. 0 auf 1 und 1 auf 0 setzt):

```
isBitSet = ! isBitSet; // Diese Zeile bei gemeinsamer Kathode entfernen
```

Das ! ist der Negationsoperator – siehe Rezept 2.20. Wenn Ihre Anzeige eine gemeinsame Kathode nutzt (d.h., alle Kathoden sind miteinander verbunden; sehen Sie auf dem Datenblatt nach, wenn Sie sich nicht sicher sind), können Sie diese Zeile entfernen.

7.11 Mehrstellige 7-Segment-LED-Anzeigen ansteuern: Multiplexing

Problem

Sie wollen Zahlen auf einer 7-Segment-Anzeige mit zwei oder mehr Ziffern darstellen.

Lösung

Mehrstellige 7-Segment-Anzeigen arbeiten üblicherweise mit Multiplexing. In früheren Rezepten haben wir Zeilen und Spalten von LEDs zur einem Array verschaltet. Hier werden die entsprechenden Segmente für alle Ziffern miteinander verbunden (siehe Abbildung 7-13):

```
/*
 * SevenSegmentMpx Sketch
 * Stellt Zahlen zwischen 0 bis 9999 auf einer vierstelligen Anzeige dar
 * Das Beispiel gibt den Wert eines Sensors aus, der mit einem Analogeingang verbunden ist
*/

// Die Bits repräsentieren die Segmente A bis G (und den Dezimalpunkt) für die Ziffern 0-9
const int numeral[10] = {
 //ABCDEFG /dp
 B11111100, // 0
 B01100000, // 1
 B11011010, // 2
 B11110010, // 3
 B01100110, // 4
 B10110110, // 5
 B00111110, // 6
 B11100000, // 7
 B11111110, // 8
 B11100110, // 9
};

// Pins für Dezimalpunkt und alle Segmente
              // dp,G,F,E,D,C,B,A
const int segmentPins[ ] = { 4,7,8,6,5,3,2,9};

const int nbrDigits= 4; // Anzahl der Ziffern der LED-Anzeige

                //dig 1 2 3 4
const int digitPins[nbrDigits] = { 10,11,12,13};
```

```
void setup()
{
 for(int i=0; i < 8; i++)
  pinMode(segmentPins[i], OUTPUT); // Segment- und DP-Pins als Ausgang festlegen

 for(int i=0; i < nbrDigits; i++)
  pinMode(digitPins[i], OUTPUT);
}


void loop()
{
 int value = analogRead(0);
 showNumber(value);
}

void showNumber( int number)
{
 if(number == 0)
  showDigit( 0, nbrDigits-1) ; // 0 in der rechten Ziffer ausgeben
 else
 {
  // Wert für jede Ziffer ausgeben
  // Linke Ziffer ist 0, rechte ist 1 kleiner als Anzahl der Stellen
  for( int digit = nbrDigits-1; digit >= 0; digit--)
  {
   if(number > 0)
   {
    showDigit( number % 10, digit)  ;
    number = number / 10;
   }
  }
 }
}

// Angegebene Ziffer in der 7-Segment-Anzeige an der angegebenen Stelle ausgeben
void showDigit( int number, int digit)
{
 digitalWrite( digitPins[digit], HIGH );
 for(int segment = 1; segment < 8; segment++)
 {
  boolean isBitSet = bitRead(numeral[number], segment);
  // isBitSet ist "wahr", wenn angegebenes Bit 1 ist
  isBitSet = ! isBitSet; // Diese Zeile bei gemeinsamer Kathode entfernen
  digitalWrite( segmentPins[segment], isBitSet);
 }
 delay(5);
 digitalWrite( digitPins[digit], LOW );
}
```

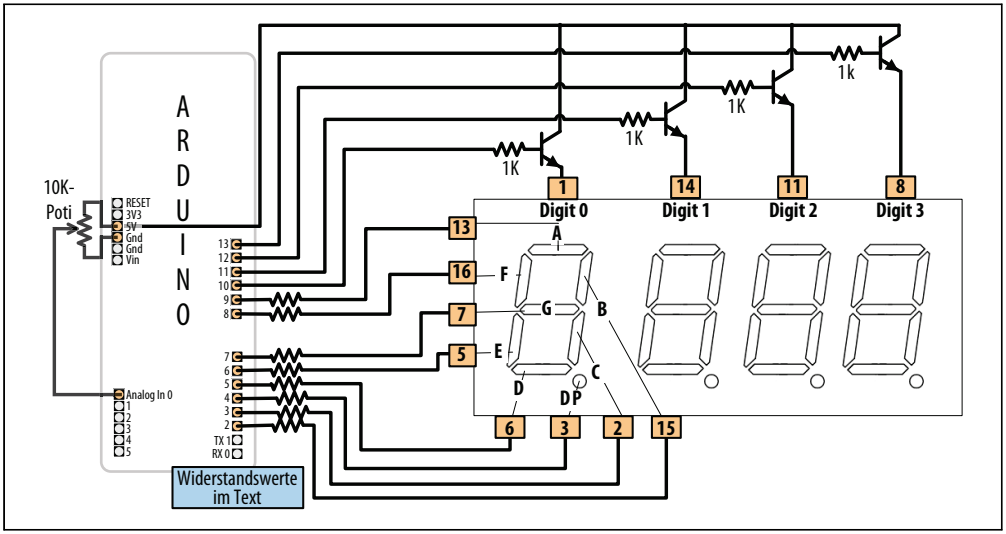


Abbildung 7-13: Anschluss einer mehrstelligen 7-Segment-Anzeige (LTC-2623)

Diskussion

Dieser Sketch nutzt eine `showDigit`-Funktion, die der aus Rezept 7.10 ähnelt. Hier wird der Funktion neben der Ziffer auch die Position in der Anzeige übergeben. Die Logik zur Ansteuerung ist die gleiche, der Code setzt aber zusätzlich den Pin für die richtige Position in der Anzeige auf `HIGH`, so dass nur diese Ziffer dargestellt wird (beachten Sie hierzu die früheren Erklärungen zum Multiplexing).

7.12 Mehrstellige 7-Segment-LED-Anzeigen mit MAX7221-Schieberegistern ansteuern

Problem

Sie wollen mehrstellige 7-Segment-Anzeigen ansteuern, aber die Zahl der benötigten Arduino-Pins reduzieren.

Lösung

Diese Lösung verwendet den beliebten MAX7221 LED-Treiber zur Ansteuerung vierstelliger Displays mit gemeinsamer Kathode, wie etwa das Lite-On LTC-4727JR (Digi-Key 160-1551-5-ND). Der MAX7221 bietet eine einfachere Lösung als Rezept 7.11, weil er das gesamte Multiplexing und die Dekodierung der Ziffern in Hardware erledigt.

Der Sketch gibt Zahlen zwischen 0 und 9999 aus (Abbildung 7-14 zeigt den Anschluss):

```
/*
 Max7221_digits
 */

#include <SPI.h>  // Arduino SPI-Bibliothek. Eingeführt mit Arduino Version 0019

const int slaveSelect = 10; // Pin zur Aktivierung des aktiven Slaves

const int numberOfDigits = 4; // An die Anzahl der Ziffern anpassen

const int maxCount     = 9999;

int number = 0;

void setup()
{
 Serial.begin(9600);
 SPI.begin();  // initialize SPI
 pinMode(slaveSelect, OUTPUT);
 digitalWrite(slaveSelect,LOW);  //Slave wählen
 // 7221 zur Anzeige von 7-Segment-Daten vorbereiten - siehe Datenblatt
 sendCommand(12,1);  // Normaler Modus (Voreinstellung ist Shutdown-Modus);
 sendCommand(15,0);  // Display-Test aus
 sendCommand(10,8);  // Mittlere Helligkeit (Wertebereich zwischen 0-15)
 sendCommand(11,numberOfDigits);  // 7221 digit scan limit command
 sendCommand(9,255); // Dekodierbefehl, verwende Standard-7-Segment-Ziffern
 digitalWrite(slaveSelect,HIGH);  //Slave deaktivieren
}

void loop()
{
 // Ausgabe einer Zahl vom seriellen Port
 // Zahl wird mit Zeilenende-Zeichen abgeschlossen
 if(Serial.available())
 {
   char ch  = Serial.read();
   if( ch == '\n')
   {
    displayNumber(number);
    number = 0;
   }
   else
   number = (number * 10) + ch - '0'; // Details in Kapitel 4
 }
}

//Anzeige von bis zu vier Ziffern in einem 7-Segment-Display
void displayNumber( int number)
{
 for(int i = 0; i < numberOfDigits; i++)
 {
  byte character = number % 10;  // Wert der äußersten rechten Ziffer bestimmen
  if(number == 0 && i > 0)
    character = 0xf;  // 7221 löscht die Segmente beim Empfang des Wertes
```

```
    // Ziffer der Zahl als Befehl senden, erste Ziffer ist Befehl 1
    sendCommand(numberOfDigits-i, character);
    number = number / 10;
  }
}

void sendCommand( int command, int value)
{
 digitalWrite(slaveSelect,LOW); //Chip-Select ist bei LOW aktiv
 //2-Byte-Datentransfer zum 7221
 SPI.transfer(command);
 SPI.transfer(value);
 digitalWrite(slaveSelect,HIGH); //Chip freigeben, Übertragungsende signalisieren
}
```

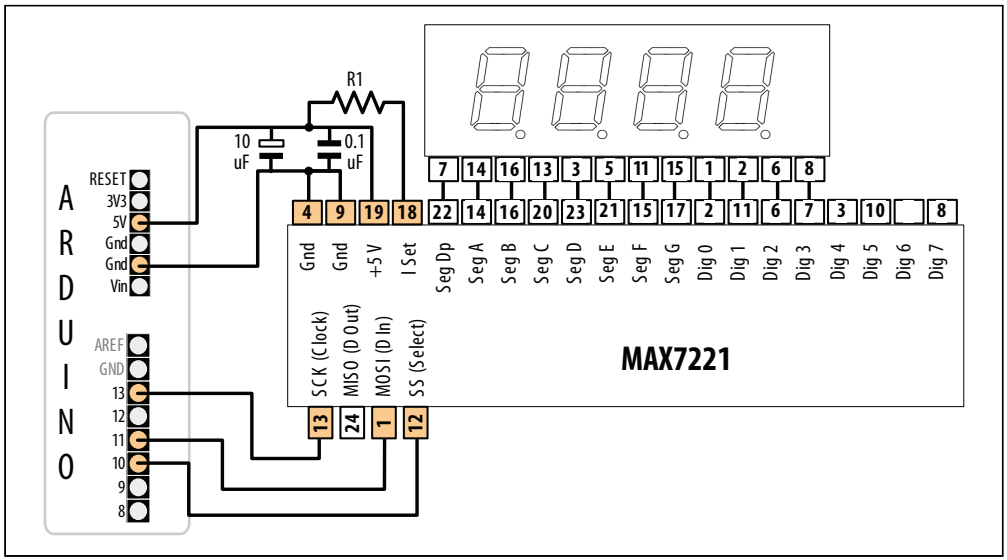


Abbildung 7-14: Ansteuerung einer mehrstelligen 7-Segment-Anzeige (gemeinsame Kathode) mit MAX7221

Lösung

Dieses Rezept nutzt die Arduino SPI-Kommunikation, um mit dem MAX7221-Chip zu kommunizieren. Kapitel 13 behandelt SPI im Detail, und Rezept 13.8 erläutert den verwendeten SPI-spezifischen Code.

Der Sketch zeigt die Zahl an, wenn bis zu vier Ziffern über den seriellen Port empfangen wurden – eine Erläuterung des Codes für die serielle Schnittstelle in `loop` finden Sie in Kapitel 4. Die Funktion `displayNumber` extrahiert (von rechts nach links) den Wert jeder Ziffer und sendet sie über die `sendCommand`-Funktion an den MAX7221.

Die Verschaltung nutzt eine vierstellige 7-Segment-Anzeige, aber Sie können ein- oder zweistellige Anzeigen mit bis zu insgesamt acht Ziffern zusammenschalten. Wenn Sie

mehrere Anzeigen kombinieren, müssen die Pins für die jeweiligen Segmente miteinander verbunden sein. (Rezept 13.8 zeigt den Anschluss einer gängigen zweistelligen Anzeige.)

Die MAX72xx-Chips sind für Anzeigen mit gemeinsamer Kathode konzipiert. Die Anode jedes Segments ist an einem separaten Pin verfügbar und die Kathoden aller Segmente jeder Ziffer sind miteinander verbunden.

7.13 Eine LED-Matrix mit MAX72xx-Schieberegistern ansteuern

Problem

Sie müssen eine 8×8-LED-Matrix ansteuern und wollen die Anzahl der dazu benötigten Arduino-Pins minimieren.

Lösung

Wie in Rezept 7.12 kann man ein Schieberegister auch nutzen, um die Anzahl der Pins zu reduzieren, die zur Ansteuerung einer LED-Matrix benötigt werden. Diese Lösung nutzt die beliebten LED-Treiber MAX7219 oder MAX7221. Schließen Sie den Arduino, die Matrix und den MAX72xx wie in Abbildung 7-15 an.

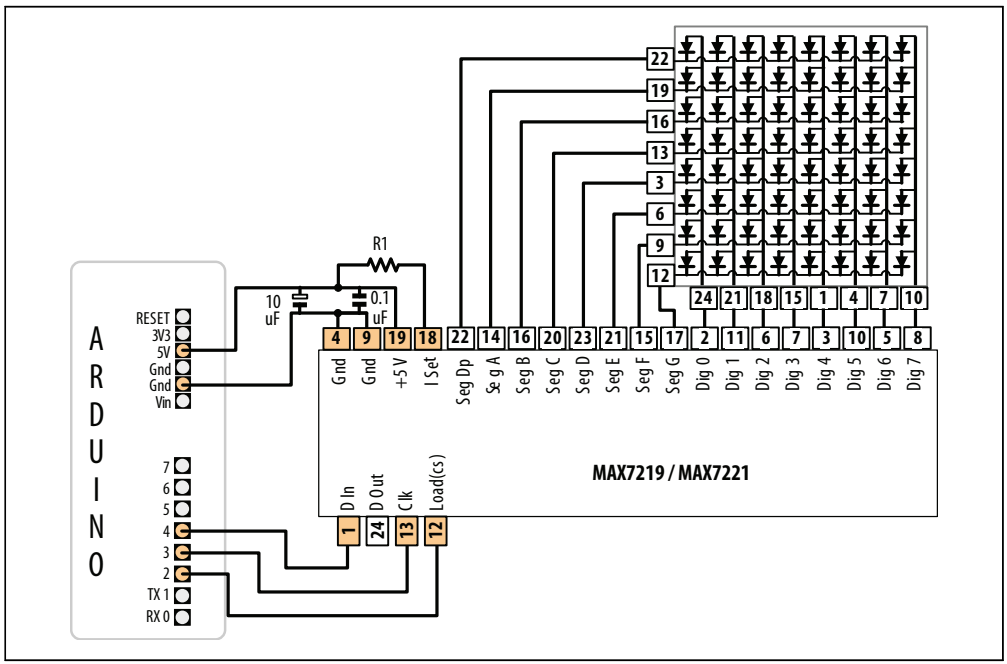


Abbildung 7-15: Ansteuerung einer 8×8-LED-Matrix mit einem MAX72xx

Der Sketch basiert auf der Arduino-Bibliothek hello_matrix von Nicholas Zambetti. Die Pin-Nummern wurden so angepasst, dass sie mit der an anderen Stellen dieses Kapitels genutzten Verschaltung übereinstimmen. Sie nutzt die Sprite- und Matrix-Bibliotheken, die mit Arduino-Releases vor 1.0 ausgeliefert wurden. Wenn Sie Arduino 1.0 verwenden und diese Bibliotheken im Arduino Playground nicht finden, können Sie die Bibliotheken aus der Release 0022 nutzen *http://arduino.cc/en/Main/Software*.

```
#include <Sprite.h>
#include <Matrix.h>

// Hello Matrix
// von Nicholas Zambetti <http://www.zambetti.com>

// Demonstriert die Verwendung der Matrix-Bibliothek
// für MAX7219 LED-Matrix-Controller
// Zeichnet ein Gesicht auf der Matrix

const int loadPin  = 2;
const int clockPin = 3;
const int dataPin  = 4;

Matrix myMatrix = Matrix(dataPin, clockPin, loadPin); // create a new Matrix

void setup()
{
}

void loop()
{
 myMatrix.clear(); // clear display

 delay(1000);

 // turn some pixels on
 myMatrix.write(1, 5, HIGH);
 myMatrix.write(2, 2, HIGH);
 myMatrix.write(2, 6, HIGH);
 myMatrix.write(3, 6, HIGH);
 myMatrix.write(4, 6, HIGH);
 myMatrix.write(5, 2, HIGH);
 myMatrix.write(5, 6, HIGH);
 myMatrix.write(6, 5, HIGH);

 delay(1000);
}
```

Diskussion

Sie erzeugen eine Matrix, indem Sie die Pin-Nummern für die Daten-, Lade- und Clock-Pins übergeben. `loop` verwendet die Methode `write`, um die Pixel einzuschalten. Die `clear`-Methode schaltet die Pixel aus. `write` verwendet drei Parameter: Die ersten beiden bestimmen die Spalte und Zeile (`x` und `y`) der LED, und der dritte (`HIGH` oder `LOW`) schaltet die LED ein oder aus.

Die hier verwendeten Pin-Nummern steuern die grünen LEDs der zweifarbigen 8×8-Matrix an, die es von folgenden Anbietern gibt:

SparkFun COM-00681
NKC Electronics COM-0006

Der Widerstand (mit der Bezeichnung R1 in Abbildung 7-15) wird verwendet, um den Maximalstrom zu kontrollieren, der die LED antreibt. Das MAX72xx-Datenblatt enthält eine Tabelle, die eine Reihe von Werten aufführt (siehe Tabelle 7-3).

Tabelle 7-3: Tabelle mit Widerstandswerten (aus MAX72xx-Datenblatt)

LED-Fluss-Spannung					
Strom	**1.5V**	**2.0V**	**2.5V**	**3.0V**	**3.5V**
40 mA	12 kΩ	12 kΩ	11 kΩ	10 kΩ	10 kΩ
30 mA	18 kΩ	17 kΩ	16 kΩ	15 kΩ	14 kΩ
20 mA	30 kΩ	28 kΩ	26 kΩ	24 kΩ	22 kΩ
10 mA	68 kΩ	64 kΩ	60 kΩ	56 kΩ	51 kΩ

Die grüne LED in der LED-Matrix aus Abbildung 7-15 hat eine Fluss-Spannung von 2,0 Volt und einen Durchlass-Strom von 20 mA. Tabelle 7-3 gibt 28K Ohm an, doch als kleine zusätzliche Sicherheit ist ein Widerstand von 30K oder 33K eine gute Wahl. Die Kondensatoren (0.1 μf und 10 μf) (DL) 0.1 uf werden benötigt, um Spannungsspitzen zu vermeiden, die beim Ein- und Ausschalten der LEDs auftreten – sehen Sie sich Rezept 17.2 in Anhang C an, wenn Sie mit dem Anschluss von Entstörkondensatoren nicht vertraut sind.

Siehe auch

Dokumentation der Matrix-Bibliothek: *http://wiring.org.co/reference/libraries/Matrix/index.html*

Dokumentation der Sprite-Bibliothek: *http://wiring.org.co/reference/libraries/Sprite/index.html*

MAX72xx-Datenblatt: *http://pdfserv.maxim-ic.com/en/ds/MAX7219-MAX7221.pdf*

7.14 Die Anzahl analoger Ausgänge mit PWM-Extender-Chips (TLC5940) erhöhen

Problem

Sie wollen individuell die Helligkeit von mehr LEDs regeln, als vom Arduino unterstützt werden (6 bei einem Standard-Board und 12 beim Mega).

Lösung

Der TLC5940-Chip steuert bis zu 16 LEDs über nur 5 Datenpins an. Abbildung 7-16 zeigt den Anschluss. Der Sketch basiert auf der exzellenten Tlc5940-Bibliothek von Alex Leone (*acleone@gmail.com*). Sie können die Bibliothek von *http://code.google.com/p/tlc5940arduino/* herunterladen:

```
/*
 * TLC Sketch
 * Erzeugt einen "Knight Rider"-Effekt mit LEDs an allen TLC-Ausgängen
 * Diese Version setzt einen TLC mit 16 LEDs voraus
 */

#include "Tlc5940.h"

void setup()
{
 Tlc.init(); // TLC-Bibliothek initialisieren
}

void loop()
{
 int direction = 1;
 int intensity = 4095; // Die mögliche Helligkeit liegt zwischen 0 und 4095, volle Helligkeit bei
4095
 int dim = intensity / 4; // 1/4 des Werts dimmt die LED
 for (int channel = 0; channel < 16; channel += direction) {
 // Die folgenden TLC-Befehle legen Werte fest, die von der update-Methode geschrieben werden
  Tlc.clear(); // Alle LEDs ausschalten
  if (channel == 0) {
   direction = 1;
  }
  else {
   Tlc.set(channel - 1, dim); // Helligkeit der vorigen LED
  }
  Tlc.set(channel, intensity); // Max. Helligkeit dieser LED
  if (channel < 16){
   Tlc.set(channel + 1, dim); // Nächste LED dimmen
  }
  else {
   direction = -1;
  }

  Tlc.update(); // Diese Methode sendet Daten an den TLC-Chip, um die LEDs anzusteuern
  delay(75);
 }
}
```

Diskussion

Der Sketch geht jeden Kanal (jede LED) durch. Er dimmt die vorherige LED, setzt den aktuellen Kanal auf die volle Helligkeit und dimmt den nächsten Kanal. Die LEDs werden über einige wenige Kern-Methoden angesteuert.

Die Methode `Tlc.init` initialisiert die `Tlc`-Funktionen, bevor alle anderen Funktionen aufgerufen werden können.

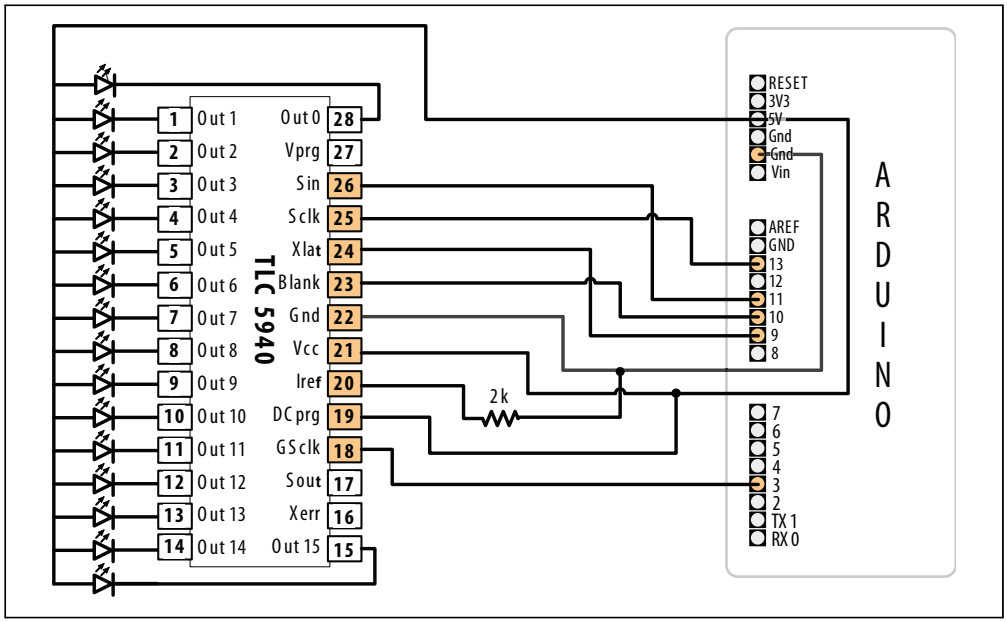


Abbildung 7-16: Sechzehn LEDs mit externem PWM ansteuern

Die folgenden Funktionen greifen nur nach dem Aufruf der `update()`-Methode:

`Tlc.clear`
: Schaltet alle Kanäle aus.

`Tlc.set`
: Setzt die Helligkeit des angegebenen Kanals auf den angegebenen Wert.

`Tlc.setAll`
: Setzt alle Kanäle auf den angegebenen Wert.

`Tlc.update`
: Sendet die Änderungen der vorangegangenen Befehle an den TLC-Chip.

Weitere Funktionen stehen in der Bibliothek zur Verfügung. Beachten Sie den Link auf die Referenz am Ende des Rezepts.

Der 2K-Widerstand zwischen TLC-Pin 20 (Iref) und Masse lässt etwa 20 mA durch jede LED fließen. Sie können den Widerstandswert für andere Ströme (in Milliampere) mit der Formel R = 40.000 / mA berechnen. R ist 1 Ohm, und die Berechnung ist nicht von der Steuerspannung der LED abhängig.

Sollen die LEDs ausgehen, wenn der Arduino zurückgesetz wird, verbinden Sie einen Pullup-Widerstand (10K) mit +5V und BLANK (Pin 23 des TLC und Arduino-Pin 10).

Hier eine Variante, die einen Sensorwert nutzt, um die maximale LED-Helligkeit festzulegen. Sie können das mit einem variablen Widerstand testen, der wie in Abbildung 7-13 oder Abbildung 7-17 angeschlossen ist:

```
#include "Tlc5940.h"

const int sensorPin = 0; // Mit Analogeingang 0 verbundener Sensor

void setup()
{
 Tlc.init(); // TLC-Bibliothek initialisieren
}

void loop()
{
 int direction = 1;
 int sensorValue = analogRead(0); // Sensorwert einlesen
 int intensity = map(sensorValue, 0,1023, 0, 4095); // Auf TLC-Wertebereich abbilden
 int dim = intensity / 4; // 1/4 des Werts dimmt die LED
 for (int channel = 0; channel < NUM_TLCS * 16; channel += direction) {
 // Die folgenden TLC-Befehle legen Werte fest, die von der update-Methode geschrieben werden
  Tlc.clear(); // Alle LEDs ausschalten
  if (channel == 0) {
   direction = 1;
  }
  else {
   Tlc.set(channel - 1, dim); // Helligkeit der vorigen LED
  }
  Tlc.set(channel, intensity); // Max. Helligkeit dieser LED
  if (channel != NUM_TLCS * 16 - 1) {
   Tlc.set(channel + 1, dim); // Nächste LED dimmen
  }
  else {
   direction = -1;
  }

  Tlc.update(); // Diese Methode sendet die Daten an den TLC-Chip, um die LEDs anzusteuern
  delay(75);
 }
}
```

Diese Version erlaubt auch mehrere TLC-Chips, wenn Sie mehr als 16 LEDs ansteuern wollen. Dazu werden die TLC-Chips miteinander »verkettet« (»daisy-chaining«), d.h., man verbindet Sout (Pin 17) des ersten TLC mit Sin (Pin 26) des nächsten. Sin (Pin 26) des ersten TLC-Chips wird mit Arduino-Pin 11 verbunden (siehe Abbildung 7-16).

Die folgenden Pins müssen miteinander verbunden werden, wenn man mehrere TLC-Chips verkettet:

- Arduino-Pin 9 mit XLAT (Pin 24) jedes TLCs
- Arduino-Pin 10 mit BLANK (Pin 23) jedes TLCs
- Arduino-Pin 13 mit SCLK (Pin 25) jedes TLCs

Jeder TLC benötigt einen eigenen Widerstand zwischen Iref (Pin 20) und Masse.

Sie müssen den Wert der Konstanten `NUM_TLCS` in der Tlc5940-Bibliothek an die Anzahl der von Ihnen genutzten Chips anpassen.

Siehe auch

Unter *http://code.google.com/p/tlc5940arduino/* können Sie diese Bibliothek herunterladen. Dort finden Sie auch die Dokumentation.

7.15 Ein analoges Anzeigeinstrument nutzen

Problem

Sie wollen den Zeiger einer Analoganzeige aus dem Sketch heraus steuern. Schwankende Messwerte lassen sich auf einer Analoganzeige leichter Interpretieren und verleihen Ihrem Projekt einen coolen Retro-Look.

Lösung

Verbinden Sie die Anzeige über einen Vorwiderstand (5K-Ohm sind für ein 1 mA-Meter üblich) mit einem analogen (PWM) Ausgang (siehe Abbildung 7-17).

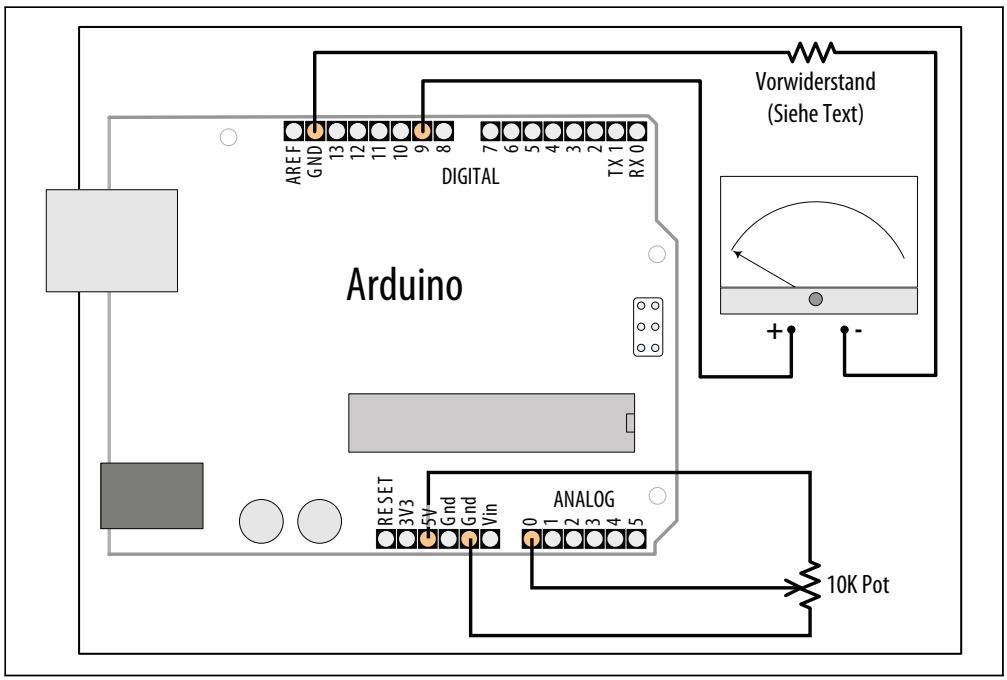


Abbildung 7-17: Eine analoge Anzeige ansteuern

Die Bewegung der Anzeige entspricht der Position eines Potentiometers (variablen Widerstands):

```
/*
 * AnalogMeter Sketch
 * Steuert eine Analoganzeige über einen Arduino-PWM-Pin
 * Der Pegel der Anzeige wird dabei durch den variablen Widerstand am Analogeingang bestimmt
 */

const int analogInPin = 0; // Analoger Eingang für Poti
const int analogMeterPin = 9; // Analoger Ausgang für Anzeige

int sensorValue = 0;     // Vom Poti eingelesener Wert
int outputValue = 0;     // PWM-Ausgabewert

void setup()
{
 // Hier ist nichts zu tun
}

void loop()
{
 sensorValue = analogRead(analogInPin);       // Analogwert einlesen
 outputValue = map(sensorValue, 0, 1023, 0, 255);  // Für analoge Ausgabe skalieren
 analogWrite(analogMeterPin, outputValue);      // Wert an analogen Ausgang schreiben
}
```

Diskussion

Bei dieser Variante von Rezept 7.2 steuert der Arduino mit `analogWrite` eine Analoganzeige. Solche Anzeigen sind üblicherweise viel empfindlicher als LEDs . Ein Widerstand muss zwischen den Arduino-Ausgang und die Anzeige geschaltet werden, um den Strom entsprechend zu begrenzen.

Der Wert des Vorwiderstands hängt von der Empfindlichkeit der Anzeige ab. 5K-Ohm sorgen bei einer 1 mA-Anzeige für einen Vollausschlag. Sie können einen 4,7K-Widerstand verwenden, da sie leichter zu beschaffen sind als 5K, allerdings müssen Sie dann den Maximalwert für `analogWrite` auf etwa 240 beschränken. Nachfolgend sehen Sie, wie Sie den Wertebereich der `map`-Funktion anpassen müssen, wenn Sie einen 4,7K-Widerstand für eine 1 mA-Anzeige verwenden:

```
outputValue = map(sensorValue, 0, 1023, 0, 240); // Auf Anzeigebereich abbilden
```

Arbeitet Ihre Anzeige nicht mit 1 mA, müssen Sie einen anderen Vorwiderstand verwenden. Die Formel für den Widerstand in Ohm lautet

Widerstand = 5000 / mA

Bei einer 500 Mikroampere-Anzeige (0,5 mA) ist das also 5000 / 0,5, d.h. 10000 (10 K) Ohm. Eine 10 mA-Anzeige benötigt 500 Ohm, bei 20 mA 250 Ohm.

Bei einigen Anzeigen sind bereits interne Vorwiderstände integriert – Sie müssen möglicherweise experimentieren, um den korrekten Wert des Vorwiderstands zu ermitteln, achten Sie aber darauf, die Anzeige nicht mit zu viel Strom zu versorgen.

Siehe auch

Rezept 7.2

KAPITEL 8

Physische Ausgabe

8.0 Einführung

Sie können Dinge bewegen, indem Sie Motoren mit dem Arduino steuern. Verschiedene Arten von Motoren sind für unterschiedliche Anwendungen geeignet, und dieses Kapitel zeigt Ihnen, wie der Arduino die unterschiedlichen Motoren ansteuern kann.

Bewegungssteuerung mit Servomotoren

Servomotoren ermöglichen die exakte Steuerung physischer Bewegungen, da sie sich nicht kontinuierlich drehen, sondern sich zu einer bestimmten Position bewegen. Sie sind ideal, wenn sich etwas in einem Bereich von 0 bis 180 Grad bewegen soll. Servos lassen sich einfach anschließen und steuern, da die Motorsteuerung in die Servos integriert ist.

Servos enthalten einen kleinen Motor, der über Zahnräder mit der Ausgangswelle verbunden ist. Die Ausgangswelle steuert einen Servoarm an und ist außerdem mit einem Potentiometer verbunden, der Positionsdaten an den internen Steuerungskreis zurückliefert (siehe Abbildung 8-1).

Sie können auch 360°-Servos kaufen, bei denen das Positions-Feedback abgeschaltet ist. Diese Servos können kontinuierlich im oder gegen den Uhrzeigersinn laufen, und Sie haben eine gewisse Kontrolle über die Drehgeschwindigkeit. Sie ähneln in der Funktion den Bürstenmotoren, die in Rezept 8.9 behandelt werden, nutzen aber die Servo-Bibliothek anstelle von `analogWrite` und benötigen kein Motor-Shield.

Dauerrotierende Servos lassen sich einfach nutzen, da sie kein Motor-Shield benötigen – die Motorsteuerung sitzt in den Servos. Die Nachteile sind, dass die Auswahlmöglichkeiten im Bezug auf Geschwindigkeit und Leistung im Vergleich zu externen Motoren beschränkt sind und dass die Geschwindigkeitssteuerung nicht so gut ist wie bei einem Motor-Shield (die Elektronik ist auf eine genaue Positionierung ausgelegt, nicht auf eine lineare Geschwindigkeitsregelung). In Rezept 8.3 erfahren Sie mehr zum Einsatz dauerrotierender Servos.

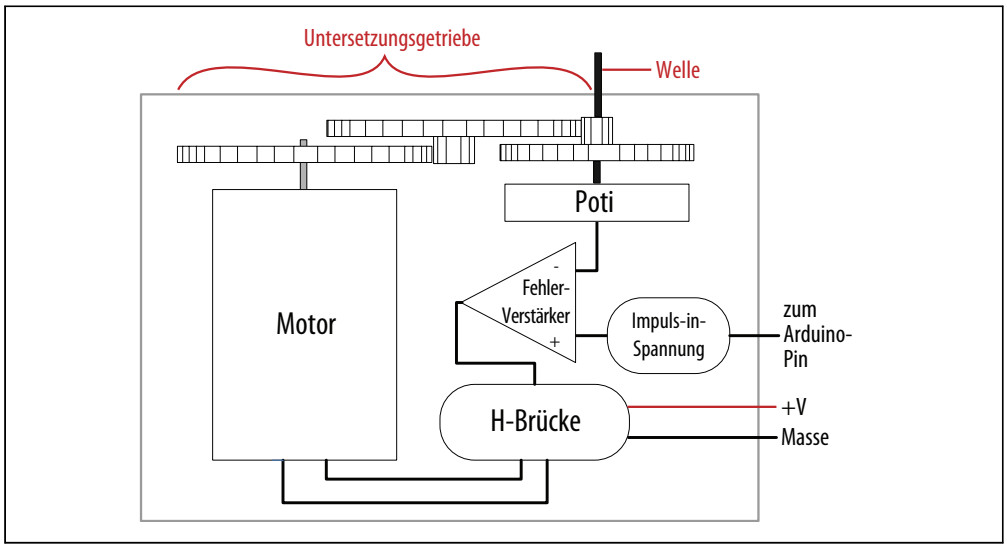


Abbildung 8-1: Bestandteile eines Servomotors

Servos reagieren auf Änderungen der Impulsdauer. Ein kurzer Impuls von einer 1 ms oder weniger lässt den Servo zu einem Extrem rotieren, eine Impulsdauer von um die 2 ms zum anderen Extrem (siehe Abbildung 8-2). Impulse zwischen diesen beiden Extremen bewegen den Servo zu einer Position proportional zur Impulsbreite. Es gibt keinen Standard für die Beziehung zwischen Impuls und Position, d.h., Sie müssen möglicherweise ein wenig mit den Befehlen im Sketch experimentieren, um den Bereich Ihrer Servos anzupassen.

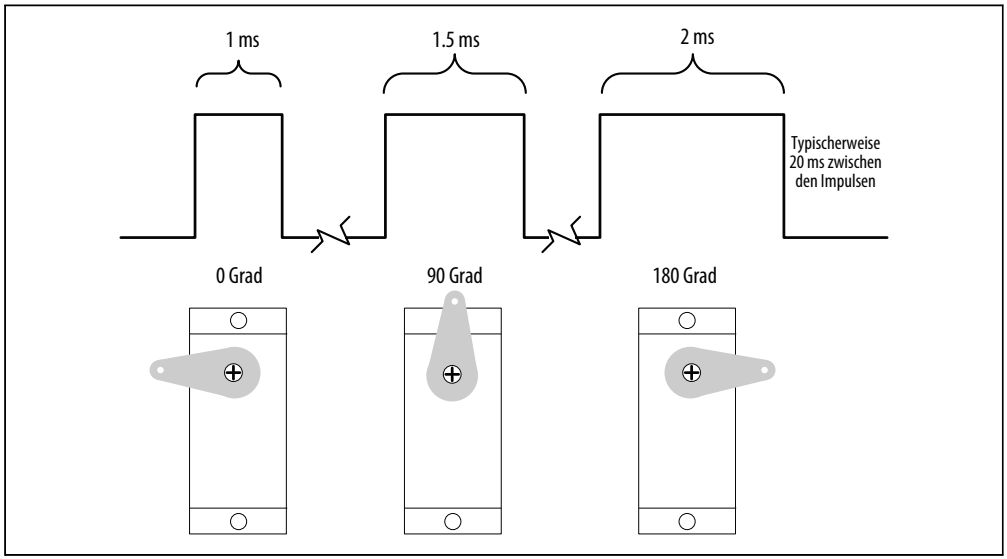
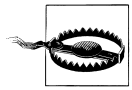


Abbildung 8-2: Beziehung zwischen Impulsbreite und Servo-Winkel; der Servo-Arm bewegt sich zwischen 1 ms und 2 ms proportional zur Impulsbreite



Zwar ist die Dauer des Impulses *moduliert*, doch Servos benötigen andere Impulse, als sie die Pulsweitenmodulation (PWM) von `analogWrite` liefert. Sie können den Servo beschädigen, wenn Sie ihn mit einem `analogWrite`-Ausgang verbinden. Arbeiten Sie stattdessen mit der Servo-Bibliothek.

Hubmagnete und Relais

Während die meisten Motoren eine Drehbewegung erzeugen, sorgt ein Hubmagnet für eine lineare Bewegung. Ein Hubmagnet besteht aus einem Metallkern, der durch ein Magnetfeld bewegt wird, wenn Strom durch eine Spule fließt. Ein mechanisches Relais ist eine Art Hubmagnet, der elektrische Kontakte schließt oder trennt (d.h. ein Hubmagnet, der einen Schalter steuert). Relais werden wie Hubmagnete gesteuert. Relais und Hubmagnete benötigen (wie die meisten Motoren) mehr Strom, als ein Arduino-Pin liefern kann. Die Rezepte zeigen, wie man einen Transistor oder eine externe Schaltung nutzt, um diese Geräte anzusteuern.

Bürsten- und bürstenlose Motoren

Die meisten billigen Gleichstrommotoren sind einfache Einheiten mit zwei Anschlüssen, die mit Bürsten (Kontakten) verbunden sind, die das Magnetfeld der Spulen steuern, die einen Magnetkern (Anker) antreiben. Die Drehrichtung kann man umkehren, indem man die Polarität an den Kontakten vertauscht. Gleichstrommotoren gibt es in vielen verschiedenen Größen, doch selbst die kleinsten (wie die in Mobiltelefonen verwendeten Vibrationsmotoren) benötigen einen Transistor oder eine andere externe Steuereinheit, um den benötigten Strom bereitzustellen. Die folgenden Rezepte zeigen, wie man Motoren über einen Transistor oder eine externe Steuerungsschaltung (eine sog. H-Brücke) ansteuert.

Das wesentliche Merkmal bei der Wahl eines Motors ist das *Drehmoment*. Das Drehmoment gibt an, wie viel Arbeit ein Motor leisten kann. Üblicherweise sind Motoren mit höherem Drehmoment größer und schwerer und ziehen mehr Strom als Motoren mit kleinerem Drehmoment.

Bürstenlose Motoren sind bei gleicher Größe üblicherweise leistungsfähiger und effektiver als Bürstenmotoren, benötigen aber eine aufwendigere Steuerungselektronik. Wo der Leistungsvorteil bürstenloser Motoren wünschenswert ist, können *elektronische Geschwindigkeitsregelungen* aus dem Fernsteuerungsbereich sehr einfach vom Arduino angesteuert werden, da man sie ansteuert wie Servomotoren.

Schrittmotoren

Schrittmotoren bewegen sich bei Steuerimpulsen um einen gewissen Winkel. Die Größe des Winkels bei jedem Schritt ist motorabhängig und reicht von ein oder zwei Grad pro Schritt bis zu 30 Grad und mehr.

Üblicherweise werden zwei Arten von Schrittmotoren mit dem Arduino genutzt: bipolar (üblicherweise vier Anschlüsse an zwei Spulen) und unipolar (fünf oder sechs Anschlüsse an zwei Spulen). Die zusätzlichen Anschlüsse eines unipolaren Schrittmotors sind intern mit der Mitte der Spule verbunden (bei fünf Anschlüssen besitzt jede Spule einen Anschluss in der Mitte, die beide miteinander verbunden sind). Die bipolare und unipolare Schrittmotoren behandelnden Rezepte beinhalten entsprechende Anschlussdiagramme.

Fehlersuche

Die häufigste Ursache für Probleme beim Anschluss von Bauelementen, die eine externe Stromversorgung benötigen, sind fehlende Masseanschlüsse. Die Masse des Arduino muss mit der Masse der externen Stromversorgung sowie mit allen externen Bauelementen verbunden sein.

8.1 Die Position eines Servos kontrollieren

Problem

Sie wollen einen Servo auf einen Winkel einstellen, der im Sketch berechnet wurde. Beispielsweise soll sich der Sensor eines Roboters vor und zurück oder zu einer von Ihnen gewählten Position bewegen.

Lösung

Nutzen Sie die mit dem Arduino mitgelieferte Servo-Bibliothek. Schließen Sie die Strom- und Masseleitung des Servos an eine geeignete Stromquelle an (ein einzelner Servo kann normalereise über die 5V-Leitung des Arduino versorgt werden). Neue Versionen der Bibliothek erlauben es Ihnen, die Signalleitungen des Servos an jeden Digitalpin des Arduino anzuschließen.

Hier der Sweep-Sketch, der mit dem Arduino mitgeliefert wird. In Abbildung 8-3 sehen Sie die Verschaltung:

```
#include <Servo.h>

Servo myservo;  // Servo-Objekt zur Steuerung des Servos erzeugen

int angle = 0;   // Diese Variable enthält die Position des Servos

void setup()
{
 myservo.attach(9); // Verbindet den Servo an Pin 9 mit dem Servo-Objekt
}


void loop()
{
 for(angle = 0; angle < 180; angle += 1)  // In 1er-Schritten
 {                                        // von 0 bis 180 Grad
```

```
  myservo.write(angle);  // Servo anweisen, die Position in der Variablen 'angle' einzunehmen
  delay(20);          // Zwischen den Servo-Befehlen 20ms warten
 }
 for(angle = 180; angle >= 1; angle -= 1) // Von 180 bis 0 Grad
 {
  myservo.write(angle);  // Servo in Gegenrichtung bewegen
  delay(20);          // Zwischen den Servo-Befehlen 20ms warten
 }
}
```

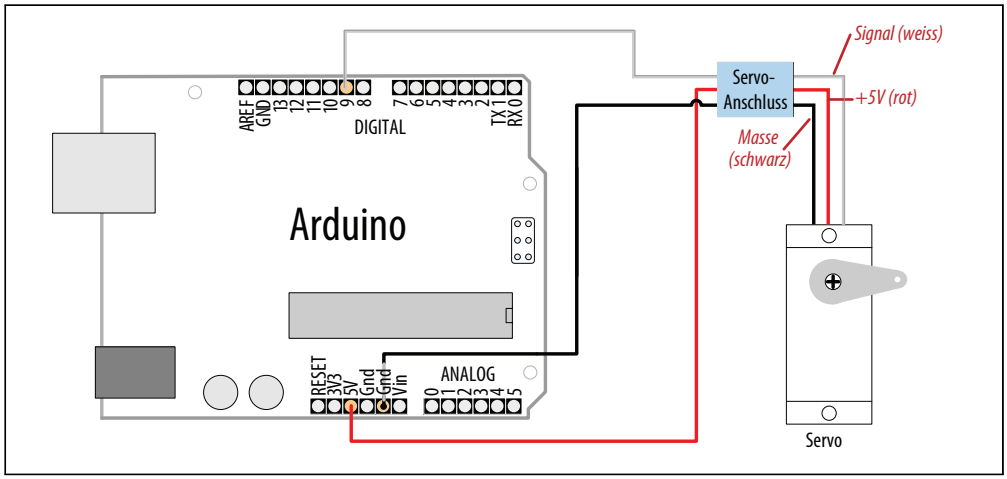


Abbildung 8-3: Anschluss eines Servos zum Testen mit dem Sweep Beispiel-Sketch

Diskussion

Dieses Beispiel bewegt den Servo zwischen 0 und 180 Grad hin und her. Möglicherweise müssen Sie die Minimal- und Maximalpositionen anpassen, um den gewünschten Bewegungsbereich einzustellen. Der Aufruf von `Servo.attach` mit optionalen Argumenten für die Minimal- und Maximalposition passen die Bewegung an:

```
myservo.attach(9,1000,2000 ); // Pin 9, min ist 1000us, max ist 2000us
```

Da typische Servos auf Impulse reagieren, die in Mikrosekunden gemessen werden (und nicht in Grad), teilen die auf die Pin-Nummer folgenden Argumente der Servo-Bibliothek mit, wie viele Mikrosekunden für 0 bzw. 180 Grad benötigt werden. Nicht alle Servos bewegen sich über die gesamten 180 Grad, weshalb Sie mit Ihren Servos möglicherweise ein wenig experimentieren müssen, um den gewünschten Bereich zu abzudecken.

Die Parameter für `servo.attach(`*`pin`*`, `*`min`*`, `*`max`*`)` sind wie folgt:

`pin`
: Die Nummer des Pins, an den der Servo angeschlossen ist (Sie können einen beliebigen Digitalpin wählen)

min (optional)
: Die Impulsbreite in Mikrosekunden für den Minimalwinkel (0 Grad) des Servos (voreingestellt ist 544)

max (optional)
: Die Impulsbreite in Mikrosekunde für den Maximalwinkel (180 Grad) des Servos (voreingestellt ist 2400)

Die Servo-Bibliothek unterstützt bei den meisten Arduino-Boards bis zu 12 Servos und beim Arduino Mega sogar bis zu 48. Bei Standard-Boards wie dem Uno deaktiviert die Bibliothek die `analogWrite()`-Funktionalität (PWM) an den Pins 9 und 10, und zwar unabhängig davon, ob ein Servo angeschlossen ist oder nicht. Weitere Informationen finden Sie in der Referenz zur Servo-Bibliothek: *http://arduino.cc/en/Reference/Servo*.

Die Energiebedarf hängt vom Servo ab und davon, wie viel Drehmoment benötigt wird, um die Welle zu drehen.

Sie benötigen eventuell eine externe Stromquelle von 5 oder 6 Volt, wenn Sie mehrere Servos anschließen wollen. Vier AA-Zellen funktionieren gut, wenn Sie mit Batterien arbeiten wollen. Denken Sie daran, dass die Masse der externen Stromversorgung mit der Masse des Arduino verbunden sein muss.

8.2 Ein oder zwei Servos mit einem Potentiometer oder Sensor steuern

Problem

Sie wollen die Drehrichtung und die Geschwindigkeit von ein oder zwei Servos mit einem Potentiometer steuern. Beispielsweise könnten Sie eine mit den Servos verbundene Kamera oder einen Sensor schwenken und neigen. Das Rezept kann mit einer variablen Spannung von einem Sensor arbeiten, der über einen analogen Eingang eingelesen wurde.

Lösung

Sie können die gleiche Bibliothek wie in Rezept 8.1 verwenden und um Code ergänzen, der die Spannung vom Potentiometer einliest. Dieser Wert wird so skaliert, dass die Position des Potentiometers (`0` bis `1023`) auf einen Winkel von 0 bis 180 Grad abgebildet wird. Der einzige Unterschied in der Verschaltung ist der zusätzliche Potentiometer; siehe Abbildung 8-4:

```
#include <Servo.h>

Servo myservo; // Servo-Objekt zur Steuerung des Servos erzeugen

int potpin = 0; // Analogpin des Potentiometers
```

```
int val;   // Variable für den Wert des Analogpins

void setup()
{
 myservo.attach(9); // Servo an Pin 9 mit dem Servo-Objekt verbinden
}

void loop()
{
 val = analogRead(potpin);       // Wert des Potentiometers einlesen
 val = map(val, 0, 1023, 0, 180);   // Wert für Servo skalieren
 myservo.write(val);           // Position setzen
 delay(15);                 // Auf Servo warten
}
```

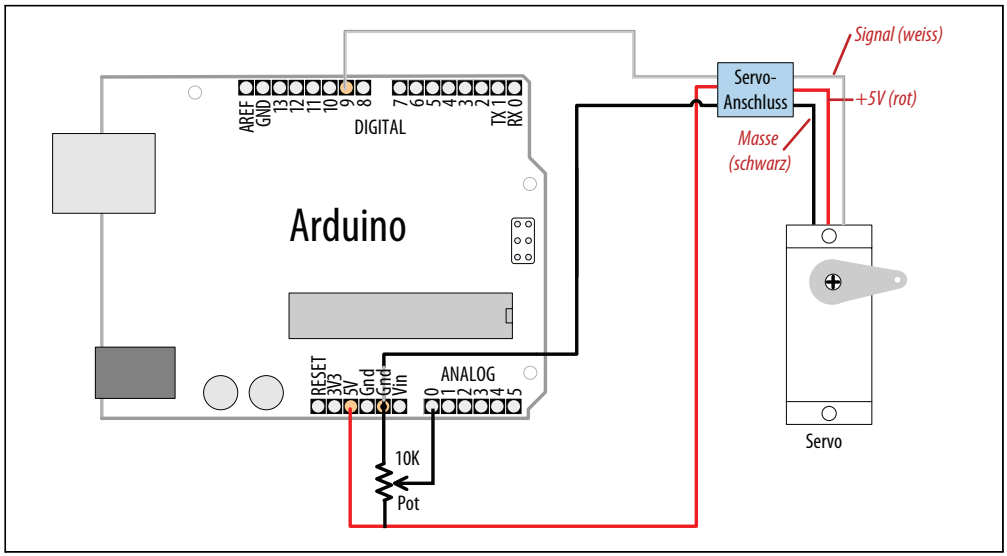


Abbildung 8-4: Servo mit Potentiometer steuern

Hobby-Servos besitzen ein Kabel mit einem weiblichen 3-Pin-Stecker, der direkt in den »Servo«-Anschluss einiger Shields (wie dem Adafruit Motor Shield) eingesteckt werden kann. Der physische Anschluss ist kompatibel mit den Arduino-Anschlüssen, d.h., Sie können die gleichen Drahtbrücken verwenden wie für die Arduino-Pins. Denken Sie daran, dass die Farben der Signalanschlüsse nicht standardisiert sind. Manchmal wird Gelb anstelle von, Weiß verwendet. Rot befindet sich immer in der Mitte und der Masse-Anschluss ist üblicherweise schwarz oder braun.

Diskussion

Sie können alles verwenden, was über `analogRead` (siehe Kapitel 5 und Kapitel 6) eingelesen werden kann. So können etwa die Gyroskop- und Beschleunigungsmesser-Rezepte

aus Kapitel 6 genutzt werden, um den Winkel des Servos über die Gierung des Gyroskops oder den Winkel des Beschleunigungsmessers festzulegen.

Nicht alle Servos können sich über den gesamten Wertebereich der Servo-Bibliothek bewegen. Wenn Ihr Servo brummt, weil er vorzeitig ein Ende erreicht, sollten Sie den Ausgabebereich der map-Funktion so lange reduzieren, bis das Brummen aufhört. Ein Beispiel:

```
val=map(val,0,1023,10,170); // Die meisten Servos funktionieren in diesem
Bereich
```

8.3 Die Geschwindigkeit dauerrotierender Servos steuern

Problem

Sie wollen die Drehrichtung und Geschwindigkeit dauerrotierender Servos steuern. Beispielsweise könnten Sie einen Roboter über zwei solche Servos antreiben und die Geschwindigkeit und Richtung über Ihren Sketch regeln wollen.

Lösung

Dauerrotierende Servos sind eine Art Getriebemotor mit Vorwärts/Rückwärts-Geschwindigkeitsregelung. Die Steuerung dauerrotierender Servos ähnelt derjenigen normaler Servos. Der Servo dreht sich in eine Richtung, während der Winkel über 90 Grad hinaus erhöht wird, und in die andere Richtung, wenn er unter 90 Grad sinkt. Wann es vorwärts oder rückwärts geht, hängt von der Verschaltung der Servos ab. Abbildung 8-5 zeigt die Verschaltung zur Steuerung zweier Servos.

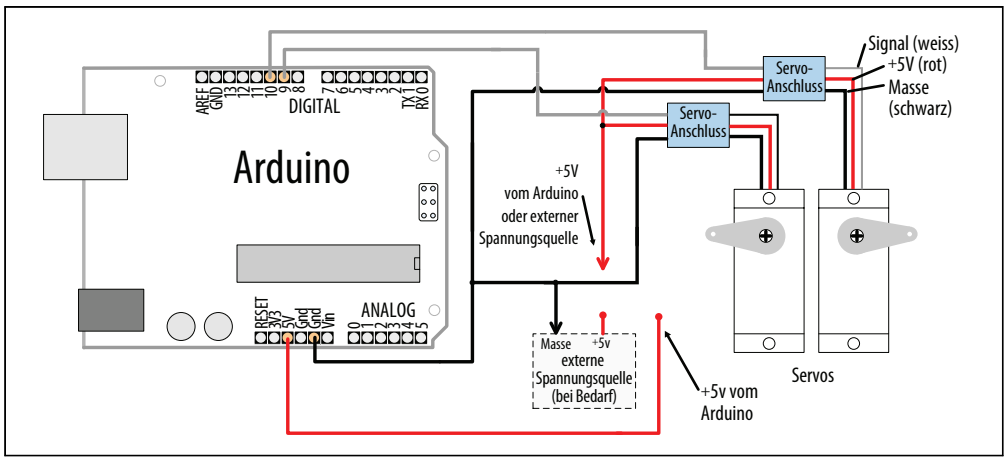


Abbildung 8-5: Steuerung zweier Servos

Servos werden üblicherweise über eine 4,8V bis 6V-Quelle gespeist. Größere Servos benötigen möglicherweise mehr Strom, als das Arduino-Board über seinen +5V-Pin liefern kann, und benötigen eine externe Stromversorgung. Vier 1,2V-Akkus können genutzt werden, um den Arduino und die Servos anzutreiben. Denken Sie daran, dass frische Alkali-Batterien eine Spannung über 1,5V liefern können. Überprüfen Sie mit Ihrem Multimeter, ob die Gesamtspannung die 6 Volt auch nicht überschreitet – das absolute Maximum für Arduino-Chips.

Der Sketch lässt die Servos zwischen 90 und 180 Grad hin und her laufen. Wären die Servos mit Rädern verbunden, würde sich das Fahrzeug immer schneller nach vorne bewegen und dann wieder langsamer werden. Da der Code zur Servo-Steuerung in `loop` liegt, würde sich das wiederholen, bis kein Saft mehr da ist:

```
#include <Servo.h>

Servo myservoLeft;  // Servo-Objekt zur Steuerung des ersten Servos
Servo myservoRight; // Servo-Objekt zur Steuerung des zweite Servos

int angle = 0;   // Variable enthält die Position der Servos

void setup()
{
 myservoLeft.attach(9);  // Linker Servo an Pin 9
 myservoRight.attach(10); // Rechter Servo an Pin 10
}

void loop()
{
 for(angle = 90; angle < 180; angle += 1)  // In Einerschritten von
 {                         // 90 bis 180 Grad
                          // Halt bei 90  Grad

  myservoLeft.write(angle);     // Servo mit Geschwindigkeit in 'angle' drehen
  myservoRight.write(180-angle);  // Zweiter Servo in Gegenrichtung

  delay(20);              // 20ms warten zwischen Servo-Befehlen
 }
 for(angle = 180; angle >= 90; angle -= 1) // Von 180 bis 90 Grad
 {
  myservoLeft.write(angle);     // Servo mit Geschwindigkeit in 'angle' drehen
  myservoRight.write(180-angle);  // Zweiter Servo in Gegenrichtung
 }
}
```

Diskussion

Sie können für dauerrotierende und normale Servos ähnlichen Code nutzen, müssen aber daran denken, dass dauerrotierende Servos nicht unbedingt bei genau 90 Grad aufhören zu rotieren. Einige Servos besitzen kleine Potentiometer, mit denen man das justieren kann. Alternativ können Sie ein paar Grad hinzufügen oder abziehen, um den Servo

anzuhalten. Hält der linke Servo beispielsweise bei 92 Grad an, passen Sie die entsprechende Servo-Steuerungszeile im Code wie folgt an:

```
myservoLeft.write(angle+TRIM); // int TRIM=2; zu beginn des Sketches deklarieren
```

8.4 Servos über Computerbefehle steuern

Problem

Sie wollen Befehle zur Verfügung stellen, mit denen sich Servos über den seriellen Port steuern lassen. Beispielsweise könnten Sie die Servos über ein Programm steuern wollen, das auf Ihrem Computer läuft.

Lösung

Sie können Software nutzen, um die Servos zu steuern. Das hat den Vorteil, dass eine beliebige Zahl von Servos unterstützt werden kann. Allerdings muss der Sketch die Servo-Position fortlaufend aktualisieren, d.h., die Logik wird komplizierter, wenn sich die Anzahl der Servos erhöht und viele andere Aufgaben erledigt werden müssen.

Das folgende Rezept steuert vier Servos über Befehle an, die über den seriellen Port eingehen. Die Befehle haben die folgende Form:

- `180a` schreibt 180 an Servo a
- `90b` schreibt 90 an Servo b
- `0c` schreibt 0 an Servo c
- `17d` schreibt 17 an Servo d

Nachfolgend der Sketch, der vier Servos an den Pins 7 bis 10 ansteuert:

```
#include <Servo.h> // Servo-Bibliothek

#define SERVOS 4 // Anzahl der Servos
int servoPins[SERVOS] = {7,8,9,10}; // Servos an Pins 7 bis 10

Servo myservo[SERVOS];

void setup()
{
 Serial.begin(9600);
 for(int i=0; i < SERVOS; i++)
  myservo[i].attach(servoPins[i]);
}

void loop()
{
 serviceSerial();
}

// serviceSerial überwacht den seriellen Port und aktualisiert die Position entsprechend der
empfangenen Daten
```

```
// Die Servo-Daten werden im folgenden Format erwartet:
//
// "180a" schreibt 180 an Servo a
// "90b schreibt 90 an Servo b
//
void serviceSerial()
{
 static int pos = 0;

 if ( Serial.available()) {
  char ch = Serial.read();

  if( isDigit(ch) )             // Wenn ch eine Ziffer ist:
   pos = pos * 10 + ch - '0';       // Wert akkumulieren
  else if(ch >= 'a' && ch <= 'a'+ SERVOS) // Wenn ch ein Buchstabe für die Servos ist:
   myservo[ch - 'a'].write(pos);     // Servo positionieren
 }
}
```

Diskussion

Der Anschluss der Servos entspricht dem vorigen Rezept. Jeder Servo-Anschluss wird mit einem Digitalpin verbunden. Alle Servo-Masseanschlüsse werden mit der Arduino-Masse verbunden. Die Servo-Stromanschlüsse werden miteinander verbunden. Möglicherweise müssen Sie eine externe 5V- oder 6V-Spannungsquelle nutzen, wenn die Servos mehr Strom ziehen, als die Arduino-Stromversorgung liefern kann.

Ein Array namens `myservo` (siehe Rezept 2.4) wird zur Referenzierung der vier Servos verwendet. Eine `for`-Schleife in `setup` verknüpft jeden Servo im Array mit den im `servoPins`-Array definierten Pins.

Ist das über den seriellen Port empfangene Zeichen eine Ziffer (ein Zeichen größer oder gleich 0 und kleiner oder gleich 9), wird deren Wert in der Variablen `pos` aufsummiert. Ist das Zeichen der Buchstabe *a*, wird die Position an den ersten Servo im Array (den Servo an Pin 7) geschrieben. Die Buchstaben *b*, *c* und *d* steuern die anderen Servos.

Siehe auch

Wie man über den seriellen Port empfangene Daten verarbeitet, erläutert Kapitel 4.

8.5 Einen bürstenlosen Motor (per Fahrtregler) steuern

Problem

Sie wollen die Geschwindigkeit eines bürstenlosen Motors regeln.

Lösung

Der Sketch verwendet den gleichen Code wie in Rezept 8.2. Bis auf den Fahrtregler und den Motor ist der Aufbau identisch. Ein elektronischer Fahrtregler ist ein Gerät, mit dem

bürstenlose Motoren in ferngesteuerten Fahrzeugen kontrolliert werden. Da sie in Massen gefertigt werden, sind sie eine kosteneffektive Möglichkeit zur Steuerung bürstenloser Motoren. Sie finden eine Auswahl, wenn Sie bei Ihrem bevorzugten Elektronik-Händler oder bei Google »Fahrtregler« eingeben.

Bürstenlose Motoren haben drei Spulen, die entsprechend der Dokumentation mit den Fahrtregler verbunden werden müssen (siehe Abbildung 8-6).

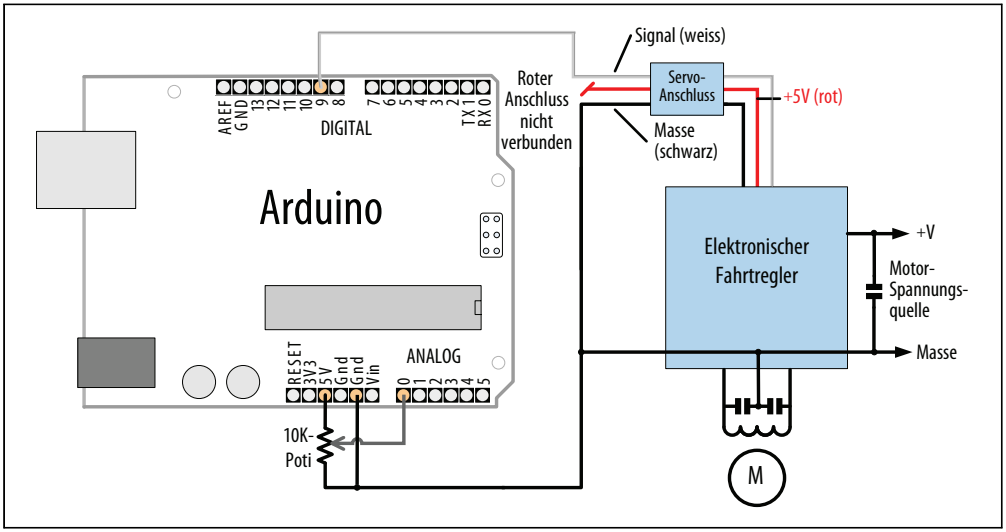
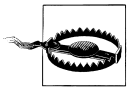


Abbildung 8-6: Anschluss eines Fahrtreglers

Diskussion

Stellen Sie anhand der Dokumentation sicher, dass Ihr Fahrtregler für Ihren bürstenlosen Motor geeignet ist, und überprüfen Sie die Verschaltung. Bürstenlose Motoren haben drei Anschlüsse für die einzelnen Spulen und zwei Anschlüsse für die Stromversorgung. Viele Fahrtregler stellen Strom am mittleren Pin des Servo-Anschlusses bereit. Wenn Sie das Arduino-Board nicht über den Fahrtregler versorgen wollen, müssen Sie den Anschluss abklemmen oder abschneiden.



Wenn Ihr Fahrtregler 5V für die Servos und andere Einheiten bereitstellt (man nennt das *Battery Eliminator Circuit*, kurz *BEC*), müssen Sie dieses Kabel abklemmen, wenn Sie den Arduino mit dem Fahrtregler verbinden (siehe Abbildung 8-6).

8.6 Hubmagnete und Relais steuern

Problem

Sie wollen einen Hubmagneten oder ein Relais aus einem Programm heraus steuern. Hubmagneten sind Elektromagnete, die elektrische Energie in mechanische Bewegung umwandeln. Ein elektromagnetisches Relais ist ein Schalter, der durch einen Hubmagneten aktiviert wird.

Lösung

Die meisten Hubmagnete benötigen mehr Strom, als ein Arduino-Pin liefern kann, weshalb ein Transistor verwendet wird, um den Strom zu schalten, der zur Aktivierung des Hubmagneten benötigt wird. Sie aktivieren den Hubmagneten, indem Sie mit `digitalWrite` den entsprechenden Pin auf `HIGH` setzen.

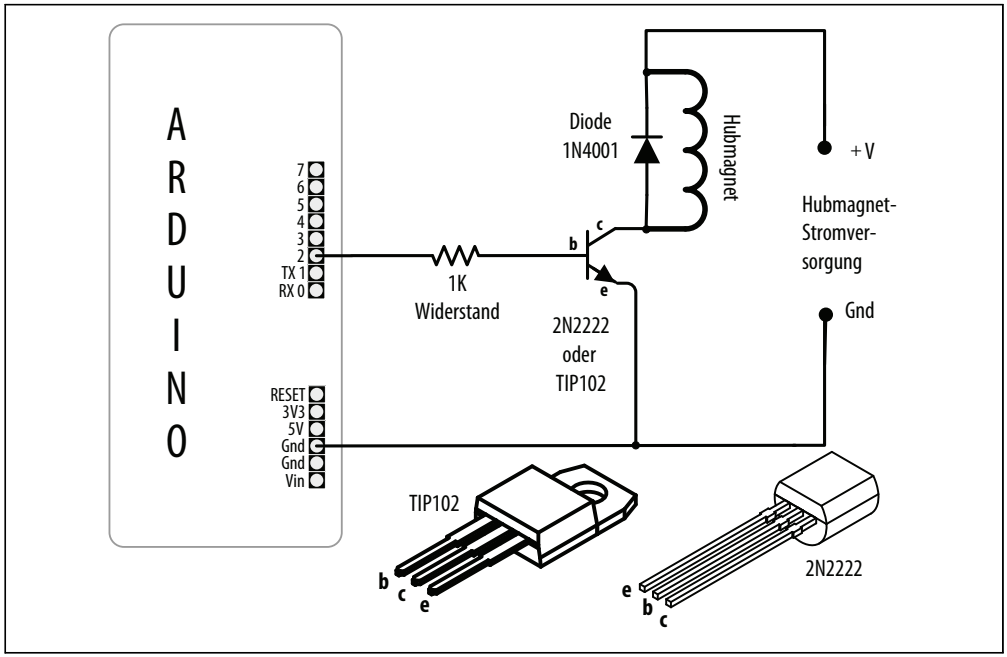


Abbildung 8-7: Hubmagneten mit einem Transistor ansteuern

Der Sketch schaltet einen Transistor, der wie in Abbildung 8-7 angeschlossen ist. Der Hubmagnet wird einmal pro Stunde für eine Sekunde aktiviert:

```
int solenoidPin = 2;        // Hubmagnet an Pin 2

void setup()
{
 pinMode(solenoidPin, OUTPUT);
}
```

```
void loop()
{
 long interval = 1000 * 60 * 60 ;  // Interval = 60 Minuten

 digitalWrite(solenoidPin, HIGH); // Aktiviert den Hubmagneten
 delay(1000);            // Wartet eine Sekunde
 digitalWrite(solenoidPin, LOW);  // Deaktiviert den Hubmagneten
 delay(interval);          // Wartet eine Stunde
}
```

Diskussion

Die Wahl des Transistors hängt von der Strommenge ab, die zur Aktivierung des Hubmagneten oder Relais benötigt wird. Das Datenblatt kann das in Milliampere (mA) angeben oder in Form des Widerstands der Spule. Um den vom Hubmagneten oder Relais benötigten Strom zu berechnen, teilen Sie die Spannung an der Spule durch ihren Widerstand (in Ohm). Ein 12V-Relais mit einem Spulenwiderstand von 185 Ohm benötigt 65 mA: 12 (Volt) / 185 (Ohm) = 0,065 Ampere, also 65 mA.

Kleine Transistoren wie der 2N2222 reichen für Hubmagnete, die mehrere hundert Milliampere benötigen. Größere Hubmagnete benötigen leistungsfähigere Transistoren wie den TIP102/TIP120. Es gibt eine Vielzahl geeigneter Transistoren. Anhang B zeigt, wie man Datenblätter liest und Transistoren auswählt.

Die Aufgabe der Freilaufdiode besteht darin, zu verhindern, dass die Gegen-EMK (elektromagnetische Kraft) der Spule den Transistor beschädigen kann (die *Gegen-EMK* ist eine Spannung, die erzeugt wird, wenn der Strom durch eine Spule abgeschaltet wird). Die Polarität der Diode ist dabei wichtig. Eine farbige Markierung kennzeichnet die Kathode – sie muss mit dem Pluspol des Hubmagneten verbunden sein.

Elektromagnetische Relais verhalten sich genau wie Hubmagnete. Ein spezielles Relais, das sog. *Solid-State-* (SSR) oder Halbleiterrelais verfügt über eine interne Elektronik, die ohne Transistor direkt über einen Arduino-Pin angesteuert werden kann. Auf dem Datenblatt zu Ihrem Relais steht, welche Spannung und welchen Strom es benötigt. Alles über 40 mA bei 5 Volt verlangt eine Schaltung wie in Abbildung 8-7.

8.7 Ein Objekt vibrieren lassen

Problem

Ihr Arduino soll etwas vibrieren lassen. Zum Beispiel soll Ihr Projekt jede Minute eine Sekunde lang wackeln.

Lösung

Schließen Sie einen Vibrationsmotor wie in Abbildung 8-8 an.

Der folgende Sketch schaltet den Vibrationsmotor jede Minute für eine Sekunde an:

```
/*
 * Vibrate Sketch
 * Vibriert jede Minute für eine Sekunde
 *
 */

const int motorPin = 3; // Transistor für Vibrationsmotor an Pin 3

void setup()
{
 pinMode(motorPin, OUTPUT);
}

void loop()
{
 digitalWrite(motorPin, HIGH); // Vibration einschalten
 delay(1000); // Eine Sekunde warten
 digitalWrite(motorPin, LOW);  // Vibration ausschalten
 delay(59000); // 59 Sekunden warten
}
```

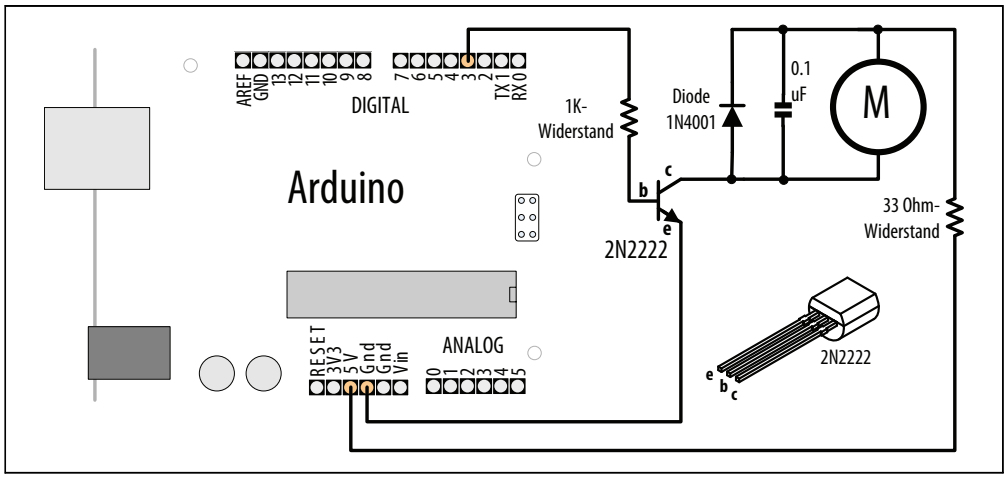


Abbildung 8-8: Anschluss eines Vibrationsmotors

Diskussion

Das Rezept nutzt einen speziellen Vibrationsmotor wie den SparkFun ROB-08449. Wenn Sie ein altes Mobiltelefon besitzen, das Sie nicht mehr benötigen, könnte es kleine geeignete Vibrationsmotoren enthalten. Vibrationsmotoren benötigen eine höhere Leistung, als ein Arduino-Pin liefern kann, weshalb ein Transistor genutzt wird, um den Motor ein- und auszuschalten. Nahezu jeder NPN-Transistor kann verwendet werden. Abbildung 8-3 zeigt den gängigen 2N2222. Auf der Website zu diesem Buch (*http://shop.oreilly.com/product/0636920022244.do*) finden Sie Anbieterinformationen zu diesen und anderen verwendeten Komponenten. Ein 1K-Widerstand verbindet den Ausgangspin mit der Basis des Tran-

sistors. Der Widerstandswert ist unkritisch und kann bis zu 4,7K betragen (er verhindert, dass zu viel Strom durch den Ausgangspin fließt). Die Diode absorbiert (oder *entkoppelt* – man nennt sie manchmal auch *Freilaufdiode*) die Spannungen, die von den Motorspulen erzeugt werden, während sich der Motor dreht. Der Kondensator absorbiert die Spannungsspitzen, die erzeugt werden, wenn die *Bürsten* (die Kontakte, die den Strom mit den Windungen verbinden) öffnen und schließen. Der 33 Ohm Widerstand wird benötigt, um den Strom zu beschränken, der durch den Motor fließt.

Der Sketch setzt den Ausgangspin für eine Sekunde (1000 Millisekunden) auf `HIGH` und wartet dann 59 Sekunden. Der Transistor schaltet (leitet), wenn der Pin `HIGH` ist und lässt Strom durch den Motor fließen.

Hier eine Variante des Sketches, der einen Sensor nutzt, um den Motor vibrieren zu lassen. Die Schaltung ähnelt der aus Abbildung 8-8, nur dass hier zusätzlich noch eine Photozelle an Analogpin 0 angeschlossen ist (siehe Rezept 6.2):

```
/*
 * Vibrate_Photocell Sketch
 * Vibriert, wenn der Photosensor ein Licht erkennt, das heller ist als die Umgebung
 *
 */

const int motorPin  = 3;  // Transistor für Vibrationsmotor an Pin 3
const int sensorPin = 0;  // Photozelle an Analogeingang 0
int sensorAmbient = 0;        // Umgebungslicht (im setup kalibriert)
const int thresholdMargin = 100;  // Vibrations-Schwellwert über Umgebungslicht

void setup()
{
 pinMode(motorPin, OUTPUT);
 sensorAmbient = analogRead(sensorPin); // Zu Beginn Umgebungslicht messen;
}

void loop()
{
 int sensorValue = analogRead(sensorPin);
 if( sensorValue > sensorAmbient + thresholdMargin)
 {
   digitalWrite(motorPin, HIGH); //Vibration starten
 }
 else
 {
   digitalWrite(motorPin, LOW);  // Vibration anhalten
 }
}
```

Hier wird der Ausgangspin eingeschaltet, wenn Licht auf die Photozelle trifft. Beim Start des Sketches wird das Umgebungslicht am Sensor gemessen und in der Variablen `sensorAmbient` gespeichert. Wird in `loop` eine Helligkeit gemessen, die über diesem Wert plus einem Schwellwert liegt, wird der Vibrationsmotor eingeschaltet.

8.8 Einen Bürstenmotor über einen Transistor ansteuern

Problem

Sie wollen einen Motor ein- und ausschalten. Sie wollen seine Geschwindigkeit kontrollieren. Der Motor muss sich nur in eine Richtung drehen.

Lösung

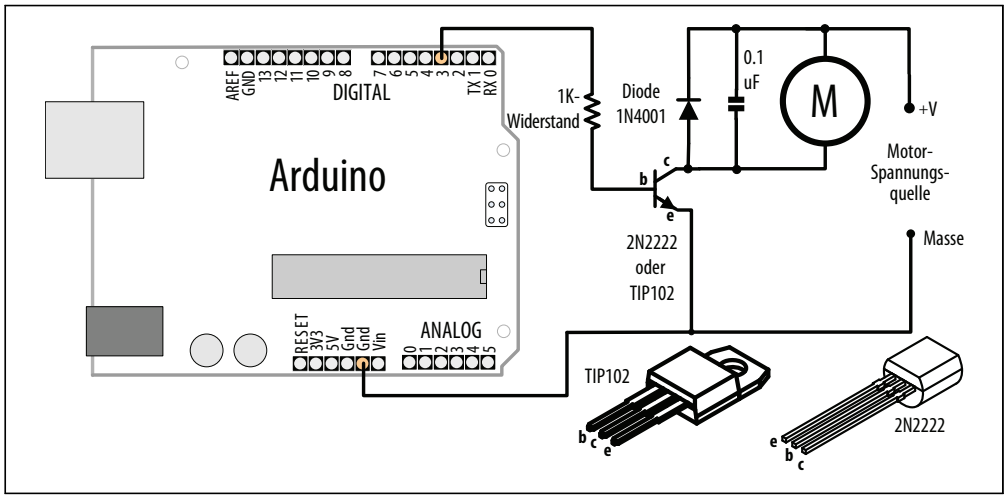


Abbildung 8-9: Anschluss eines Bürstenmotors

Der folgende Sketch schaltet den Motor ein und aus und kontrolliert dessen Geschwindigkeit über Befehle, die vom seriellen Port eingehen (Abbildung 8-9 zeigt die Verschaltung):

```
/*
 * SimpleBrushed Sketch
 * Befehle vom seriellen Port steuern die Motorgeschwindigkeit
 * Die Ziffern '0' bis '9' sind gültig. '0' bedeutet aus, '9' die max. Geschwindigkeit
 */

const int motorPin = 3; // Motortreiber an Pin 3

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

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();

  if(isDigit(ch))        // Ist ch eine Ziffer?
  {
```

```
    int speed = map(ch, '0', '9', 0, 255);
    analogWrite(motorPin, speed);
    Serial.println(speed);
   }
   else
   {
    Serial.print("Unbekanntes Zeichen ");
    Serial.println(ch);
   }
  }
}
```

Diskussion

Das Rezept ähnelt Rezept 8.7. Der Unterschied besteht darin, dass `analogWrite` genutzt wird, um die Geschwindigkeit des Motors zu regeln. In Rezept 7.1 erfahren Sie mehr über `analogWrite` und die Pulsweitenmodulation (PWM).

8.9 Die Drehrichtung eines Bürstenmotors über eine H-Brücke steuern

Problem

Sie wollen die Drehrichtung eines Bürstenmotors steuern. Zum Beispiel könnten Sie den Motor über Befehle vom seriellen Port in die eine oder in die andere Richtung drehen lassen.

Lösung

Eine H-Brücke kann zwei Bürstenmotoren steuern. Abbildung 8-10 zeigt den Anschluss des Motortreiber-ICs L293D. Sie können auch den SN754410 verwenden, der das gleiche Pin-Layout besitzt:

```
/*
 * Brushed_H_Bridge_simple Sketch
 * Befehle vom seriellen Port steuern die Drehrichtung des Motors
 * + und - legen die Drehrichtung fest, alle anderen Tasten halten den Motor an
 */

const int in1Pin = 5;  // H-Brücken-Eingangspins
const int in2Pin = 4;

void setup()
{
 Serial.begin(9600);
 pinMode(in1Pin, OUTPUT);
 pinMode(in2Pin, OUTPUT);
 Serial.println("+ - ändern die Drehrichtung, alle anderen Tasten halten den Motor an");
}
void loop()
{
```

```
 if ( Serial.available()) {
  char ch = Serial.read();
  if (ch == '+')
  {
   Serial.println("CW");
   digitalWrite(in1Pin,LOW);
   digitalWrite(in2Pin,HIGH);
  }
  else if (ch == '-')
  {
   Serial.println("CCW");
   digitalWrite(in1Pin,HIGH);
   digitalWrite(in2Pin,LOW);
  }
  else
  {
   Serial.print("Motor angehalten");
   digitalWrite(in1Pin,LOW);
   digitalWrite(in2Pin,LOW);
  }
 }
}
```

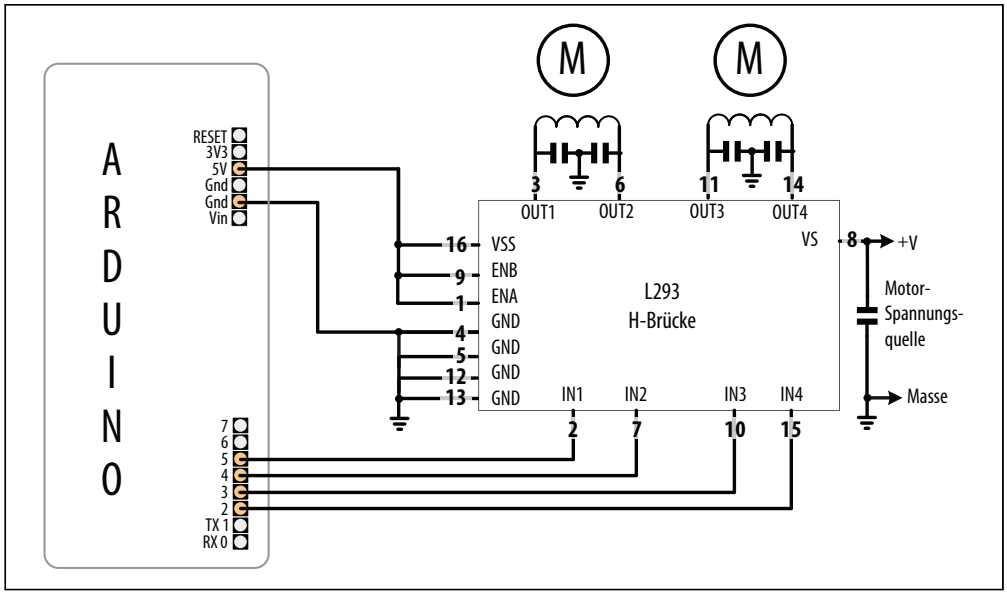


Abbildung 8-10: Anschluss zweier Bürstenmotoren mit H-Brücke L293D

Diskussion

Tabelle 8-1 zeigt, wie die Werte am H-Brücken-Eingang den Motor steuern. Im obigen Sketch wird ein einzelner Motor über die Pins IN1 und IN2 angesteuert. Der EN-Pin ist immer `HIGH`, da er direkt mit +5V verbunden ist.

Tabelle 8-1: Logiktabelle für H-Brücke

EN	IN1	IN2	Funktion?
HIGH	LOW	HIGH	Im Uhrzeigersinn drehen
HIGH	HIGH	LOW	Gegen den Uhrzeigersinn drehen
HIGH	LOW	LOW	Motor anhalten
HIGH	HIGH	HIGH	Motor anhalten
LOW	Ignoriert	Ignoriert	Motor anhalten

Abbildung 8-10 zeigt, wie man einen zweiten Motor anschließt. Der folgende Sketch steuert beide Motoren:

```
/*
 * Brushed_H_Bridge_simple2 Sketch
 * Befehle vom seriellen Port steuern die Drehrichtung des Motors
 * + und - legen die Drehrichtung fest, alle anderen Tasten halten die Motoren an
 */

const int in1Pin = 5;  // H-Brücken-Eingangspins
const int in2Pin = 4;

const int in3Pin = 3;  // H-Brücken-Pins für zweiten Motor
const int in4Pin = 2;

void setup()
{
 Serial.begin(9600);
 pinMode(in1Pin, OUTPUT);
 pinMode(in2Pin, OUTPUT);
 pinMode(in3Pin, OUTPUT);
 pinMode(in4Pin, OUTPUT);
 Serial.println("+ - ändern die Drehrichtung, alle anderen Tasten halten die Motoren an");
}

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();
  if (ch == '+')
{
   Serial.println("CW");
   // first motor
   digitalWrite(in1Pin,LOW);
   digitalWrite(in2Pin,HIGH);
   //second motor
   digitalWrite(in3Pin,LOW);
   digitalWrite(in4Pin,HIGH);
  }
  else if (ch == '-')
  {
   Serial.println("CCW");
   digitalWrite(in1Pin,HIGH);
   digitalWrite(in2Pin,LOW);
```

```
    digitalWrite(in3Pin,HIGH);
    digitalWrite(in4Pin,LOW);
   }
   else
   {
    Serial.print("Motoren angehalten");
    digitalWrite(in1Pin,LOW);
    digitalWrite(in2Pin,LOW);
    digitalWrite(in3Pin,LOW);
    digitalWrite(in4Pin,LOW);
   }
  }
 }
```

8.10 Drehrichtung und Geschwindigkeit eines Bürstenmotors mit einer H-Brücke steuern

Problem

Sie wollen die Drehrichtung und die Geschwindigkeit eines Bürstenmotors steuern. Das erweitert die Funktionalität von Rezept 8.9, indem es neben der Drehrichtung auch die Geschwindigkeit über Befehle steuert, die vom seriellen Port eingehen.

Lösung

Verbinden Sie den Bürstenmotor wie in Abbildung 8-11 zu sehen mit den Ausgangspins der H-Brücke.

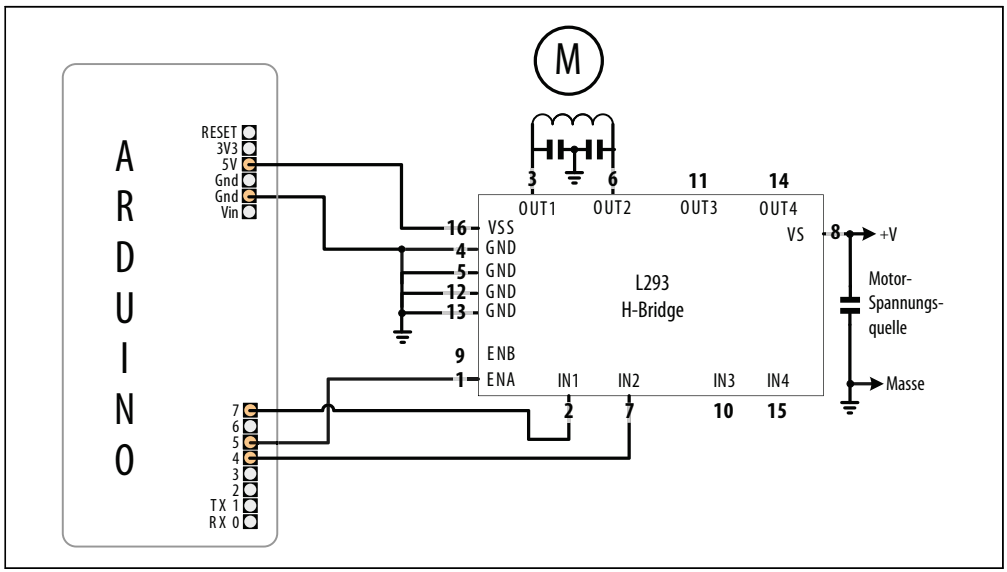


Abbildung 8-11: Anschluss eines Bürstenmotors. Nutzt analogWrite zur Geschwindigkeitsregelung

Der Sketch verarbeitet Befehle vom seriellen Monitor, um die Geschwindigkeit und die Drehrichtung des Motors zu steuern. Eine 0 hält den Motor an und die Ziffern 1 bis 9 steuern die Geschwindigkeit. Mit »+« und »-« wird die Drehrichtung des Motors festgelegt:

```
/*
 * Brushed_H_Bridge Sketch
 * Befehle vom seriellen Port steuern Drehrichtung und Geschwindigkeit des Motors
 * Die Ziffer '0' bis '9' regeln die Geschwindigkeit; '0' steht für Motor aus und '9' ist die
Maximalgeschwindigkeit
 * + und - legen die Drehrichtung fest, alle anderen Tasten halten die Motoren an

const int enPin  = 5;  // H-Brücken Enable-Pin
const int in1Pin = 7;  // H-Brücken-Eingangspins
const int in2Pin = 4;

void setup()
{
 Serial.begin(9600);
 pinMode(in1Pin, OUTPUT);
 pinMode(in2Pin, OUTPUT);
 Serial.println("Geschwindigkeit (0-9) oder + - fuer Drehrichtung");
}

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();
  if(isDigit(ch))        // Ist ch eine Ziffer?
  {
   int speed = map(ch, '0', '9', 0, 255);
   analogWrite(enPin, speed);
   Serial.println(speed);
  }
  else if (ch == '+')
  {
   Serial.println("CW");
   digitalWrite(in1Pin,LOW);
   digitalWrite(in2Pin,HIGH);
  }
  else if (ch == '-')
  {
   Serial.println("CCW");
   digitalWrite(in1Pin,HIGH);
   digitalWrite(in2Pin,LOW);
  }
  else
  {
   Serial.print("Unbekanntes Zeichen ");
   Serial.println(ch);
  }
 }
}
```

Diskussion

Das Rezept ähnelt Rezept 8.9, bei dem die Drehrichtung über die Pegel an den Pins IN1 und IN2 gesteuert wird. Zusätzlich wird aber die Geschwindigkeit über den `analogWrite`-Wert am EN-Pin geregelt (mehr über PWM erfahren Sie in Kapitel 7). Der Wert `0` hält den Motor an. Bei `255` läuft der Motor mit Höchstgeschwindigkeit. Die Geschwindigkeit verhält sich proportional zum Wert innerhalb dieses Wertebereichs.

8.11 Richtung und Geschwindigkeit von Bürstenmotoren über Sensoren steuern (L293 H-Brücke)

Problem

Sie wollen die Drehrichtung und Geschwindigkeit von Bürstenmotoren über das Feedback von Sensoren steuern. Zum Beispiel könnten Sie zwei Photozellen nutzen, die die Geschwindigkeit und Richtung eines Roboters kontrollieren, so dass er einem Lichtstrahl folgt.

Lösung

Die Lösung verwendet die gleichen Motor-Anschlüsse wie in Abbildung 8-10, nutzt aber zusätzlich noch zwei lichtempfindliche Widerstände (siehe Abbildung 8-12).

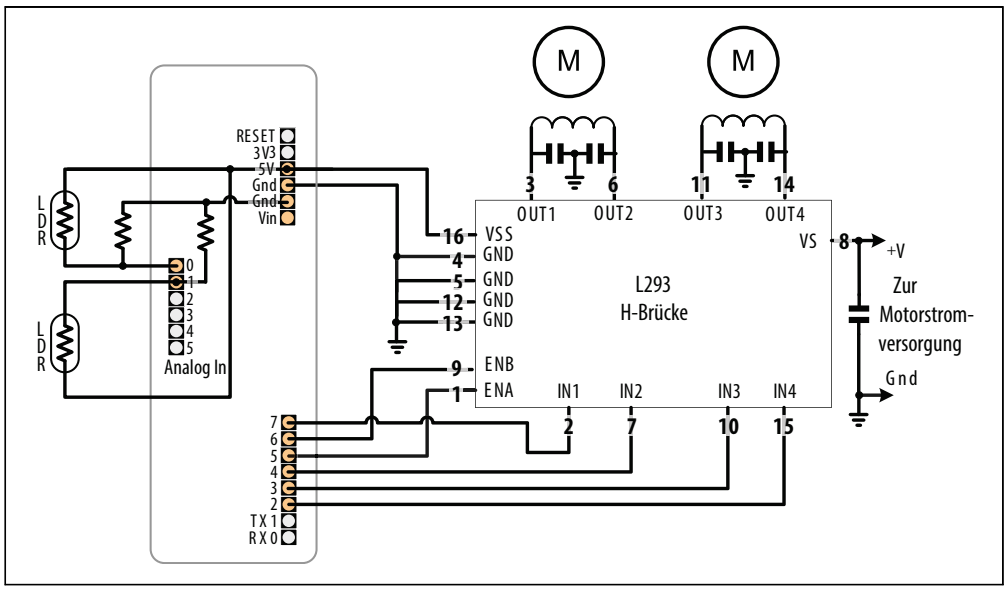


Abbildung 8-12: Zwei über Sensoren gesteuerte Motoren

Der Sketch überwacht die Helligkeit der Sensoren und steuert die Motoren in die Richtung, in der es heller ist:

```
/*
 * Brushed_H_Bridge_Direction Sketch
 * Nutzt Photosensoren zur Steuerung der Richtung
 * Roboter bewegt sich zum Licht hin
 */

 int leftPins[] = {5,7,4}; // Ein Pin für PWM, zwei Pins für die Richtung
 int rightPins[] = {6,3,2};

const int MIN_PWM     = 64; // Kann zwischen 0 und MAX_PWM liegen;
const int MAX_PWM     = 128; // Kann zwischen ca. 50 und 255 liegen;
const int leftSensorPin = 0; // Analogpins für Sensoren
const int rightSensorPin = 1;

int sensorThreshold = 0;    // Schwellwert für Bewegung

void setup()
{
 for(int i=1; i < 3; i++)
 {
  pinMode(leftPins[i], OUTPUT);
  pinMode(rightPins[i], OUTPUT);
 }
}

void loop()
{
 int leftVal = analogRead(leftSensorPin);
 int rightVal = analogRead(rightSensorPin);

 if(sensorThreshold == 0){  // Wurden die Sensoren kalibriert?
 // Wenn nicht, leicht über dem Durchschnitt liegenden Wert verwenden
   sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
 }

 if( leftVal > sensorThreshold || rightVal > sensorThreshold)
 {
  // Dem Licht folgen
  setSpeed(rightPins, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
  setSpeed(leftPins, map(leftVal ,0,1023, MIN_PWM, MAX_PWM));
 }
}

void setSpeed(int pins[], int speed )
{
 if(speed < 0)
 {
  digitalWrite(pins[1],HIGH);
  digitalWrite(pins[2],LOW);
  speed = -speed;
 }
 else
 {
  digitalWrite(pins[1],LOW);
```

```
    digitalWrite(pins[2],HIGH);
  }
  analogWrite(pins[0], speed);
}
```

Diskussion

Der Sketch regelt die Geschwindigkeit zweier Motoren als Reaktion auf die von zwei Photozellen gemessene Lichtmenge. Die Photozellen sind so angeordnet, dass die Erhöhung der Helligkeit auf einer Seite die Geschwindigkeit des Motors auf der anderen Seite erhöht. Das lässt den Roboter in die Richtung des Lichts fahren. Fällt das Licht gleichmäßig auf beide Zellen, fährt der Roboter in einer geraden Linie vorwärts. Ist nicht genug Licht vorhanden, hält der Roboter an.

Das Licht wird über die Analogeingänge 0 und 1 per `analogRead` (siehe Rezept 6.2) gemessen. Beim Programmstart wird das Umgebungslicht gemessen und als Schwellwert für die minimale Lichtstärke verwendet, die zur Bewegung des Roboters benötigt wird. Dieser Durchschnittswert wird noch um 100 erhöht, damit sich der Roboter bei kleinen Änderungen des Umgebungslichts nicht bewegt. Die Lichtstärke wird mit `analogRead` gemessen und über die `map`-Funktion in einen PWM-Wert umgewandelt. Legen Sie `MIN_PWM` auf einen Wert fest, der ausreicht, um den Roboter zu bewegen (zu niedrige Werte liefern nicht genug Drehmoment; Sie werden es einfach ausprobieren müssen). Setzen Sie `MAX_PWM` auf einen Wert (bis zu 255), der die Höchstgeschwindigkeit festlegt, mit der sich der Roboter bewegen soll.

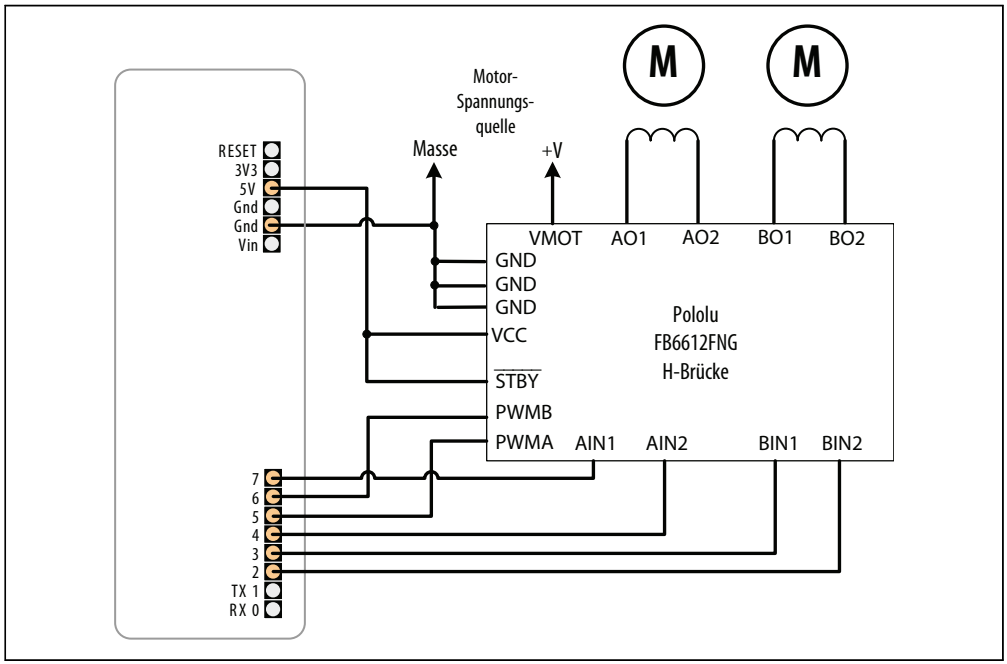


Abbildung 8-13: Anschluss der H-Brücke beim Pololu-Breakout-Board

Die Geschwindigkeit des Motors wird über die Funktion `setSpeed` gesteuert. Zwei Pins kontrollieren die Richtung eines Motors und ein weiterer Pin die Geschwindigkeit. Die Pin-Nummern werden in den Arrays `leftPins` und `rightPins` vorgehalten. Der erste Pin jedes Arrays ist der Geschwindigkeitspin, die beiden anderen sind für die Richtung verantwortlich.

Eine Alternative zum L293 ist der Toshiba FB6612FNG. Er kann in allen Rezepten eingesetzt werden, die den L293D verwenden. Abbildung 8-13 zeigt die Verschaltung des FB6612, wie sie beim Pololu-Breakout-Board (SparkFun ROB-09402) verwendet wird.

Sie können die Zahl der benötigten Pins reduzieren, indem Sie die Richtungspins über zusätzliche Hardware steuern. Dabei verwenden Sie für die Richtung nur jeweils einen Pin je Motor und einen Transistor oder ein Logik-Gatter, das den Pegel des anderen H-Brücken-Eingangs invertiert. Entsprechende Schaltungen finden Sie im Arduino-Wiki, aber wenn Sie etwas Fertiges wünschen, können Sie ein H-Brücken-Shield wie das Freeduino Motor Control Shield (NKC Electronics ARD-0015) oder das Ardumoto von SparkFun (DEV-09213) verwenden. Diese Shields passen direkt auf den Arduino. Sie müssen nur noch den Strom und die Spulen anschließen.

Der folgende Sketch wurde für das Ardumoto-Shield angepasst:

```
/*
 * Brushed_H_Bridge_Direction Sketch für Ardumotor-Shield
 * Nutzt Photosensoren zur Steuerung der Richtung
 * Roboter bewegt sich in Richtung eines Lichts
 */

 int leftPins[] = {10,12};  // Ein Pin für PWM, ein Pin für die Richtung
 int rightPins[] = {11,13};

const int MIN_PWM     = 64; // Kann zwischen 0 und MAX_PWM liegen;
const int MAX_PWM     = 128; // Kann zwischen ca. 50 und 255 liegen;
const int leftSensorPin = 0; // Analogpins für Sensoren
const int rightSensorPin = 1;

int sensorThreshold = 0;    // Schwellwert für Bewegung

void setup()
{
 pinMode(leftPins[1], OUTPUT);
 pinMode(rightPins[1], OUTPUT);
}

void loop()
{
 int leftVal = analogRead(leftSensorPin);
 int rightVal = analogRead(rightSensorPin);
 if(sensorThreshold == 0){  // Wurden die Sensoren kalibriert?
  // Wenn nicht, leicht über dem Durchschnitt liegenden Wert verwenden
   sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
 }

 if( leftVal > sensorThreshold || rightVal > sensorThreshold)
 {
```

```
  // Dem Licht folgen
  setSpeed(rightPins, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
  setSpeed(leftPins, map(leftVal, 0,1023, MIN_PWM, MAX_PWM));
 }
}

void setSpeed(int pins[], int speed )
{
 if(speed < 0)
 {
  digitalWrite(pins[1],HIGH);
  speed = -speed;
 }
 else
 {
  digitalWrite(pins[1],LOW);
 }
 analogWrite(pins[0], speed);
}
```

Die `loop`-Funktion entspricht der aus dem obigen Sketch. `setSpeed` umfasst weniger Code, weil die Hardware des Shields es erlaubt, die Richtung des Motors mit einem einzigen Pin zu steuern.

Die Zuordnung der Pins für das Freeduino-Shield ist wie folgt:

```
int leftPins[] = {10,13}; // PWM, Richtung
int rightPins[] = {9,12};  // PWM, Richtung
```

Nachfolgend implementieren wir die gleiche Funktionalität mit dem Adafruit Motor Shield (*http://www.ladyada.net/make/mshield/*); siehe Abbildung 8-14. Wir nutzen eine Bibliothek namens `AFMotor`, die von der Adafruit-Website heruntergeladen werden kann.

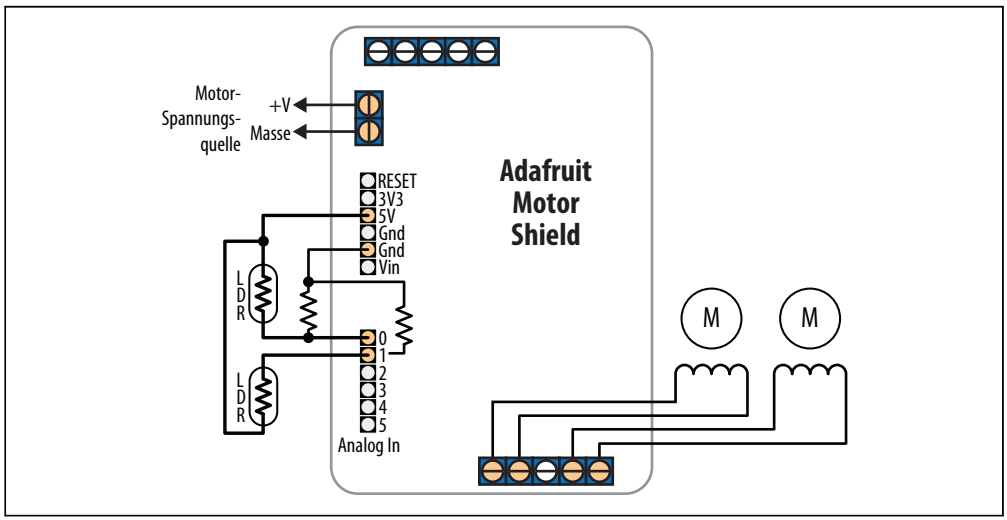


Abbildung 8-14: Verwendung des Adafruit Motor Shields

Das Adafruit-Shield unterstützt vier Anschlüsse für Motoren. Im folgenden Sketch sind die Motoren mit den Anschlüssen 3 und 4 verbunden:

```
/*
 * Brushed_H_Bridge_Direction Sketch für Adafruit Motor Shield
 * Nutzt Photosensoren zur Steuerung der Richtung
 * Roboter bewegt sich in Richtung eines Lichts
 */

#include "AFMotor.h" // Adafruit Motor Shield-Bibliothek

AF_DCMotor leftMotor(3, MOTOR12_1KHZ); // Motor #3, 1 KHz PWM nutzt Pin 5
AF_DCMotor rightMotor(4, MOTOR12_1KHZ); // Motor #4, 1 KHz PWM nutzt Pin 6

const int MIN_PWM     = 64;  // Kann zwischen 0 und MAX_PWM liegen;
const int MAX_PWM     = 128; // Kann zwischen ca. 50 und 255 liegen;
const int leftSensorPin  = 0;  // Analogpins für Sensoren
const int rightSensorPin = 1;

int sensorThreshold = 0; // Schwellwert für Bewegung

void setup()
{
}

void loop()
{
 int leftVal  = analogRead(leftSensorPin);
 int rightVal = analogRead(rightSensorPin);

 if(sensorThreshold == 0){  // Wurden die Sensoren kalibriert?
 // Wenn nicht, leicht über dem Durchschnitt liegenden Wert verwenden
   sensorThreshold = ((leftVal + rightVal) / 2) + 100 ;
 }

 if( leftVal > sensorThreshold || rightVal > sensorThreshold)
 {
  // Dem Licht folgen
  setSpeed(rightMotor, map(rightVal,0,1023, MIN_PWM, MAX_PWM));
  setSpeed(leftMotor, map(leftVal ,0,1023, MIN_PWM, MAX_PWM));
 }
}

void setSpeed(AF_DCMotor &motor, int speed )
{
 if(speed < 0)
 {
  motor.run(BACKWARD);
  speed = -speed;
 }
 else
 {
  motor.run(FORWARD);
 }
 motor.setSpeed(speed);
}
```

Wenn Sie keines der obigen Shields nutzen, müssen Sie auf dem Datenblatt nachsehen, welche Pins für PWM und Richtung verwendet werden müssen und das im Sketch entsprechend korrigieren.

Siehe auch

Das Datenblatt zum Pololu-Board: *http://www.pololu.com/file/0J86/TB6612FNG.pdf*

Die Produktseite des Freeduino-Shields: *http://www.nkcelectronics.com/freeduino-arduino-motor-control-shield-kit.html*

Die Produktseite des Ardumoto-Shields: *http://www.sparkfun.com/commerce/product_info.php?products_id=9213*

Die Dokumentation und die Bibliothek zum Adafruit Motor Shield finden Sie unter *http://www.ladyada.net/make/mshield/*

8.12 Einen bipolaren Schrittmotor ansteuern

Problem

Sie besitzen einen bipolaren (vieradrigen) Schrittmotor und wollen ihn aus einem Programm heraus über eine H-Brücke steuern.

Lösung

Der Sketch steuert den Motor über serielle Befehle. Ein numerischer Wert gefolgt von einem + bewegt ihn schrittweise in die eine Richtung, bei einem - in die andere. Beispielsweise vollzieht ein 24-Schritt-Motor mit »24+« eine vollständige Umdrehung in einer Richtung und mit »12-« eine halbe Umdrehung in der anderen Richtung (Abbildung 8-15 zeigt den Anschluss eines vierpoligen bipolaren Schrittmotors über eineL293-H-Brücke):

```
/*
 * Stepper_bipolar Sketch
 *
 * Schrittmotor wird über den seriellen Port gesteuert.
 * Ein numerischer Wert gefolgt von '+' oder '-' bewegt den Motor schrittweise
 *
 *
 * http://www.arduino.cc/en/Reference/Stepper
 */

#include <Stepper.h>

// Tragen Sie hier die Zahl der Schritte Ihres Motors ein
#define STEPS 24

// Instanz der stepper-Klasse erzeugen. Wir geben die
// Zahl der Motorschritte an und die Pins, mit denen
// er verbunden ist.
```

```
Stepper stepper(STEPS, 2, 3, 4, 5);

int steps = 0;


void setup()
{
 // Geschwindigkeit des Motors auf 30 U/min setzen
 stepper.setSpeed(30);
 Serial.begin(9600);
}

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();

  if(isDigit(ch)){ // Ist ch eine Ziffer?
   steps = steps * 10 + ch - '0'; // Ja, Wert akkumulieren
  }
  else if(ch == '+'){
   stepper.step(steps);
   steps = 0;
  }
  else if(ch == '-'){
   stepper.step(steps * -1);
   steps = 0;
  }
 }
}
```

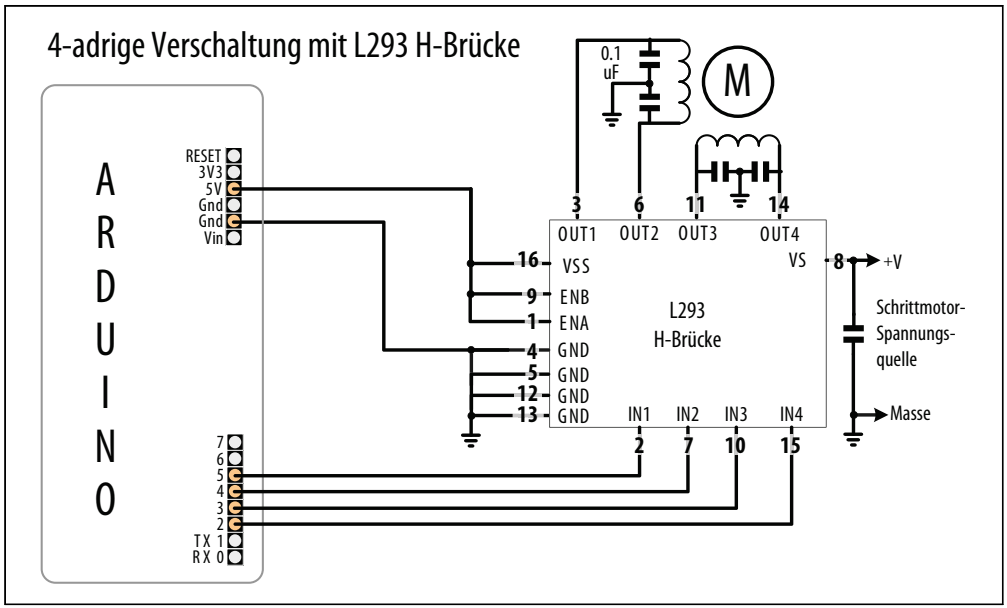


Abbildung 8-15: Vieradriger bipolarer Schrittmotor an L293-H-Brücke

Diskussion

Wenn Ihr Schrittmotor einen höheren Strom benötigt, als der L293 liefern kann (600 mA beim L293D), können Sie ersatzweise den SN754410 verwenden. Code und Verschaltung sind mit dem L293 identisch. Bei Strom von bis zu 2 Ampere können Sie den L298 verwenden. Sie können den obigen Sketch nutzen und müssen ihn entsprechend Abbildung 8-16 verschalten.

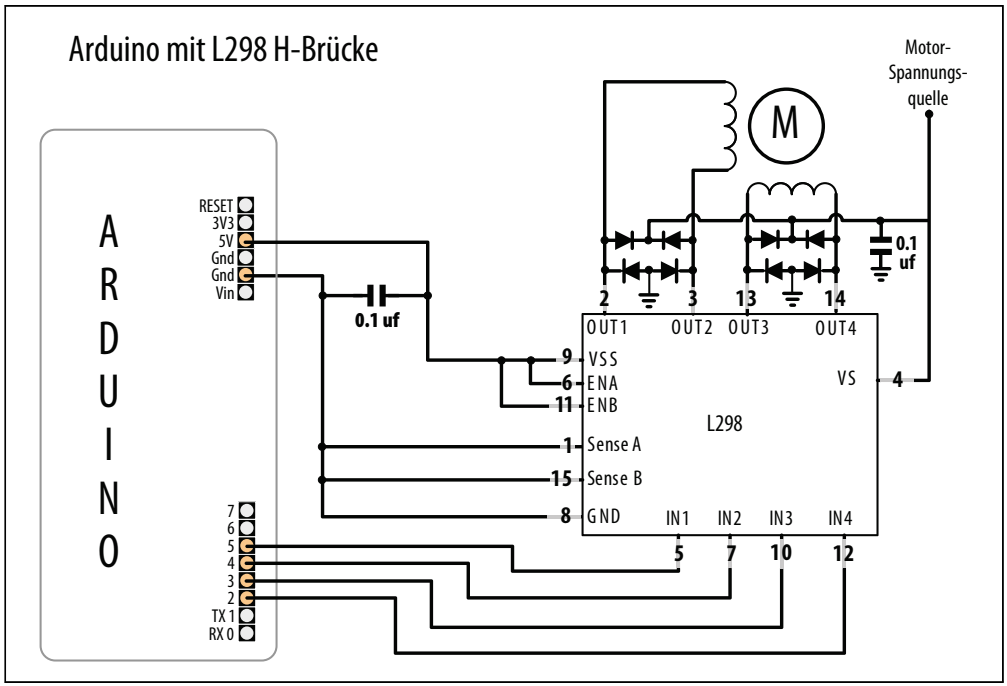


Abbildung 8-16: Unipolarer Schrittmotor an L298

Eine einfache Möglichkeit, einen L298 an den Arduino anzuschließen, bietet das SparkFun Ardumoto-Shield (DEV-09213). Es wird einfach auf das Arduino-Board aufgesteckt und benötigt nur die Verbindung zu den Motor-Spulen. Die Stromversorgung des Motors erfolgt über den Arduino-Pin Vin. In1/2 wird durch Pin 12 gesteuert und ENA liegt an Pin 10. In3/4 ist mit Pin 13 verbunden und ENB liegt an Pin 11. Nehmen Sie die folgenden Änderungen am obigen Code vor, um den Sketch mit dem Ardumoto nutzen zu können:

```
Stepper stepper(STEPS, 12,13);
```

Ersetzen Sie den Code in `setup()` durch

```
pinMode(10, OUTPUT);
digitalWrite(10, LOW);  // A aktivieren

pinMode(11, OUTPUT);
digitalWrite(11, LOW);  // B aktivieren
```

```
stepper.setSpeed(30);  // Geschwindigkeit auf 30 U/min setzen

Serial.begin(9600);
```

Der Code in `loop` ist mit dem obigen Sketch identisch.

Siehe auch

Weitere Informationen zur Verschaltung von Schrittmotoren finden Sie in Tom Igoes Schrittmotor-Notizen: *http://www.tigoe.net/pcomp/code/circuits/motors*.

8.13 Einen bipolaren Schrittmotor ansteuern (mit EasyDriver-Board)

Problem

Sie besitzen einen bipolaren (vieradrigen) Schrittmotor und wollen ihn aus einem Programm heraus über ein EasyDriver-Board steuern.

Lösung

Die Lösung ähnelt der aus Rezept 8.12 und verwendet auch die gleichen seriellen Befehle, nutzt aber das beliebte EasyDriver-Board. Abbildung 8-17 zeigt die Verschaltung.

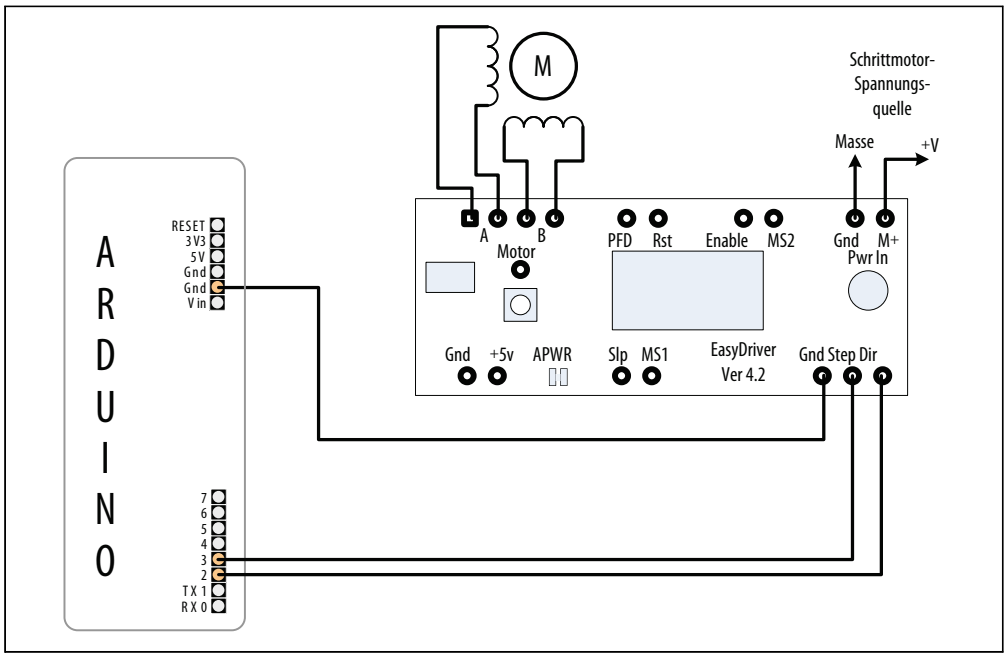


Abbildung 8-17: Anschluss des EasyDriver-Boards

Der folgende Sketch steuert die Schrittrichtung und die Anzahl der Schritte über den seriellen Port. Im Gegensatz zum Code in Rezept 8.12 wird die Stepper-Bibliothek nicht benötigt, da das EasyDriver-Board die Spulen des Motors per Hardware steuert:

```
/*
 * Stepper_Easystepper Sketch
 *
 * Schrittmotor wird über den seriellen Port gesteuert.
 * Ein numerischer Wert gefolgt von '+' oder '-' bewegt den Motor schrittweise
 *
 */

const int dirPin = 2;
const int stepPin = 3;

int speed = 100;   // Gewünschte Geschwindigkeit in Schritten pro Sekunde
int steps = 0;    // Anzahl der Schritte

void setup()
{
 pinMode(dirPin, OUTPUT);
 pinMode(stepPin, OUTPUT);
 Serial.begin(9600);
}

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();

  if(isDigit(ch)){        // Ist ch eine Ziffer?
   steps = steps * 10 + ch - '0';     // Ja, Wert akkumulieren
  }
  else if(ch == '+'){
   step(steps);
   steps = 0;
  }
  else if(ch == '-'){
   step(-steps);
   steps = 0;
  }
  else if(ch == 's'){
   speed = steps;
   Serial.print("Setze Geschwindigkeit auf ");
   Serial.println(steps);
   steps = 0;
  }
 }
}

void step(int steps)
{
 int stepDelay = 1000 / speed; //Verzögerung in ms für Geschwindigkeit in Schritten pro Sekunde
 int stepsLeft;

 // Vorzeichen von steps bestimmt Richtung
```

```
 if (steps > 0)
 {
  digitalWrite(dirPin, HIGH);
  stepsLeft = steps;
 }
 if (steps < 0)
 {
  digitalWrite(dirPin, LOW);
  stepsLeft = -steps;
 }
 // Schritte bei jedem Durchlauf dekrementieren
 while(stepsLeft > 0)
 {
  digitalWrite(stepPin,HIGH);
  delayMicroseconds(1);
  digitalWrite(stepPin,LOW);
  delay(stepDelay);
  stepsLeft--;    // Verbliebene Schritte dekrementieren
 }
}
```

Diskussion

Das EasyDriver-Board wird über die Pins M+ und Gnd (in der oberen rechten Ecke in Abbildung 8-17) mit Strom versorgt. Das Board arbeitet mit Spannungen zwischen 8 und 30 Volt. Die richtige Betriebsspannung Ihres Schrittmotors entnehmen Sie dem Datenblatt. Wenn Sie einen 5V-Schrittmotor nutzen, müssen Sie 5V an die mit Gnd und +5V gekennzeichneten Pins (unten links auf dem EasyDriver-Board) anlegen und den mit APWR gekennzeichneten Jumper entfernen (damit wird der boardeigene Regler abgetrennt und der Motor und das EasyDriver-Board werden über eine externe 5V-Quelle versorgt).

Sie können den Stromverbrauch bei nicht laufendem Motor reduzieren, indem Sie den Enable-Pin mit einem digitalen Ausgang verbinden und ihn auf `HIGH` setzen, um den Ausgang zu deaktivieren (ein `LOW` aktiviert ihn).

Die Schrittoptionen legen Sie fest, indem Sie MS1 und MS2 mit +5V (`HIGH`) oder Masse (`LOW`) verbinden (siehe Tabelle 8-2). Die Standardoptionen für das wie in Abbildung 8-17 verschaltete Board verwendet Achtel-Schritte (MS1 und MS2 sind `HIGH`, Reset ist `HIGH` und Enable ist `LOW`).

Tabelle 8-2: Mikroschritt-Optionen

Auflösung	MS1	MS2
Voller Schritt	LOW	LOW
Halber Schritt	HIGH	LOW
Viertel Schritt	LOW	HIGH
Achtel Schritt	HIGH	HIGH

Sie können den Code auch so modifizieren, dass die Umdrehungen pro Sekunde über die Geschwindigkeit bestimmt werden:

```
// Für Geschwindigkeit in U/min
int speed = 100;   // Gewünschte Geschwindigkeit in U/min
int stepsPerRevolution = 200; // Schritte für eine Umdrehung
```

Ändern Sie die erste Zeile der step-Funktion wie folgt:

```
int stepDelay = 60L * 1000L / stepsPerRevolution / speed; // Geschwindigkeit in U/min
```

Alles andere bleibt unverändert, aber der gesendete Geschwindigkeitsbefehl gibt nun die Umdrehungen pro Minute an.

8.14 Einen unipolaren Schrittmotor ansteuern (ULN2003A)

Problem

Sie verfügen über einen unipolaren (fünf- oder sechsadrigen) Schrittmotor und wollen ihn über einen ULN2003A Darlington-Treiber steuern..

Lösung

Schließen Sie den unipolaren Schrittmotor wie in Abbildung 8-18 zu sehen an. Der +V-Anschluss wird mit einer Stromquelle verbunden, die die vom Motor benötigte Spannung und den entsprechenden Strom liefert.

Der folgende Sketch bewegt den Motor über Befehle vom seriellen Port. Ein numerischer Wert gefolgt von einem + bewegt ihn in eine Richtung, ein - in die andere:

```
/*
 * Stepper Sketch
 *
 * Schrittmotor wird über den seriellen Port gesteuert
 * Ein numerischer Wert gefolgt von '+' oder '-' bewegt den Motor schrittweise
 *
 *
 * http://www.arduino.cc/en/Reference/Stepper
 */

#include <Stepper.h>

// Tragen Sie hier die Zahl der Schritte Ihres Motors ein
#define STEPS 24

// Instanz der stepper-Klasse erzeugen. Wir geben die
// Zahl der Motorschritte an und die Pins, mit denen
// er verbunden ist
Stepper stepper(STEPS, 2, 3, 4, 5);

int steps = 0;
```

```
void setup()
{
 stepper.setSpeed(30);  // Geschwindigkeit auf 30 U/min setzen
 Serial.begin(9600);
}

void loop()
{
 if ( Serial.available()) {
  char ch = Serial.read();

  if(isDigit(ch)){        // Ist ch eine Ziffer?
   steps = steps * 10 + ch - '0';     // Ja, Wert akkumulieren
  }
  else if(ch == '+'){
   stepper.step(steps);
   steps = 0;
  }
  else if(ch == '-'){
   stepper.step(steps * -1);
   steps = 0;
  }
  else if(ch == 's'){
   stepper.setSpeed(steps);
   Serial.print("Setze Geschwindigkeit auf ");
   Serial.println(steps);
   steps = 0;
  }
 }
}
```

Diskussion

Dieser Motortyp besitzt zwei Spulenpaare und jede Spule hat in der Mitte einen Anschluss. Bei Motoren mit fünf Anschlüssen sind die beiden mittleren Anschlüsse über einen einzelnen Draht nach außen geführt. Wenn die Anschlüsse nicht gekennzeichnet sind, können Sie die Verschaltung mit einem Multimeter bestimmen. Messen Sie den Widerstand der Anschlusspaare und finden Sie die beiden Paare mit dem maximalen Widerstand. Der mittlere Anschluss hat einen halb so großen Widerstand wie die ganze Spule. Eine Schritt-für-Schritt-Anweisung finden Sie unter *http://techref.massmind.org/techref/io/stepper/wires.asp*.

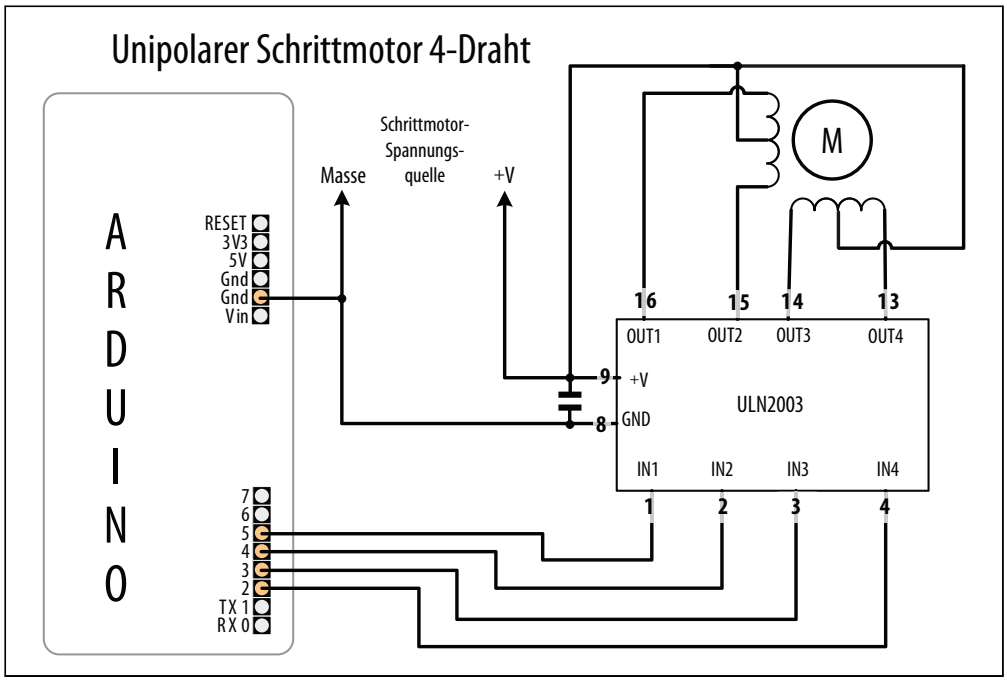


Abbildung 8-18: Anschluss eines unipolaren Schrittmotors über ULN2003-Treiber

KAPITEL 9

Audio-Ausgabe

9.0 Einführung

Der Arduino wurde nicht als Synthesizer konzipiert, aber natürlich kann er über ein Ausgabegerät wie einen Lautsprecher Töne erzeugen.

Töne werden durch Schwingung der Luft erzeugt. Ein Ton hat eine bestimmte Höhe, wenn man ihn ständig wiederholt. Der Arduino kann Töne erzeugen, indem er einen Lautsprecher oder ein *Piezo-Element* (ein kleiner keramischer Signalgeber, der bei Impulsen Töne erzeugt) ansteuert. Dabei werden elektronische Impulse in Schwingung am Lautsprecher umgewandelt, die die Luft vibrieren lassen. Die Höhe des Tons (die Frequenz) wird durch die Zeit bestimmt, die es braucht, um den Lautsprecher ein- und auszuschalten. Je kürzer diese Zeitspanne ist, desto höher ist die Frequenz.

Frequenzen werden in der Einheit Hertz gemessen. Sie gibt an, wie oft pro Sekunde das Signal seinen wiederkehrenden Zyklus durchläuft. Das menschliche Gehör nimmt Töne von etwa 20 Hertz (Hz) bis zu 20000 Hertz wahr (wenngleich das von Mensch zu Mensch variiert und sich mit dem Alter verändert).

Die Arduino-Software enthält eine `tone`-Funktion, mit der Sie Töne erzeugen können. Rezepte 9.1 und 9.2 zeigen, wie man die Funktion nutzt, um Töne zu erzeugen und Melodien abzuspielen. Die `tone`-Funktion arbeitet mit Hardware-Timern. Bei einem Standard-Arduino-Board kann nur jeweils ein Ton erzeugt werden. Sketches, bei denen der Timer (`timer2`) für andere Aufgaben benötigt wird, etwa `analogWrite` für Pin 9 oder 10, können die `tone`-Funktion nicht nutzen. Rezept 9.3 zeigt, wie man diese Einschränkung mithilfe einer Bibliothek umgehen und mehrere Töne erzeugen kann. Rezept 9.4 zeigt, wie man Töne ohne die `tone`-Funktion oder Hardware-Timer erzeugen kann.

Der Sound, den man erzeugen kann, indem man Impulse an einen Lautsprecher sendet, ist beschränkt und klingt nicht besonders gut. Ausgegeben wird eine Rechteckwelle (siehe Abbildung 9-1), die recht herb klingt und eher an ein antikes Computerspiel erinnert als an ein Musikinstrument.

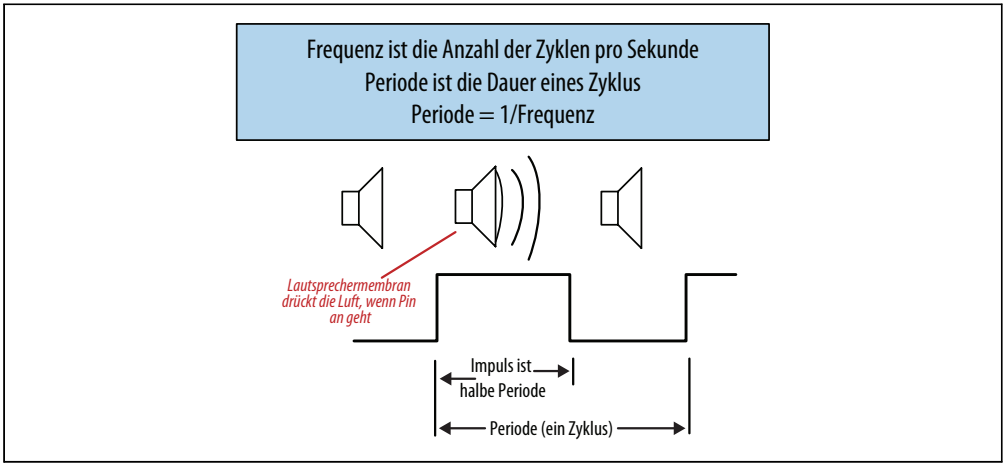


Abbildung 9-1: Töne mit digitalen Impulsen erzeugen

Es ist für den Arduino schwierig, musikalisch komplexere Sounds ohne externe Hardware zu erzeugen. Sie können ein Shield nutzen, um die Fähigkeiten des Arduino in dieser Hinsicht zu erweitern. Rezept 9.5 zeigt, wie man das Adafruit Wave Shield nutzt, um Audiodateien wiederzugeben, die auf einer Speicherkarte auf dem Shield gespeichert sind.

Sie können den Arduino auch nutzen, um ein externes Gerät anzusteuern, das Sound erzeugen kann. Rezept 9.6 zeigt, wie man MIDI-Nachrichten (Musical Instrument Digital Interface) an ein MIDI-Gerät sendet. Solche Geräte erzeugen qualitativ hochwertige Sounds für eine Vielzahl unterschiedlicher Instrumente und können viele Instrumente gleichzeitig spielen. Der Sketch in Rezept 9.6 zeigt, wie man MIDI-Nachrichten erzeugt, die eine Tonleiter spielen.

Rezept 9.7 enthält einen Überblick über eine Anwendung namens Arduino, die eine komplexe Software-Verarbeitung zu Synthetisierung von Sound verwendet.

Dieses Kapitel behandelt die vielen Möglichkeiten, mit denen Sie Sound elektronisch erzeugen können. Wenn Sie den Arduino akustische Instrumente (wie Glockenspiele, Trommeln und Klaviere) spielen lassen wollen, können Sie Aktuatoren wie Hubmagneten oder Servomotoren einsetzen, die in Kapitel 8 behandelt werden.

Viele Rezepte in diesem Kapitel steuern einen kleinen Lautsprecher oder ein Piezo-Element an. Wie man ihn mit einem Arduino-Pin verbindet, zeigt Abbildung 9-2.

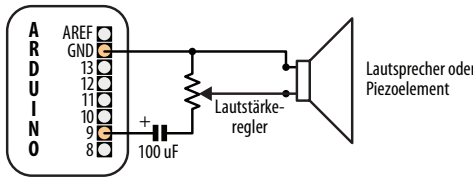


Abbildung 9-2: Anschluss eines Audio-Transducers

Der Lautstärkeregler ist ein variabler Widerstand, dessen Wert unkritisch ist. Alles zwischen 200 und 500 Ohm wird funktionieren. Der Kondensator ist ein 100 Mikrofarad-Elektrolyt, dessen positives Ende mit dem Arduino-Pin verbunden ist. Ein Lautsprecher funktioniert unabhängig davon, welcher Anschluss mit Masse verbunden ist, doch bei Piezo-Elementen spielt die Polung eine Rolle, d.h., Sie müssen den Masseanschluss (üblicherweise schwarz) mit dem Masse-Pin verbinden.

Alternativ können Sie den Ausgang mit einem externen Audioverstärker verbinden. Rezept 9.7 zeigt, wie ein Ausgangspin mit einem Klinkenstecker verbunden werden kann.

Der Spannungspegel (5 Volt) ist höher, als Audioverstärker es erwarten, weshalb Sie einen variablen 4,7K-Widerstand benötigen könnten, um die Spannung zu reduzieren. (Verbinden Sie ein Ende mit Pin 9 und das andere Ende mit Masse. Verbinden Sie dann den Schieber mit dem Klinkenstecker. Das Gehäuse des Klinkensteckers verbinden Sie mit Masse.)

9.1 Töne ausgeben

Problem

Sie wollen Töne über einen Lautsprecher oder einen anderen Audio-Transducer ausgeben. Sie wollen dabei die Frequenz und die Dauer des Tons festlegen.

Lösung

Verwenden Sie die Arduino-Funktion `tone`. Der folgende Sketch gibt einen Ton aus, dessen Frequenz über einen variablen Widerstand (oder anderen Sensor) am Analogeingang 0 festgelegt wird (siehe Abbildung 9-3):

```
/*
 * Tone Sketch
 *
 * Gibt Töne über einen Lautsprecher an Digitalpin 9 aus.
 * Die Frequenz wird durch den Wert bestimmt, der vom Analogport eingelesen wird.
 */

const int speakerPin = 9;   // Lautsprecher an Pin 9
const int pitchPin = 0;     // Poti bestimmt Frequenz des Tons

void setup()
{
}

void loop()
{
  int sensor0Reading = analogRead(pitchPin);   // Poti-Wert für Frequenz einlesen
  // Analogwert auf geeigneten Wertebereich abbilden
  int frequency  = map(sensor0Reading, 0, 1023, 100,5000); // 100Hz bis 5kHz
  int duration = 250;  // Dauer des Tons
```

```
  tone(speakerPin, frequency, duration); // Ton ausgeben
  delay(1000); // Eine Sekunde warten
}
```

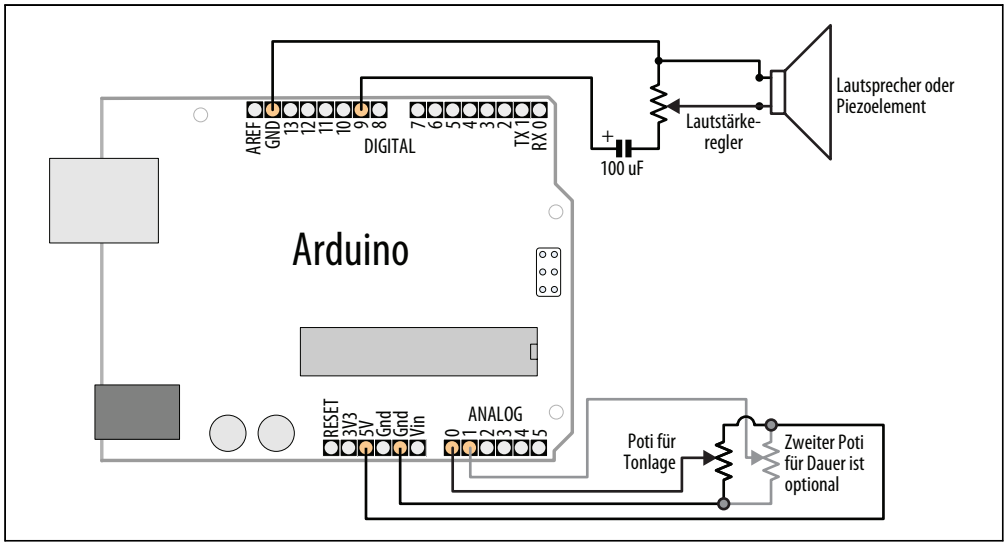


Abbildung 9-3: Anschluss für den Tone-Sketch

Die `tone`-Funktion verarbeitet bis zu drei Parameter: den Pin, an dem der Lautsprecher angeschlossen ist, die zu spielende Frequenz (in Hertz) und die Dauer des Tons (in Millisekunden). Der dritte Parameter ist optional. Lässt man ihn weg, wird der Ton so lange gespielt, bis `tone` erneut aufgerufen wird (oder `noTone`). Der Wert für die Frequenz wird in der folgenden Zeile auf geeignete Werte abgebildet:

```
int frequency = map(sensor0Reading, 0, 1023, 100,5000); //100Hz bis 5kHz
```

Die nachfolgende Variante nutzt einen zweiten variablen Widerstand (der untere rechte Poti in Abbildung 9-3), um die Dauer des Tons festzulegen:

```
const int speakerPin = 9;   // Lautsprecher an Pin 9
const int pitchPin = 0;     // 1. Poti bestimmt Frequenz des Tons
const int durationPin = 1;  // 2. Poti bestimmt Dauer des Tons

void setup()
{
}

void loop()
{
  int sensor0Reading = analogRead(pitchPin);   // Poti-Wert für Frequenz einlesen
  int sensor1Reading = analogRead(durationPin); // Poti-Wert für Dauer einlesen

  // Analogwerte auf geeignete Wertebereiche abbilden
  int frequency = map(sensor0Reading, 0, 1023, 100,5000); // 100Hz bis 5kHz
  int duration = map(sensor1Reading, 0, 1023, 100,1000);  // 0,1-1 Sekunde
```

```
  tone(speakerPin, frequency, duration); // Ton für
  delay(duration); // gewünschte Dauer ausgeben
}
```

Eine weitere Variante nutzt einen zusätzlichen Taster, so dass die Töne nur erzeugt werden, wenn der Taster gedrückt wird.

Aktivieren Sie mit der folgenden Zeile in `setup` die Pullup-Widerstände (ein Anschlussdiagramm und eine Erläuterung finden Sie in Rezept 5.2):

```
digitalWrite(inputPin,HIGH); // Internen Pullup am Eingangspin aktivieren
```

Modifizieren Sie den `loop`-Code so, dass die `tone`- und `delay`-Funktionen nur dann aufgerufen werden, wenn der Taster gedrückt wird:

```
if( digitalRead(inputPin) = LOW) // Tasterwert einlesen
{
  tone(speakerPin, frequency, duration); // Ton für
  delay(duration); // gewünschte Dauer ausgeben
}
```

Sie können nahezu jeden Audio-Transducer (Wandler) nutzen, um Töne mit dem Arduino zu erzeugen. Kleine Lautsprecher funktionieren sehr gut. Piezo-Elemente funktionieren ebenfalls und sind kostengünstig, robust und können aus alten Grußkarten wiederverwendet werden. Piezo-Elemente ziehen nur wenig Strom (sie sind hochohmige Elemente), d.h., sie können direkt an einen Pin angeschlossen werden. Lautsprecher haben üblicherweise einen deutlich kleineren Widerstand und müssen den Strom über einen Widerstand beschränken. Die Komponenten, die zum Aufbau der Schaltung in Abbildung 9-3 benötigt werden, sollten sich einfach auftreiben lassen. Hinweise zur Beschaffung dieser Teile finden Sie auf der Website zum Buch (*http://shop.oreilly.com/product/0636920022244.do*).

Siehe auch

Eine größere Funktionalität bietet die Tone-Bibliothek von Brett Hagman, die in Rezept 9.3 beschrieben wird.

9.2 Eine einfache Melodie spielen

Problem

Sie wollen den Arduino eine einfache Melodie spielen lassen.

Lösung

Nutzen Sie die `tone`-Funktion aus Rezept 9.1, um Töne auszugeben, die den Noten eines Musikinstruments entsprechen. Der folgende Sketch verwendet `tone`, um einen String von Noten auszugeben, und zwar das »Hallo, Welt« des Klavierunterrichts: »Morgen kommt der Weihnachtsmann«

```
/*
 * Twinkle Sketch
 *
 * Spielt "Morgen kommt der Weihnachtsmann"
 *
 * Lautsprecher an Digitalpin 9
 */

const int speakerPin = 9; // Lautsprecher an Pin 9

char noteNames[] =   {'C','D','E','F','G','a','b'};
unsigned int frequencies[] = {262,294,330,349,392,440,494};
const byte noteCount = sizeof(noteNames); // Anzahl der Noten (hier 7)

//Noten, Leerzeichen steht für eine Pause
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
const byte scoreLen = sizeof(score); // Anzahl der Noten

void setup()
{
}

void loop()
{
 for (int i = 0; i < scoreLen; i++)
 {
  int duration = 333;  // Jede Note für eine drittel Sekunde spielen
  playNote(score[i], duration); // Note ausgeben
 }

 delay(4000); // Vier Sekunden warten, bevor die Melodie wiederholt wird
}

void playNote(char note, int duration)
{
 // Den Ton ausgeben, der dem Notennamen entspricht
 for (int i = 0; i < noteCount; i++)
 {
  // Passeden noteNamen finden, um den Index der Note zu ermitteln
  if (noteNames[i] == note) // Notenname gefunden
   tone(speakerPin, frequencies[i], duration); //  Note ausgeben
 }
 // Gibt es keinen Treffer, ist die Note eine Pause
 delay(duration);
}
```

`noteNames` ist ein Array von Zeichen, die für die Noten einer Partitur stehen. Jeder Eintrag in diesem Array ist mit einer Frequenz verknüpft, die im `notes`-Array definiert ist. So hat die Note C (der erste Eintrag im `noteNames`-Array) eine Frequenz von 262 Hz (der erste Eintrag im `notes`-Array).

`score` ist ein Array, das die Noten des zu spielenden Stücks enthält:

```
// Leerzeichen steht für eine Pause
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
```

Bei jedem Zeichen in score, das einem Zeichen im noteNames-Array entspricht, wird die entsprechende Note gespielt. Das Leerzeichen wird als Pause genutzt, aber jedes Zeichen, das nicht in noteNames definiert ist, erzeugt ebenfalls eine Pause (d.h., es wird keine Note gespielt).

Der Sketch ruft playNote für jedes Zeichen in score für eine drittel Sekunde auf.

Die playNote-Funktion sucht das noteNames-Array nach einem passenden Zeichen ab und verwendet bei einem Treffer den entsprechenden Eintrag im frequencies-Array, um einen Ton mit der gewünschten Frequenz auszugeben.

Jede Note wird für die gleiche Dauer ausgegeben. Wollen Sie auch die Dauer für jede Note festlegen, können Sie den Sketch um den folgenden Code erweitern:

```
byte beats[scoreLen] = {1,1,1,1,1,1,2, 1,1,1,1,1,1,2,1,
                        1,1,1,1,1,1,2, 1,1,1,1,1,1,2,1,
                        1,1,1,1,1,1,2, 1,1,1,1,1,1,2};
byte beat = 180; // Takte pro Minute für Achtelnoten
unsigned int speed = 60000 / beat; // Zeit in ms für einen Takt
```

beats ist ein Array mit der Dauer jeder Note: 1 ist eine Achtelnote,2 eine Viertelnote und so weiter.

beat ist die Anzahl der Takte pro Minute.

Die speed-Zeile wandelt die Takte pro Minute in eine Dauer in Millisekunden um.

Die einzige Änderung am loop-Code besteht darin, die Spieldauer aus dem beats-Array zu ermitteln. Ändern Sie

```
int duration = 333; // Jede Note für eine drittel Sekunde spielen
```

in

```
int duration = beats[i] * speed; // Dauer über beats-Array bestimmen
```

9.3 Mehr als einen Ton gleichzeitig erzeugen

Problem

Sie wollen zwei Töne gleichzeitig ausgeben. Die Arduino Tone-Bibliothek kann bei einem Standard-Board nur einen einzelnen Ton erzeugen, aber Sie brauchen zwei Töne gleichzeitig. Beachten Sie, dass das Mega-Board mehr Timer besitzt und bis zu sechs Töne erzeugen kann.

Lösung

Die Arduino Tone-Bibliothek ist auf einen einzelnen Ton beschränkt, da für jeden Ton ein eigener Timer benötigt wird. Zwar besitzt ein Standard- Arduino-Board drei Timer, aber einer wird von der millis-Funktion und ein weiterer für Servos genutzt. Das folgende Rezept nutzt eine von Brett Hagman (dem Autor der Arduino tone-Funktion) entwickelte Bibliothek. Diese Bibliothek ermöglicht es, mehrere Töne gleichzeitig zu erzeugen. Sie

können sie von *http://code.google.com/p/rogue-code/wiki/ToneLibraryDocumentation* herunterladen.

Hier ein Beispiel-Sketch aus dem Download, der zwei Töne ausgibt, die man über den seriellen Port eingeben kann:

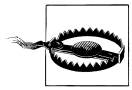
```
/*
 * Dual Tones - Simultane Tonerzeugung
 *  Spielt die Noten 'a' bis 'g', die über den seriellen Monitor gesendet werden
 * Kleinbuchstaben für den ersten, Großbuchstaben für den zweite Ton
 * 's' beendet den gerade gespielten Ton
 */
#include <Tone.h>

int notes[] = { NOTE_A3,
         NOTE_B3,
         NOTE_C4,
         NOTE_D4,
         NOTE_E4,
         NOTE_F4,
         NOTE_G4 };

// Sie können die Töne als Array deklarieren
Tone notePlayer[2];

void setup(void)
{
 Serial.begin(9600);
 notePlayer[0].begin(11);
 notePlayer[1].begin(12);
}

void loop(void)
{
 char c;

 if(Serial.available())
 {
  c = Serial.read();

  switch(c)
  {
   case 'a'...'g':
    notePlayer[0].play(notes[c - 'a']);
    Serial.println(notes[c - 'a']);
    break;
   case 's':
    notePlayer[0].stop();
    break;

   case 'A'...'G':
    notePlayer[1].play(notes[c - 'A']);
    Serial.println(notes[c - 'A']);
    break;
   case 'S':
    notePlayer[1].stop();
```

```
      break;

     default:
      notePlayer[1].stop();
      notePlayer[0].play(NOTE_B2);
      delay(300);
      notePlayer[0].stop();
      delay(100);
      notePlayer[1].play(NOTE_B2);
      delay(300);
      notePlayer[1].stop();
      break;
    }
   }
  }
```

Diskussion

Um die Ausgabe zweier Töne an einem einzelnen Lautsprecher zu mischen, verwenden Sie einen 500 Ohm-Widerstand an jedem Ausgangspin und verbinden sie am Lautsprecher. Der andere Anschluss des Lautsprechers wird mit Masse verbunden (wie bei den vorangegangenen Sketches gezeigt).

Bei einem Standard-Arduino-Board nutzt der erste Ton Timer 2 (d.h., PWM ist an den Pins 9 und 10 nicht verfügbar). Der zweite Ton verwendet Timer 1 (d.h., die Servo-Bibliothek und PWM an den Pins 11 und 12 funktionieren nicht). Bei einem Mega-Board verwendet jeder simultane Ton die Timer in der folgenden Reihenfolge: 2, 3, 4, 5, 1, 0.



Es ist möglich, auf einem Standard-Arduino-Board mehr als drei Noten simultan auszugeben (und mehr als sechs bei einem Mega), aber `millis` und `delay` funktionieren dann nicht mehr richtig. Wenn Sie auf der sicheren Seite sein wollen, verwenden Sie simultan nicht mehr als zwei Töne (bzw. fünf beim Mega).

9.4 Einen Ton erzeugen und eine LED ansteuern

Problem

Sie wollen Töne über einen Lautsprecher oder einen anderen Audio-Transducer ausgeben, müssen sie aber per Software erzeugen, da Sie den Timer brauchen, z.B. weil Sie `analogWrite` für Pin 9 oder 10 benötigen.

Lösung

Die in den vorangegangenen Rezepten behandelte `tone`-Funktion ist einfach zu nutzen, benötigt aber einen Hardware-Timer, den Sie für andere Aufgaben wie `analogWrite` benötigen könnten. Der folgende Code nutzt keinen Timer, macht aber nichts anderes, während eine Note gespielt wird. Im Gegensatz zur Arduino `tone`-Funktion »blockiert«

die hier beschriebene playTone-Funktion, d.h., sie kehrt erst zurück, wenn die Note gespielt wurde.

Der Sketch spielt sechs Noten, jede mit der doppelten Frequenz der vorangegangenen (also eine Oktave höher). Die playTone-Funktion erzeugt einen Ton der angegebenen Dauer an einem Lautsprecher oder Piezo-Element, das mit einem digitalen Ausgangspin und Masse verbunden ist (siehe Abbildung 9-4):

```
byte speakerPin = 9;
byte ledPin = 10;

void setup()
{
 pinMode(speakerPin, OUTPUT);
}

void playTone(int period, int duration)
{
// period ist ein Takt
// duration ist die Dauer in Millisekunden
 int pulse = period / 2;
 for (long i = 0; i < duration * 1000L; i += period )
 {
  digitalWrite(speakerPin, HIGH);
  delayMicroseconds(pulse);
  digitalWrite(speakerPin, LOW);
  delayMicroseconds(pulse);
 }
}

void fadeLED(){
 for (int brightness = 0; brightness < 255; brightness++)
 {
  analogWrite(ledPin, brightness);
  delay(2);
 }
 for (int brightness = 255; brightness >= 0; brightness--) {
  analogWrite(ledPin, brightness);
  delay(2);
 }



}
void loop()
{
 // Eine Note mit einer Dauer von 15289 ist ein tiefes C (das zweitniedrigste C auf einem Klavier)
 for(int period=15289; period >= 477;  period=period / 2)  // 6 Oktaven spielen
 {
   playTone( period, 200); // Ton für 200 ms spielen
 }
 fadeLED();
}
```

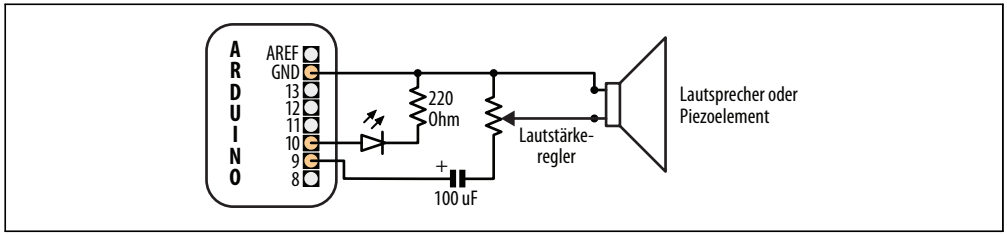


Abbildung 9-4: Anschluss von Lautsprecher und LED

Diskussion

Die beiden von `playTone` verwendeten Werte sind `period` und `duration`. Die Variable `period` repräsentiert die Zeit eines Taktes des zu spielenden Tons. Der Lautsprecher wird für die in `period` angegebene Dauer (in Mikrosekunden) ein- und ausgeschaltet. Die `for`-Schleife wiederholt dieses Pulsieren für die im `duration`-Argument festgelegte Zeit in Millisekunden.

Wenn Sie lieber mit Frequenzen arbeiten als mit Takten, können Sie die reziproke Beziehung zwischen Frequenz und Zeit nutzen. Die Schwingungsdauer ist 1 durch die Frequenz. Sie benötigen die Schwingungsdauer in Mikrosekunden. Da eine Sekunde einer Million Mikrosekunden entspricht, wird die Schwingungsdauer als 1000000L / Frequenz berechnet (das »L« am Ende der Zahl weist den Compiler an, mit langen Integerwerten zu rechnen, damit der Wertebereich normaler Integerzahlen nicht überschritten wird. Beachten Sie hierzu die Erläuterungen in Rezept 2.2):

```
void playFrequency(int frequency, int duration)
{
 int period = 1000000L / frequency;
 int pulse = period / 2;
```

Der Rest des Codes ist mit `playTone` identisch:

```
 for (long i = 0; i < duration * 1000L; i += period )
 {
  digitalWrite(speakerPin, HIGH);
  delayMicroseconds(pulse);
  digitalWrite(speakerPin, LOW);
  delayMicroseconds(pulse);
 }
}
```

Der Code dieses Rezepts wartet, bis der Ton vollständig gespielt wurde, bevor er sich anderen Dingen zuwenden kann. Es ist möglich, den Ton im Hintergrund zu erzeugen (ohne darauf warten zu müssen, dass er vollständig ausgegeben wurde), indem man den Code zur Tongenerierung in einen Interrupthandler packt. Der Quellcode der `tone`-Funktion, der mit der Arduino-Distribution geliefert wird, zeigt, wie das geht.

Siehe auch

Rezept 9.7

Hier einige Beispiel für etwas komplexere Audio-Synthesen, die mit dem Arduino erreicht werden können:

Pulse-Code Modulation
: PCM erlaubt die Approximierung analoger Audiosignale mit digitalen Signalen. Ein Arduino-Wiki-Artikel erklärt, wie man 8-Bit-PCM mit Hilfe eines Timers erzeugt. Den Artikel finden Sie unter *http://www.arduino.cc/playground/Code/PCMAudio*.

Pocket Piano-Shield
: Critter and Guitaris Pocket Piano-Shield bietet Ihnen eine klavierähnliche Tastatur, Wavetable-Synthese, FM-Synthese und mehr. Siehe *http://www.critterandguitari.com/home/store/arduino-piano.php*.

9.5 Eine WAV-Datei abspielen

Problem

Der Arduino soll aus einem Programm heraus das Abspielen einer WAV-Datei anstoßen.

Lösung

Dieser Sketch nutzt das Adafruit Wave Shield und basiert auf einem Beispiel-Sketch von der Produktseite (*http://www.adafruit.com/index.php?main_page=product_info&products_id=94*).

Der Sketch spielt eine von neun Dateien ab. Welche Datei das ist, hängt davon ab, in welcher Stellung sich der variable Widerstand an Analogeingang 0 befindet, wenn die Taste an Pin 15 gedrückt wird:

```
/*
 * WaveShieldPlaySelection Sketch
 *
 * Ausgewählte WAV-Datei abspielen
 *
 * Position des variablen Widerstands beim Drücken des Tasters wählt die Datei aus
 *
 */

#include <FatReader.h>
#include <SdReader.h>

#include "WaveHC.h"
#include "WaveUtil.h"

SdReader card;  // Dieses Objekt enthält Informationen zur Karte
FatVolume vol;  // Enthält Informationen zur Partition der Karte
FatReader root; // Enthält Informationen zum Root-Verzeichnis des Volumes
```

```
FatReader file;  // Dieses Objekt repräsentiert die WAV-Datei
WaveHC wave;     // Wave- (Audio) Objekt - es wird nur jeweils eine Datei abgespielt


const int buttonPin = 15;
const int potPin = 0; // Analogeingang an Pin 0

char * wavFiles[] = {
"1.WAV","2.WAV","3.WAV","4.WAV","5.WAV","6.WAV","7.WAV","8.WAV","9.WAV"};

void setup()
{
 Serial.begin(9600);
 pinMode(buttonPin, INPUT);
 digitalWrite(buttonPin, HIGH); // Pullup-Widerstand einschalten

 if (!card.init())
 {
  // Etwas ist schiefgegangen, sdErrorCheck gibt einen Fehlercode aus
  putstring_nl("Initialisierung der Karte fehlgeschlagen!");
  sdErrorCheck();
  while(1);                // 'Anhalten' - wir machen nichts!
 }


 // Optimiertes Lesen aktivieren - bei manchen Karten kann es zum Timeout kommen
 card.partialBlockRead(true);

 // FAT-Partition finden!
 uint8_t part;
 for (part = 0; part < 5; part++)     // Wir müssen bis zu 5 Slots untersuchen
 {
  if (vol.init(card, part))
   break;                 // Wir haben eine gefunden, also Schleife abbrechen
 }
 if (part == 5)               // Gültige Partitionen sind 0 bis 4; andere sind nicht gültig
 {
  putstring_nl("Keine gültige FAT-Partition!");
  sdErrorCheck();             // Etwas ist schiefgegangen. Fehler ausgeben
  while(1);                // und 'anhalten' - nichts tun!
 }

 // Dem Benutzer mitteilen, was wir gefunden haben
 putstring("Verwende Partition ");
 Serial.print(part, DEC);
 putstring(", Typ ist FAT");
 Serial.println(vol.fatType(),DEC);   // FAT16 oder FAT32?

 // Versuche, Stammverzeichnis zu öffnen
 if (!root.openRoot(vol))
 {
  putstring_nl("Kann Stammvereichnis nicht oeffnen!"); // Etwas ist schiefgegangen, Fehler
ausgeben
  while(1);                 // Dann 'anhalten' - nichts tun!
 }

 // An diesem Punkt waren alle Vorbereitungen erfolgreich.
```

```
 putstring_nl("Bereit!");
}

void loop()
{
 if(digitalRead(buttonPin) == LOW)
 {
  int value = analogRead(potPin);
  int index = map(value,0,1023,0,8); // Index auf eine der 9 Dateien
  playcomplete(wavFiles[index]);
  Serial.println(value);
 }
}

// Datei ohne Pause von Anfang bis Ende komplett abspielen.
void playcomplete(char *name)
{
 // playfile findet die Datei und spielt sie ab
 playfile(name);
 while (wave.isplaying) {
  // Wir warten, solange sie abgespielt wird
 }
 // Fertig mit Abspielen
}

void playfile(char *name) {
 // Prüfen, ob das Wave-Objekt gerade etwas macht
 if (wave.isplaying) {
 // Es wird bereits atwas abgespielt,
  wave.stop(); // also anhalten
 }
 // Im Stammverzeichis nachsehen und Datei öffnen
 if (!file.open(root, name)) {
  putstring("Kann Datei nicht oeffnen: ");
  Serial.print(name);
  return;
 }
 // Datei einlesen und in wave-Objekt umwandeln
 if (!wave.create(file)) {
  putstring_nl("Keine gueltige WAV-Datei");
  return;
 }
 // start playback
 wave.play();
}

void sdErrorCheck(void)
{
 if (!card.errorCode()) return;
 putstring("\n\rSD E/A-Fehler: "); Serial.print(card.errorCode(), HEX);
 putstring(", ");
 Serial.println(card.errorData(), HEX);
 while(1)
  ; // Bei Fehler anhalten
}
```

Diskussion

Das Wave-Shield liest Daten von einer SD-Karte ein. Es nutzt eine eigene Bibliothek, die von der Ladyada-Website (*http://www.ladyada.net/make/waveshield/*) heruntergeladen werden kann. Die abzuspielenden WAV-Dateien müssen über einen Computer auf der Speicherkarte abgelegt werden. Sie müssen als 22 kHz, 12-Bit unkomprimierte Monodateien vorliegen und die Dateinamen müssen das 8.3-Format verwenden. Sie können das Open Source Audio-Utility Audacity verwenden, um Audiodateien zu editieren und in das richtige Format umzuwandeln. Das Wave-Shield liest die Audiodatei von der SD-Karte ein d.h., die Länge der Audiodatei wird nur durch die Größe der Speicherkarte beschränkt.

Siehe auch

Die Bibliothek und Dokumentation zum Ladyada Wave-Shield: *http://www.ladyada.net/make/waveshield/*

Audacity Audiobearbeitungs- und Konvertierungssoftware: *http://audacity.sourceforge.net/*

SparkFun bietet verschiedene Audiomodule an, darunter ein Audio-Sound-Modul (*http://www.sparkfun.com/products/9534*) und ein MP3-Breakout-Board (*http://www.sparkfun.com/products/8954*).

9.6 MIDI steuern

Problem

Sie wollen mit dem Arduino Musik über einen MIDI-Synthesizer abspielen lassen.

Lösung

Um die Verbindung mit einem MIDI-Gerät herzustellen, benötigen Sie eine(n) fünfpoligen DIN-Stecker/-Buchse. Bei einer Buchse benötigen Sie außerdem ein Verlängerungskabel, um die Verbindung mit dem Gerät herzustellen. Verbinden Sie den MIDI-Anschluss mit dem Arduino über einen 220 Ohm-Widerstand, wie in Abbildung 9-5 zu sehen.

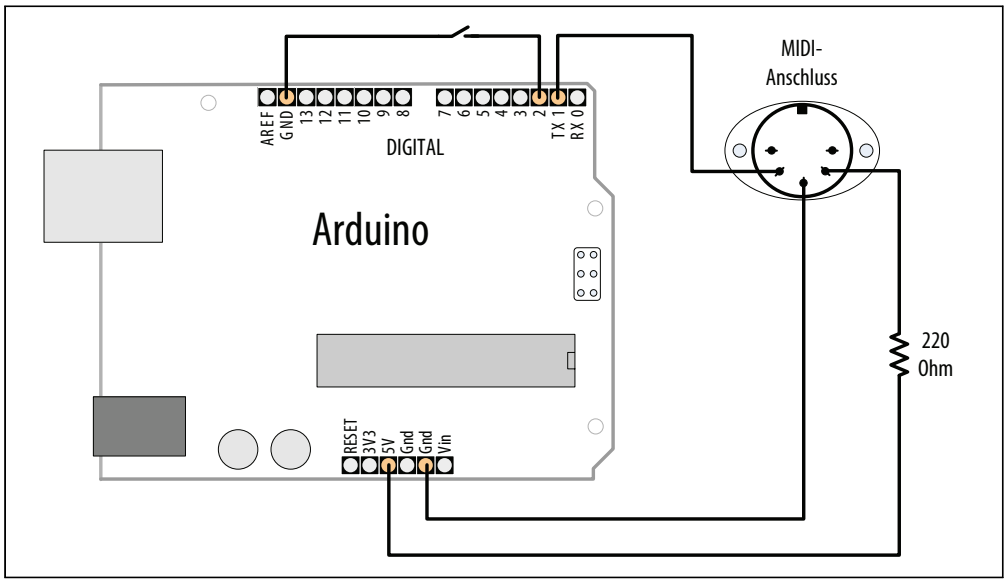


Abbildung 9-5: MIDI-Anschlüsse

Beim Upload des Codes auf den Arduino sollten Sie das MIDI-Gerät abklemmen, da es den Upload stören könnte. Nachdem der Sketch hochgeladen wurde, verbinden Sie das MIDI-Gerät mit dem Arduino-Ausgang. Ein Musikstück wird gespielt, sobald Sie den mit Pin 2 verbundenen Taster drücken:

```
/*
midiOut Sketch
Sendet MIDI-Nachrichten an ein MIDI-Instrument, die ein Musikstück abspielen, sobald der Taster
an Pin 2 gedrückt wird
*/

//Diese Zahlen spezifizieren die Noten
const byte notes[8] = {60, 62, 64, 65, 67, 69, 71, 72};
//Sie sind Teil der MIDI-Spezifikation
const int length = 8;
const int switchPin = 2;
const int ledPin = 13;

void setup() {
 Serial.begin(31250);
 pinMode(switchPin, INPUT);
 digitalWrite(switchPin, HIGH);
 pinMode(ledPin, OUTPUT);
}

void loop() {
 if (digitalRead(switchPin == LOW))
 {
  for (byte noteNumber = 0; noteNumber < 8; noteNumber++)
  {
```

```
    playMidiNote(1, notes[noteNumber], 127);
    digitalWrite(ledPin, HIGH);
    delay(70);
    playMidiNote(1, notes[noteNumber], 0);
    digitalWrite(ledPin, HIGH);
    delay(30);
   }
  }
}

void playMidiNote(byte channel, byte note, byte velocity)
{
 byte midiMessage= 0x90 + (channel - 1);
 Serial.write(midiMessage);
 Serial.write(note);
 Serial.write(velocity);
}
```

Diskussion

Der Sketch nutzt den seriellen Port, um die MIDI-Daten zu senden. Die Schaltung an Pin 1 kann also den Upload des Codes auf das Board stören. Entfernen Sie den Draht von Pin 1 während des Uploads und schließen Sie ihn danach wieder an.

MIDI wurde ursprünglich genutzt, um digitale Musikinstrumente miteinander zu verbinden, so dass sie sich gegenseitig steuern können. Die MIDI-Spezifikation beschreibt die elektrischen Anschlüsse und die Nachrichten, die Sie senden müssen.

MIDI ist genau genommen eine serielle Verbindung (mit einer ungewöhnlichen Geschwindigkeit von 31250 Baud), d.h., der Arduino kann MIDI-Nachrichen über seinen seriellen Hardware-Port an den Pins 0 und 1 senden. Da der serielle Port für die MIDI-Nachrichten benötigt wird, können wir keine Meldungen an den seriellen Monitor ausgeben, weshalb der Sketch die LED an Pin 13 aufblinken lässt, wenn er eine Note sendet.

Jede MIDI-Nachricht besteht aus zumindest einem Byte. Dieses Byte legt fest, was zu tun ist. Einige Befehle benötigen keine weiteren Informationen, andere brauchen zusätzliche Daten. Die Nachricht in diesem Sketch ist *note on*, die zwei zusätzliche Informationen benötigt: eine Note und ihre Lautstärke. Beide Informationen werden als Bytes im Bereich von 0 bis 127 übertragen.

Der Sketch initialisiert den seriellen Port auf eine Geschwindigkeit von 31250 Baud. Der restliche MIDI-spezifische Code befindet sich in der Funktion `playMidiNote`:

```
void playMidiNote(byte channel, byte note, byte velocity)
{
 byte midiMessage= 0x90 + (channel - 1);
 Serial.write(midiMessage);
 Serial.write(note);
 Serial.write(velocity);
}
```

Die Funktion verlangt drei Parameter und berechnet das erste zu sendende Byte über die Kanal-Information.

MIDI-Informationen werden über verschiedene Kanäle (1 bis 16) gesendet. Jeder Kanal kann mit einem anderen Instrument belegt werden, so dass Mehrkanal-Musik gespielt werden kann. Der Befehl für *note on* (zum Spielen eines Sounds) ist eine Kombination aus 0x90 (die höherwertigen vier Bits sind b1001) und dem gewünschten MIDI-Kanal in den unteren vier Bits (mit Werten zwischen b0000 und b1111). Das Byte verwendet 0 bis 15 für die Kanäle 1 bis 16, weshalb wir noch eine 1 abziehen.

Dann wird die Note gesendet und die Lautstärke (die bei MIDI *Geschwindigkeit,* engl. velocity, genannt wird, da sie ursprünglich angab, wie schnell die Taste einer Tastatur bedient wird).

Die seriellen `write`-Anweisungen geben an, dass die Daten als Bytes (und nicht als ASCII-Werte) gesendet werden sollen. `println` wird nicht verwendet, weil ein Zeilenvorschub-Zeichen zusätzliche Bytes im Signal erzeugen würde, die wir dort nicht gebrauchen können.

Der Sound wird mit der gleichen Nachricht ausgeschaltet, nur dass die Lautstärke auf 0 gesetzt wird.

Dieses Rezept funktioniert mit MIDI-Geräten, die einen fünfpoligen MIDI-In-Anschluss haben. Wenn Ihr MIDI-Gerät einen USB-Anschluss besitzt, funktioniert es nicht. Der Arduino kann MIDI-Musikprogramme auf Ihrem Computer nicht ohne zusätzliche Hardware (einen MIDI-nach-USB-Adapter) steuern. Zwar besitzt der Arduino einen USB-Anschluss, doch Ihr Computer erkennt ihn als serielles Gerät und nicht als MIDI-Gerät.

Siehe auch

Um MIDI zu senden und zu empfangen, sehen Sie sich die MIDI-Bibliothek auf *http://www.arduino.cc/playground/Main/MIDILibrary* an.

MIDI-Nachrichten werden detailliert in *http://www.midi.org/techspecs/midimessages.php* beschrieben.

Weitere Informationen zum SparkFun MIDI Breakout-Shield (BOB-09598), finden Sie auf *http://www.sparkfun.com/products/9598*.

Um den Arduino Uno als natives USB-MIDI-Gerät einzurichten, sehen Sie sich Rezept 18.14 an.

9.7 Audio-Synthesizer

Problem

Sie wollen komplexe Sounds erzeugen, wie sie bei der Produktion elektronischer Musik verwendet werden.

Lösung

Die Simulation von Audio-Oszillatoren, wie sie in Sound-Synthesizern verwendet werden, ist kompliziert, aber Peter Knight hat einen Sketch namens Auduino entwickelt, der es dem Arduino ermöglicht, komplexere und interessantere Sounds zu erzeugen.

Laden Sie den Sketch von *http://code.google.com/p/tinkerit/wiki/Auduino* herunter.

Verbinden Sie fünf lineare 4,7 KOhm-Potentiometer mit den Analogpins 0 bis 4 (siehe Abbildung 9-6). Potentiometer mit langem Stift sind besser, da man sie besser einstellen kann. Pin 3 wird als Audio-Ausgang verwendet und wird über einen Klinkenstecker mit dem Verstärker verbunden.

Diskussion

Der Sketch-Code ist komplex, da er direkt die Hardware-Timer manipuliert, um die gewünschten Frequenzen zu erzeugen, die per Software transformiert werden, um die gewünschten Audio-Effekte zu erzeugen. Der Code ist hier nicht enthalten, weil Sie ihn nicht verstehen müssen, um Auduino nutzen zu können.

Auduino verwendet eine als *Granularsynthese* bezeichnete Technik zur Sound-Generierung. Sie verwendet zwei elektronisch erzeugte Soundquellen (sog. *Grains*). Die variablen Widerstände steuern die Frequenz und das Abschwellen der Grains (Eingänge 0 und 2 für den einen Grain sowie 3 und 1 für den anderen). Eingang 4 regelt die Synchronisation zwischen den beiden Grains.

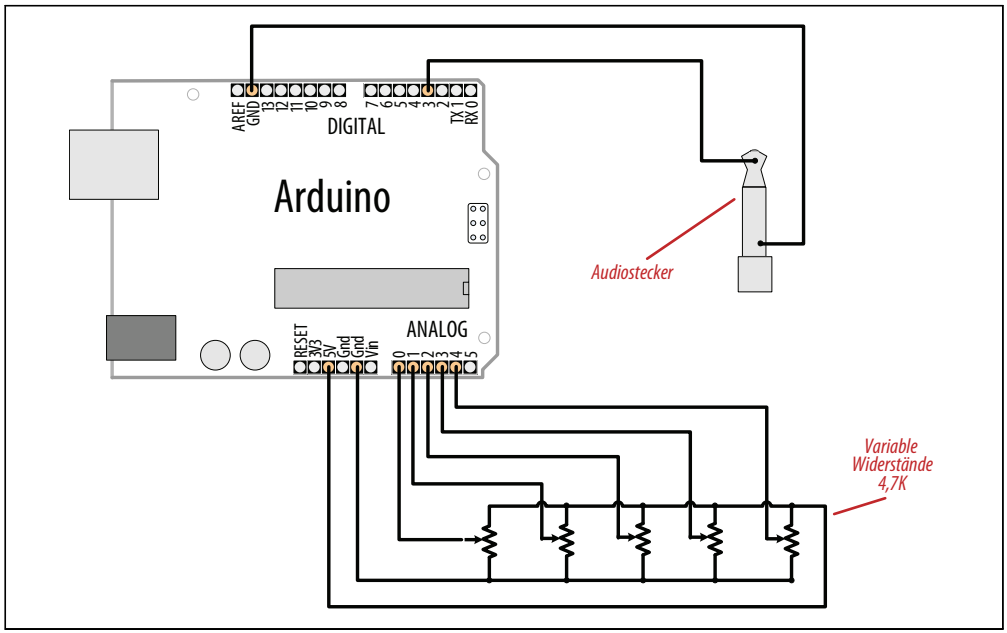


Abbildung 9-6: Auduino

Wenn Sie den Code anpassen wollen, können Sie die Skala verändern, die zur Berechnung der Frequenzen verwendet wird. Voreingestellt ist die pentatonische Skala, doch Sie können das Auskommentieren und eine andere Option aktivieren, die eine andere Skala verwendet.

Seien Sie vorsichtig, wenn Sie zusätzlichen Code in die Hauptschleife einfügen. Der Sketch ist stark optimiert und zusätzlicher Code könnte ihn so stark verlangsamen, dass die Audio-Synthese nicht mehr gut funktioniert.

Sie können jeden der Potis durch Sensoren ersetzen, die ein analoges Signal erzeugen (siehe Kapitel 6). Beispielsweise könnte ein lichtempfindlicher Widerstand (siehe Rezept 6.2) oder ein Entfernungsmesser (der gegen Ende von Rezept 6.4 beschrieben wurde) an einem der Frequenzeingänge (Pin 0 oder 3) die Tonlage steuern, je nachdem, wie weit Ihre Hand vom Sensor entfernt ist (schauen Sie bei Wikipedia oder Google nach »Theremin«, wenn Sie etwas über das Musikinstrument erfahren wollen, das über Handbewegungen gespielt wird).

Siehe auch

Video-Demonstration des Auduino: *http://www.vimeo.com/2266458*

Wikipedia-Artikel zur Granularsynthese: *http://de.wikipedia.org/wiki/Granularsynthese*

Wikipedia-Artikel zum Theremin: *http://de.wikipedia.org/wiki/Theremin*

KAPITEL 10

Externe Geräte fernsteuern

10.0 Einführung

Der Arduino kann mit nahezu jedem Gerät interagieren, das irgendeine Form von Fernbedienung verwendet. Dazu gehören Fernseher, Audioanlagen, Kameras, Garagentore, Haushaltsgeräte und Spielzeug. Die meisten Fernbedienungen senden digitale Daten mittels Infrarot oder Funk von einem Sender (Transmitter) an einen Empfänger (Receiver). Unterschiedliche Protokolle (Signalmuster) werden genutzt, um einen Tastendruck in ein digitales Signal umzuwandeln. Die Rezepte in diesem Kapitel zeigen, wie man weitverbreitete Fernbedienungen und deren Protokolle nutzt.

Eine IR-Fernbedienung schaltet eine LED in bestimmten Mustern an und aus, um eindeutige Codes zu erzeugen. Diese Codes umfassen typischerweise 12 bis 32 Bit. Jede Taste auf der Fernbedienung ist mit einem bestimmten Code verknüpft, der gesendet wird, wenn man diese Taste drückt. Wird die Taste gedrückt, sendet die Fernbedienung üblicherweise wiederholt den gleichen Code, auch wenn einige Fernbedienungen (z.B. NEC) einen speziellen Wiederholungscode senden. Bei RC-5- oder RC-6-Fernbedienungen von Philips wird bei jedem Tastendruck ein Bit im Code umgeschaltet. Der Empfänger nutzt dieses Bit, um zu erkennen, ob eine Taste ein zweites Mal gedrückt wurde. Mehr über die Technik von IR-Fernbedienungen erfahren Sie auf *http://www.sbprojects.com/knowledge/ir/ir.htm*.

Die hier vorgestellten Rezepte verwenden ein kostengünstiges IR-Empfängermodul zur Erkennung des Signals, das eine digitale Ausgabe erzeugt, die der Arduino lesen kann. Diese digitale Ausgabe wird dann von einer Bibliothek namens IRremote verarbeitet. Sie wurde von Ken Shirriff entwickelt und kann von *http://www.arcfn.com/2009/08/multi-protocol-infrared-remote-library.html* heruntergeladen werden.

Die gleiche Bibliothek wird auch in den Rezepten genutzt, bei denen der Arduino als Fernbedienung fungiert und Befehle sendet.

Installieren Sie die Bibliothek im *libraries*-Ordner in Ihrem Arduino-Sketch-Ordner. Hilfe bei der Installation von Bibliotheken finden Sie in .

Funktechnik nutzende Fernbedienungen sind schwieriger zu emulieren als IR-Fernbedienungen. Die Kontakte dieser Fernbedienungen lassen sich aber über den Arduino akti-

vieren. Die Rezepte für Funk-Fernbedienungen simulieren Tastendrücke, indem Sie die entsprechenden Kontakte in der Fernbedienung schließen. Bei Funk-Fernbedienungen müssen Sie die Fernbedienung möglicherweise auseinandernehmen und Drähte von den Kontakten mit dem Arduino verbinden, um sie nutzen zu können. Als *Optokoppler* bezeichnete Bauelemente werden genutzt, um eine elektrische Trennung zwischen dem Arduino und der Fernbedienung zu erreichen. Diese Trennung verhindert, dass Strom vom Arduino die Fernbedienung beschädigt (und umgekehrt).

Optokoppler ermöglichen dem Arduino die sichere Steuerung eines anderen Schaltkreises, der mit anderen Spannungen betrieben wird. Wie es der Name andeutet, ermöglichen Optokoppler eine elektrische Trennung. Diese Bauelemente enthalten eine LED, die über einen Arduino-Digitalpin angesteuert werden kann. Das Licht der LED im Optokoppler ist auf einen lichtempfindlichen Transistor gerichtet. Wird die LED eingeschaltet, leitet der Transistor und schließt den Kreis zwischen seinen beiden Anschlüssen – so, als würde man eine Taste drücken.

10.1 Auf eine Infrarot-Fernbedienung reagieren

Problem

Sie wollen auf Tasten reagieren, die an einer Fernbedienung gedrückt wurden.

Lösung

Arduino kann die IR-Signale einer Fernbedienung über ein *IR-Empfangsmodul* (IR-Receiver) verarbeiten. Gängige Bauteile sind das TSOP4838, PNA4602 und TSOP2438. Die ersten beiden haben identische Anschlüsse, d.h., die Schaltung ist gleich. Beim TSOP2438 sind die +5V- und Masseanschlüsse vertauscht. Stellen Sie mit Hilfe des Datenblattes sicher, dass Sie ihr Bauelement richtig anschließen.

Dieses Rezept nutzt die IRremote-Bibliothek von *http://www.arcfn.com/2009/08/multi-protocol-infrared-remote-library.html*. Schließen Sie den IR-Empfänger Ihrem Datenblatt entsprechend an. Die Anschlüsse in Abbildung 10-1 sind für den TSOP4838/PNA4602 gedacht.

Der folgende Sketch schaltet eine LED um, wenn eine Taste an der Infrarot-Fernbedienung gedrückt wird:

```
/*
 IR_remote_detector Sketch
 Ein IR-Empfänger ist mit Pin 2 verbunden.
 Die LED an Pin 13 wird bei jedem Tastendruck an der Fernbedienung ein- und ausgeschaltet.
*/

#include <IRremote.h>            //Bibliothek einbinden

const int irReceiverPin = 2;       // Pin für Empfänger
const int ledPin = 13;
```

```
IRrecv irrecv(irReceiverPin);       // IRrecv-Objekt erzeugen
decode_results decodedSignal;       // Dekodierte Daten vom IR-Detektor
void setup()
{
 pinMode(ledPin, OUTPUT);
 irrecv.enableIRIn();           // Receiver-Objekt starten
}

boolean lightState = false;   // Zustand der LED nachhalten
unsigned long last = millis(); // Festhalten, wann die letzte IR-Nachricht empfangen wurde

void loop()
{
 if (irrecv.decode(&decodedSignal) == true) // Wahr, wenn eine Nachricht
                                            // empfangen wurde
 {
  if (millis() - last > 250) {    // Ist seit der letzten Nachricht 1/4 Sekunde vergangen?
   lightState = !lightState;        //Ja: LED umschalten
   digitalWrite(ledPin, lightState);
  }
  last = millis();
  irrecv.resume();              // Auf nächste Nachricht warten
 }
}
```

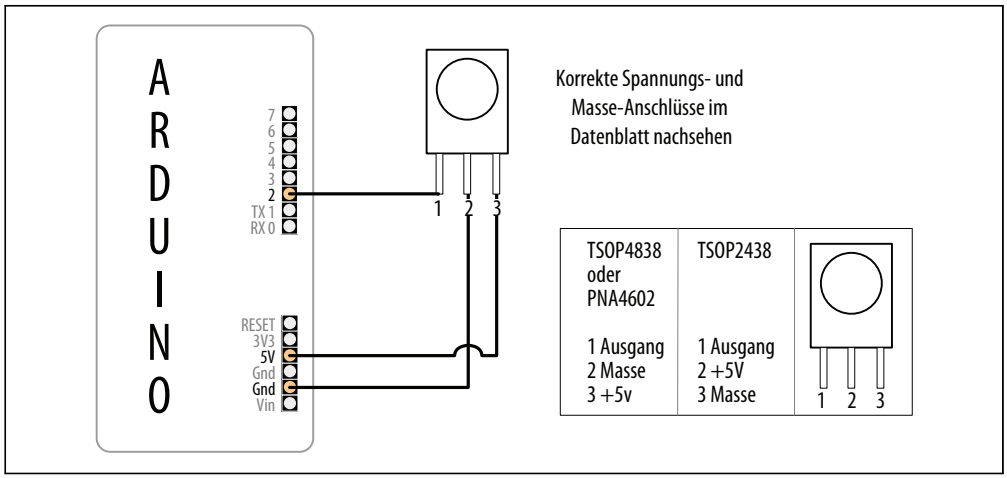


Abbildung 10-1: Anschluss eine IR-Empfangsmoduls

Diskussion

Der IR-Receiver wandelt das IR-Signal in digitale Impulse um, die eine Folge von Einsen und Nullen bilden, die für die Tasten der Fernbedienung stehen. Die IRremote-Bibliothek dekodiert diese Impulse und liefert einen numerischen Wert für jede Taste zurück. (Die Werte selbst hängen von der verwendeten Fernbedienung ab).

Das `#include <IRremote.h>` zu Beginn des Sketches stellt den Bibliothekscode innerhalb des Sketches zur Verfügung und die Zeile `IRrecv irrecv(irReceiverPin);` erzeugt ein

IRrecv-Objekt namens irrecv, um die Signale vom IR-Receiver am irReceiverPin (hier Pin 2) zu empfangen. In erfahren Sie mehr über die Verwendung von Bibliotheken.

Sie können das irrecv-Objekt nutzen, um auf das Signal des IR-Empfängers zuzugreifen. Sie können ihm Befehle geben, die nach Signalen Ausschau halten und sie dekodieren. Die dekodierten Antworten der Bibliothek werden in einer Variablen namens decode_results gespeichert. Das receiver-Objekt wird im setup mit der Zeile irrecv.enableIRIn(); gestartet. Die Ergebnisse werden in loop mit Hilfe der Funktion irrecv.decode(&decoded-Signal) überprüft.

Die decode-Funktion gibt true zurück, wenn Daten vorhanden sind, die dann in der Variablen decodedSignal gespeichert werden. Rezept 2.11 erklärt, wie das &-Symbol bei Funktionsaufrufen verwendet wird, bei denen die Parameter so modifiziert werden, dass Informationen zurückgegeben werden können.

Wurde eine Nachricht von der Fernbedienung empfangen, schaltet der Code die LED um, wenn seit dem letzten Wechsel mehr als eine viertel Sekunde vergangen ist. Anderenfalls würde die LED sehr schnell ein- und ausgeschaltet werden, wenn die Fernbedienung die Codes bei einem Tastendruck wiederholt sendet, und das Ganze würde eher willkürlich wirken.

Die Variable decodedSignal enthält einen Wert, der mit einer Taste verknüpft ist. Dieser Wert wird in diesem Rezept ignoriert (im nächsten aber verwendet), aber Sie können den Wert ausgeben, indem Sie die im folgenden Code hervorgehobene Serial.println-Zeile in Ihren Code einfügen:

```
if (irrecv.decode(&decodedSignal) == true) // Wahr, wenn eine Nachricht
                                           // empfangen wurde
{
  Serial.println(results.value); // Dekodiertes Ergebnis ausgeben
```

Die Bibliothek muss angewiesen werden, weiterhin auf Signale zu warten, was mit der Zeile irrecv.resume(); erreicht wird.

Dieser Sketch schaltet eine LED ein und aus, wenn eine Taste an der Fernbedienung gedrückt wird, aber Sie können andere Dinge steuern – zum Beispiel können Sie einen Servomotor nutzen, um eine Lampe zu dimmen (mehr zur Steuerung physischer Geräte finden Sie in Kapitel 8).

10.2 IR-Signale einer Fernbedienung dekodieren

Problem

Sie wollen eine bestimmte Taste erkennen, die auf einer Fernbedienung gedrückt wurde.

Lösung

Der folgende Sketch kontrolliert die Helligkeit einer LED über die Tasten einer Fernbedienung. Der Code fordert beim Start die Tasten 0 bis 4 der Fernbedienung an. Die ent-

sprechenden Codes werden im Arduino-Speicher (RAM) festgehalten. Der Sketch reagiert auf diese Tasten, indem er die Helligkeit der LED in Abhängigkeit von der gedrückten Taste einstellt. Die Taste 0 schaltet die LED aus und 1 bis 4 erhöhen die Helligkeit:

```
/*
 RemoteDecode Sketch
 IR-Signale werden dekodiert, um die Helligkeit einer LED zu steuern
 Die Werte der Tasten 0 bis 4 werden zu Beginn des Sketches abgefragt und gespeichert
 Die Taste 0 schaltet die LED aus, die Helligkeit wird mit den Tasten 1 bis 4 schrittweise erhöht
 */

#include <IRremote.h>       // Bibliothek einbinden

const int irReceivePin = 2;    // Pin für IR-Empfänger
const int ledPin     = 9;    // LED an PWM-Pin

const int numberOfKeys = 5;    //  5 Tasten werden gelernt(0 bis 4)
long irKeyCodes[numberOfKeys]; // Codes für die jeweiligen Tasten

IRrecv irrecv(irReceivePin);   // IR-Objekt erzeugen
decode_results results;      // IR-Daten stehen hier

void setup()
{
 Serial.begin(9600);
 pinMode(irReceivePin, INPUT);
 pinMode(ledPin, OUTPUT);
 irrecv.enableIRIn();         // IR-Empfänger starten
 learnKeycodes();           // Tastencodes der Fernbedienung lernen
 Serial.println("Druecken Sie eine Taste der Fernbedienung");
}

void loop()
{
 long key;
 int brightness;

 if (irrecv.decode(&results))
 {
  // Daten wurden empfangen
  irrecv.resume();
  key = convertCodeToKey(results.value);
  if(key >= 0)
  {
   Serial.print("Taste empfangen: ");
   Serial.println(key);
   brightness = map(key, 0,numberOfKeys-1, 0, 255);
   analogWrite(ledPin, brightness);
  }
 }
}

/*
 * Codes der Fernbedienung lernen
 */
void learnKeycodes()
```

```
{
 while(irrecv.decode(&results))  // Puffer leeren
  irrecv.resume();

 Serial.println("IR-Codes lernen...");
 long prevValue = -1;
 int i=0;
 while( i < numberOfKeys)
 {
  Serial.print("Folgende Taste auf der Fernbedienung druecken: ");
  Serial.print(i);
  while(true)
  {
   if( irrecv.decode(&results) )
   {
     if(results.value != -1 && results.value != prevValue)
     {
      showReceivedData();
      irKeyCodes[i] = results.value;
      i = i + 1;
      prevValue = results.value;
      irrecv.resume(); // Nächsten Wert empfangen
      break;
     }
    irrecv.resume(); // Nächsten Wert empfangen
   }
  }
 }
 Serial.println("Lernen abgeschlossen...");
}

/*
 * IR-Code in logischen Tastencode umwandeln
 * (oder -1, wenn keine Ziffer empfangen wurde)
 */
int convertCodeToKey(long code)
{
 for( int i=0; i < numberOfKeys; i++)
 {
  if( code == irKeyCodes[i])
  {
   return i; // Gefundene Taste zurückgeben
  }
 }
 return -1;
}

/*
 * Protokoll-Typ und Wert ausgeben
 */
void showReceivedData()
{
 if (results.decode_type == UNKNOWN)
 {
  Serial.println("- Nachricht konnte nicht dekodiert werden");
 }
 else
```

```
{
 if (results.decode_type == NEC) {
  Serial.print("- NEC dekodiert: ");
 }
 else if (results.decode_type == SONY) {
  Serial.print("- SONY dekodiert: ");
 }
 else if (results.decode_type == RC5) {
  Serial.print("- RC5 dekodiert: ");
 }
 else if (results.decode_type == RC6) {
  Serial.print("- RC6 dekodiert: ");
 }
 Serial.print("Hexwert = ");
 Serial.println( results.value, HEX);
}
}
```

Diskussion

Die Lösung basiert auf der IRremote-Bibliothek. Details finden Sie in der Einführung zu diesem Kapitel.

Der Sketch startet die IR-Bibliothek mit dem folgenden Code:

```
irrecv.enableIRIn(); // IR-Empfänger starten
```

Er ruft dann die Funktion `learnKeyCodes` auf, um den Benutzer aufzufordern, die Tasten 0 bis 4 der Fernbedienung zu drücken. Der Code jeder Taste wird im Array `irKeyCodes` festgehalten. Nachdem alle Taste erkannt und gespeichert wurden, wartet der `loop`-Code auf einen Tastendruck und überprüft, ob es sich dabei um eine der Ziffern aus dem `irKeyCodes`-Array handelt. Ist das der Fall, wird der Wert zur Steuerung der Helligkeit der LED mittels `analogWrite` genutzt.

In Rezept 5.7 erfahren Sie mehr über die Verwendung von `map` und `analogWrite` zur Steuerung der Helligkeit einer LED.

Die Bibliothek sollte mit den meisten IR-Fernbedienungen zurechtkommen. Sie kann die Timings erkennen und speichern und bei Bedarf wiedergeben.

Sie können die Tastencodes auch fest vorgeben, damit sie im Sketch nicht immer wieder neu erlernt werden müssen. Ersetzen Sie die Deklaration von `irKeyCodes` durch die folgenden Zeilen, um die Werte für die jeweiligen Tasten festzulegen. Ändern Sie die Werte so ab, dass sie zu Ihrer Fernbedienung passen. Die Codes erscheinen im seriellen Monitor, wenn die Tasten in der `learnKeyCodes`-Funktion gedrückt werden):

```
long irKeyCodes[numberOfKeys] = {
 0x18E758A7, //0-Taste
 0x18E708F7, //1-Taste
 0x18E78877, //2-Taste
 0x18E748B7, //3-Taste
 0x18E7C837, //4-Taste
 };
```

Siehe auch

Rezept 18.1 erklärt, wie man erlernte Daten im EEPROM (nichtflüchtiger Speicher) speichern kann.

10.3 IR-Signale imitieren

Problem

Sie wollen mit dem Arduino einen Fernseher oder ein anderes fernbedientes Gerät steuern, indem Sie entsprechende IR-Signale emulieren. Das ist das Gegenstück zu Rezept 10.2 – wir senden Befehle, statt sie zu empfangen.

Lösung

Der Sketch verwendet die Fernbedienungs-Codes aus Rezept 10.2, um das Gerät zu steuern. Fünf Taster wählen und senden einen von fünf Codes. Die Verschaltung ist in Abbildung 10-2 zu sehen:

```
/*
 irSend Sketch
 Der Code benötigt eine IR-LED an Pin 3
 und 5 Taster an den Pins 4-8
*/

#include <IRremote.h>    // IR-Bibliothek

const int numberOfKeys = 5;
const int firstKey = 4;  // Der erste Pin der 5 hintereinander
                         // angeschlossenen Taster
boolean buttonState[numberOfKeys];
boolean lastButtonState[numberOfKeys];
long irKeyCodes[numberOfKeys] = {
 0x18E758A7, //0 key
 0x18E708F7, //1 key
 0x18E78877, //2 key
 0x18E748B7, //3 key
 0x18E7C837, //4 key
};


IRsend irsend;

void setup()
{
 for (int i = 0; i < numberOfKeys; i++){
  buttonState[i]=true;
  lastButtonState[i]=true;
  int physicalPin=i + firstKey;
  pinMode(physicalPin, INPUT);
  digitalWrite(physicalPin, HIGH); // Pullups einschalten
 }
```

```
 Serial.begin(9600);
}

void loop() {
 for (int keyNumber=0; keyNumber<numberOfKeys; keyNumber++)
 {
  int physicalPinToRead=keyNumber+4;
  buttonState[keyNumber] = digitalRead(physicalPinToRead);
  if (buttonState[keyNumber] != lastButtonState[keyNumber])
  {
   if (buttonState[keyNumber] == LOW)
   {
    irsend.sendSony(irKeyCodes[keyNumber], 32);
    Serial.println("Sending");
   }
   lastButtonState[keyNumber] = buttonState[keyNumber];
  }
 }
}
```

Man kann nichts sehen, wenn die Codes gesendet werden, da das Licht einer Infrarot-LED mit bloßem Auge nicht zu erkennen ist.

Sie können aber mit einer Digitalkamera prüfen, ob die Infrarot-LED funktioniert – Sie sollten sie in der LCD-Anzeige der Kamera aufleuchten sehen.

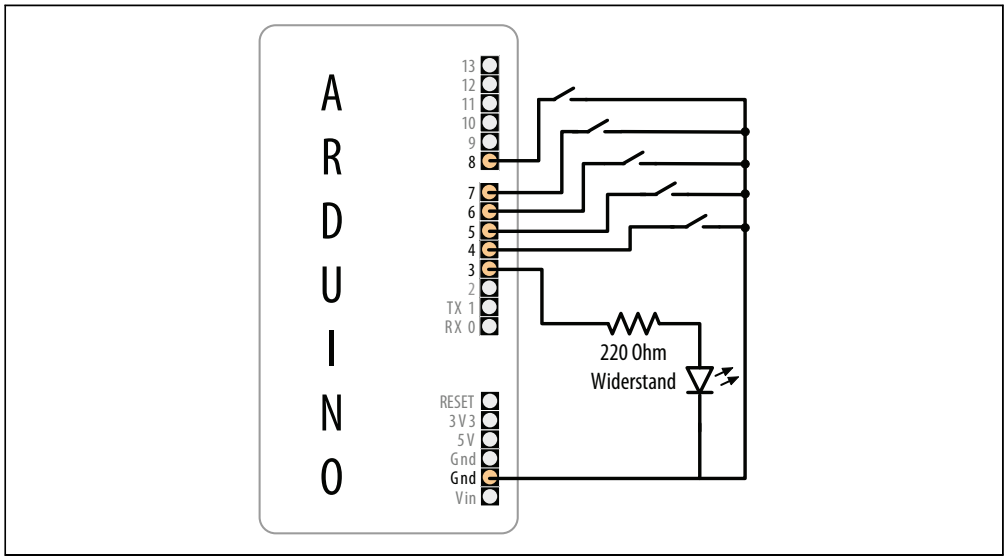


Abbildung 10-2: Taster und LED für IR-Sender

Diskussion

Hier steuert der Arduino ein Gerät, indem er eine IR-LED so blinken lässt, dass sie die Signale einer Fernbedienung imitiert. Dazu wird eine IR-LED benötigt, deren genaue

Spezifikation aber nicht weiter wichtig ist. Geeignete Komponenten finden Sie in Anhang A.

Die IR-Bibliothek übernimmt die Umwandlung numerischer Codes in das Blinken der IR-LED. Sie müssen ein Objekt zum Senden von IR-Nachrichten erzeugen. Die folgende Zeile erzeugt ein `IRsend`-Objekt, das die LED an Pin 3 steuert (Sie können den Pin nicht angeben, er ist in der Bibliothek fest vorgegeben):

```
IRsend irsend;
```

Der Code verwendet ein Array (siehe Rezept 2.4) namens `irKeyCodes`, um die Werte zu speichern, die gesendet werden sollen. Er prüft, ob einer von fünf Tastern gedrückt wurde und sendet den entsprechenden Code mit der folgenden Zeile:

```
irsend.sendSony(irKeyCodes[keyNumber], 32);
```

Das `irSend`-Objekt besitzt unterschiedliche Funktionen für verschiedene weitverbreitete Infrarot-Codeformate. Sehen Sie sich also die Dokumentation der Bibliothek an, wenn Sie ein anderes Format benötigen. Sie können Rezept 10.2 verwenden, wenn Sie sich das von Ihrer Fernbedienung verwendete Format ansehen wollen.

Der Sketch übergibt den Code aus dem Array und die darauffolgende Zahl gibt an, wie viele Bits der Wert besitzt. Das 0x vor den Zahlen der `irKeyCodes` bedeutet, dass es sich um hexadezimale Codes handelt (Details zu Hexadezimalzahlen finden Sie in Kapitel 2). Jedes Hex-Zeichen steht für einen 4-Bit-Wert. Die Codes nutzen acht Zeichen, sind also 32 Bit lang.

Die LED ist mit einem strombegrenzenden Widerstand verbunden (siehe die Einführung zu Kapitel 7).

Wenn Sie den Sendebereich erhöhen wollen, können Sie mehrere LEDs (oder eine stärkere) verwenden.

Siehe auch

Kapitel 7 enthält weiterführende Informationen zur Steuerung von LEDs.

Mitch Altmans TV-B-Gone ist eine clevere Fernbedienungs-Anwendung. Eine Bauanleitung finden Sie auf *http://www.ladyada.net/make/tvbgone/*.

10.4 Eine Digitalkamera steuern

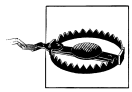
Problem

Sie wollen mit dem Arduino eine Digitalkamera steuern und aus einem Programm heraus Fotos aufnehmen. Sie könnten beispielsweise Zeitraffer-Aufnahmen über den Arduino steuern.

Lösung

Es gibt verschiedene Lösungen. Wenn Ihre Kamera eine Infrarot-Fernbedienung besitzt, können Sie Rezept 10.2 verwenden, um die relevanten IR-Codes zu lernen und Rezept 10.3, um den Arduino diese Codes an die Kamera senden zu lassen.

Wenn Ihre Kamera keine Infrarot-Fernbedienung besitzt, aber einen Anschluss für eine kabelgebundene Fernbedienung hat, können Sie dieses Rezept nutzen, um die Kamera zu steuern.



Der *Klinkenstecker* (engl. TRS) für die Kamera ist typischerweise 2,5 mm oder 3,5 mm groß, Länge und Form entsprechen aber möglicherweise keinem Standard. Die sicherste Möglichkeit, sich den richtigen Stecker zu beschaffen, ist der Kauf eines einfachen Kabels für Ihre Kamera, das Sie dann modifizieren, oder der Erwerb eines Adapterkabels von einem spezialisierten Anbieter (googeln Sie nach »Kamera TRS«).

Sie verbinden den Arduino über Optokoppler mit einem geeigneten Kabel, wie in Abbildung 10-3 zu sehen.

Der folgende Sketch nimmt 20 Bilder auf:

```
/*
 camera Sketch
 Nimmt 20 Bilder mit einer Digitalkamera auf
 Pin 4 steuert den Fokus
 Pin 3 steuert den Verschluss
*/

int focus = 4;              // Optokoppler für Fokus
int shutter = 3;            // Optokoppler für Verschluss
long exposure = 250;        // Belichtungsdauer in Millisekunden
long interval = 10000;      // Zeit in Millisekunden zwischen den Aufnahmen

void setup()
{
 pinMode(focus, OUTPUT);
 pinMode(shutter, OUTPUT);
 for (int i=0; i<20; i++)  // Kamera nimmt 20 Bilder auf
 {
  takePicture(exposure);    // Bild schießen
  delay(interval);          // Bis zum nächsten Bild warten
 }
}


void loop()
{
          // Nach 20 Bildern sind wir fertig,
          // weshalb die Schleife leer ist.
          // loop muss hier aber trotzdem stehen,
          // da der Sketch sonst nicht kompiliert wird
}
```

```
void takePicture(long exposureTime)
{
 int wakeup = 10;     // Kamera braucht etwas Zeit, um aufzuwachen und zur Fokussierung
                      // Passen Sie das an Ihre Kamera an

 digitalWrite(focus, HIGH);          // Kamera und Fokus aufwachen lassen
 delay(wakeup);                      // Aufwachen und Fokussierung abwarten
 digitalWrite(shutter, HIGH);        // Verschluss öffnen
 delay(exposureTime);                // Belichtungsdauer abwarten
 digitalWrite(shutter, LOW);         // Verschluss freigeben
 digitalWrite(focus, LOW);           // Fokus freigeben
}
```

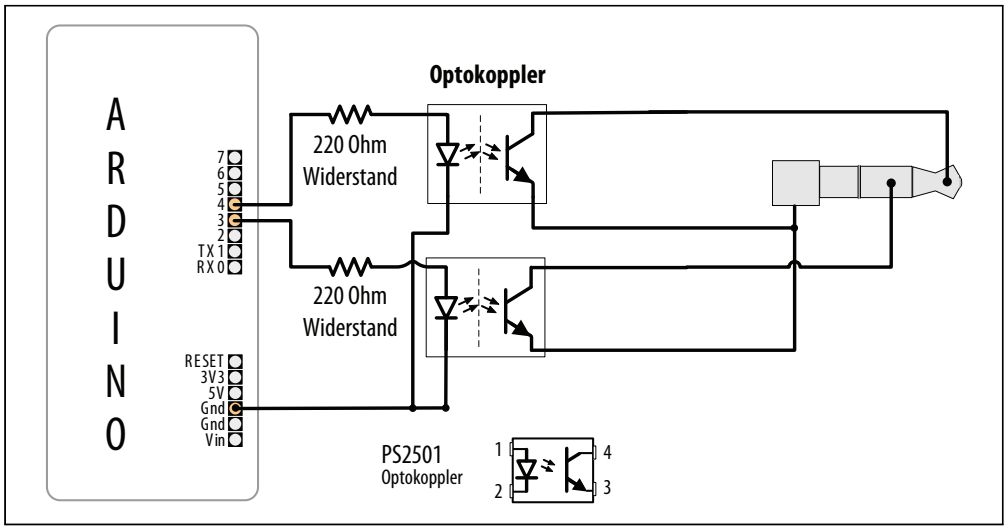


Abbildung 10-3: Optokoppler an Kamera-Klinkenstecker

Diskussion

Es ist nicht ratsam, die Arduino-Pins direkt mit der Kamera zu verbinden – die Spannungen sind möglicherweise nicht kompatibel und Sie riskieren die Beschädigung Ihres Arduino oder der Kamera. Optokoppler werden genutzt, um Arduino und Kamera zu trennen. Mehr über diese Bauelemente erfahren Sie in der Einführung zu diesem Kapitel.

Sie müssen im Benutzerhandbuch der Kamera den richtigen Klinkenstecker heraussuchen.

Sie müssen möglicherweise die Reihenfolge ändern, in der die Pins in der `takePicture`-Funktion ein- und ausgeschaltet werden, um das gewünschte Verhalten zu erzielen. Für Langzeitbelichtungen mit einer Canon-Kamera müssen Sie den Fokus einschalten, dann den Verschluss öffnen, ohne den Fokus freizugeben, dann den Verschluss und zum Schluss den Verschluss wieder freigeben (wie im Sketch). Um ein Bild aufzunehmen und die Kamera die Belichtungsdauer berechnen zu lassen, drücken Sie die Fokus-Taste, geben sie wieder frei und drücken dann den Verschluss.

Siehe auch

Wenn Sie Aspekte des Kamerabetriebs steuern wollen, sollten Sie sich das Canon Hack Development Kit auf *http://chdk.wikia.com/wiki/CHDK* ansehen.

Siehe auch *The Canon Camera Hackers Manual: Teach Your Camera New Tricks* von Berthold Daum (Rocky Nook).

Auf ähnliche Weise lassen sich auch Videokameras über LANC steuern. Die Suche nach »LANC« im Arduino Playground liefert die entsprechenden Details.

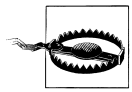
10.5 Wechselstromgeräte über eine gehackte Fernbedienung steuern

Problem

Sie wollen auf sichere Weise Wechselstrom ein- und ausschalten, um Lichter und Geräte über eine Fernbedienung steuern zu können.

Lösung

Arduino kann die Tasten einer Fernbedienung über einen Optokoppler ansteuern. Das kann bei Fernbedienungen notwendig sein, die mit Funk- statt mit Infrarot-Technik arbeiten. Diese Technik kann für nahezu jede Fernsteuerung verwendet werden. Ein Hack der Fernbedienung ist besonders nützlich, um potentiell gefährlichen Wechselstrom von Ihnen und dem Arduino fernzuhalten, da nur die batteriebetriebene Fernbedienung modifiziert wird.



Mit dem Öffnen der Fernbedienung erlischt die Garantie und der Gerät kann dabei beschädigt werden. Die Infrarot-Rezepte in diesem Kapitel sind vorzuziehen, weil eine Modifikation der Fernbedienung unnötig ist.

Wenn Sie dieses Rezept nutzen, die Fernbedienung aber weiterhin nutzen wollen, sollten Sie sich eine Ersatz-Fernbedienung zum Hacken besorgen. Die meisten Hersteller werden Ihnen mit Vergnügen eine Ersatz-Fernbedienung verkaufen (doch Sie müssen auf die richtige Frequenz für das zu steuernde Gerät achten). Sobald Sie diese Fernbedienung haben, müssen Sie möglicherweise noch den von ihr verwendeten Kanal einstellen.

Öffnen Sie die Fernbedienung und schließen Sie den Optokoppler so an, dass der Photo-Emitter (Pins 1 und 2 in Abbildung 10-4) mit dem Arduino und der Photo-Transistor (Pins 3 und 4) mit den Kontakten der Fernbedienung verbunden ist.

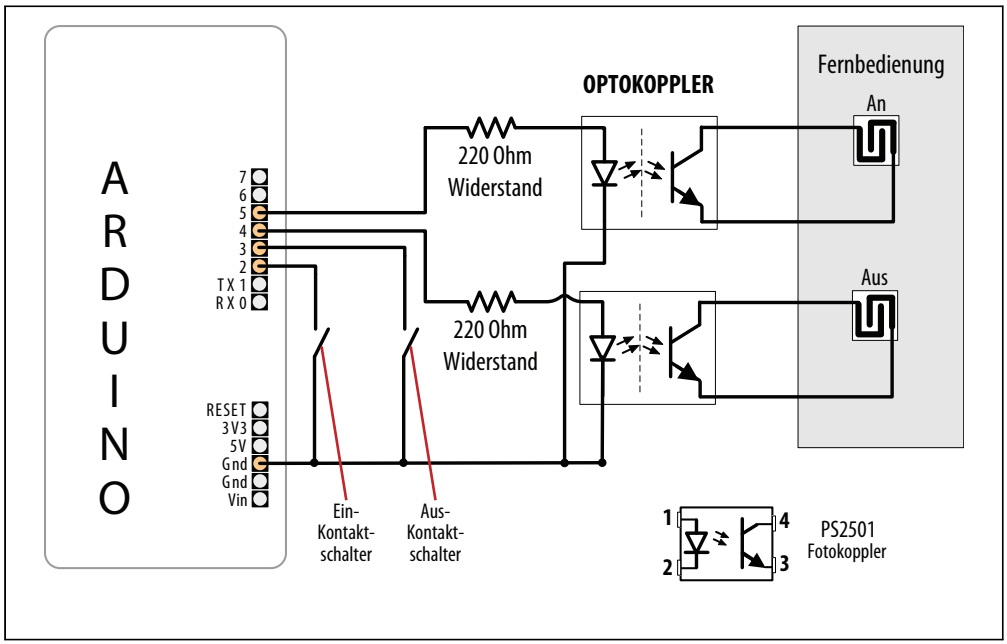


Abbildung 10-4: Anschluss eines Optokoppler an die Kontakte einer Fernbedienung

Der Sketch verwendet Kontaktschalter, um die Tasten der Fernbedienung ein- und auszuschalten:

```
/*
 OptoRemote Sketch
 Taster an den Pins 2 und 3 schalten ferngesteuertes Gerät über Optokoppler ein und aus.

 Die Ausgänge werde bei einem Tastendruck für mindestens eine halbe Sekunde gepulst
 */
const int onSwitchPin  = 2;       // Eingangspin für Ein-Schalter
const int offSwitchPin = 3;       // Eingangspin für Aus-Schalter
const int remoteOnPin  = 4;       // Ausgangspin zum Einschalten der Fernbedienung
const int remoteOffPin = 5;      // Ausgangspin zum Ausschalten der Fernbedienung
const int PUSHED     = LOW;       // Wert bei gedrückter Taste


void setup() {
 Serial.begin(9600);
 pinMode(remoteOnPin, OUTPUT);
 pinMode(remoteOffPin, OUTPUT);
 pinMode(onSwitchPin, INPUT);
 pinMode(offSwitchPin, INPUT);
 digitalWrite(onSwitchPin,HIGH); // Internen Pullup für inputPins aktivieren
 digitalWrite(offSwitchPin,HIGH);
}

void loop(){
 int val = digitalRead(onSwitchPin); // Eingabewert einlesen
 // Einschalten, wenn Taster gedrückt
```

```
 if( val == PUSHED)
 {
  pulseRemote(remoteOnPin);
 }
 val = digitalRead(offSwitchPin);  // Eingabewert einlesen
 // Ausschalten, wenn Taster gedrückt
 if( val == PUSHED)
 {
  pulseRemote(remoteOffPin);
 }
}

// Optokoppler für eine halbe Sekunde einschalten, um das Signal an die Fernbedienung zu
übergeben
void pulseRemote(int pin )
{
 digitalWrite(pin, HIGH);   // Optokoppler einschalten
 delay(500);                // Halbe Sekunde warten
 digitalWrite(pin, LOW);    // Optokopller ausschalten
}
```

Diskussion

Die Taster der meisten Fernbedienungen bestehen aus einfachen Leiterbahnen und einem leitenden Taster, der den Kontakt schließt, wenn er gedrückt wird. Weniger weit verbreitet sind Fernbedienungen mit konventionellen Drucktastenschaltern. Sie sind einfacher zu nutzen, weil die Beinchen der Schalter eine einfache Anschlussmöglichkeit bieten.

Zwar können die Fernbedienungstaste und der Optokoppler gemeinsam verwendet werden – der Schaltvorgang erfolgt unabhängig von der verwendeten Methode (Tastendruck oder Optokoppler) –, aber die mit dem Arduino verbundenen Kabel können das schwierig machen.

Der Transistor des Optokopplers lässt den Strom nur in einer Richtung fließen. Wenn die Sache beim ersten Versuch nicht funktioniert, vertauschen Sie einfach die beiden Anschlüsse und schauen Sie, ob dies das Problem behebt.

Bei einigen Fernbedienungen sind alle Tasten auf einer Seite miteinander verbunden (üblicherweise mit der Masse der Schaltung). Sie können die Leitungen auf dem Board verfolgen, um das zu überprüfen, oder Sie können ein Multimeter nutzen, um den Widerstand der Anschlüsse verschiedener Tasten zu messen. Bei einem gemeinsamen Anschluss muss nur eine Ader für jede Gruppe verwendet werden. Weniger Anschlüsse machen es einfacher, da die Verdrahtung der Adern recht fummelig sein kann, wenn die Fernbedienung klein ist.

Optokoppler werden in Rezept 10.4 erläutert. Sehen Sie sich dieses Rezept an, wenn Sie mit Optokopplern nicht vertraut sind.

Die Fernbedienung kann mehrere Kontakte für jede Taste verwenden. Sie könnten mehrere Optokoppler für jede Taste benötigen, um die Kontakte herzustellen. Abbildung 10-5 zeigt drei Optokoppler, die über einen einzelnen Arduino-Pin angesteuert werden.

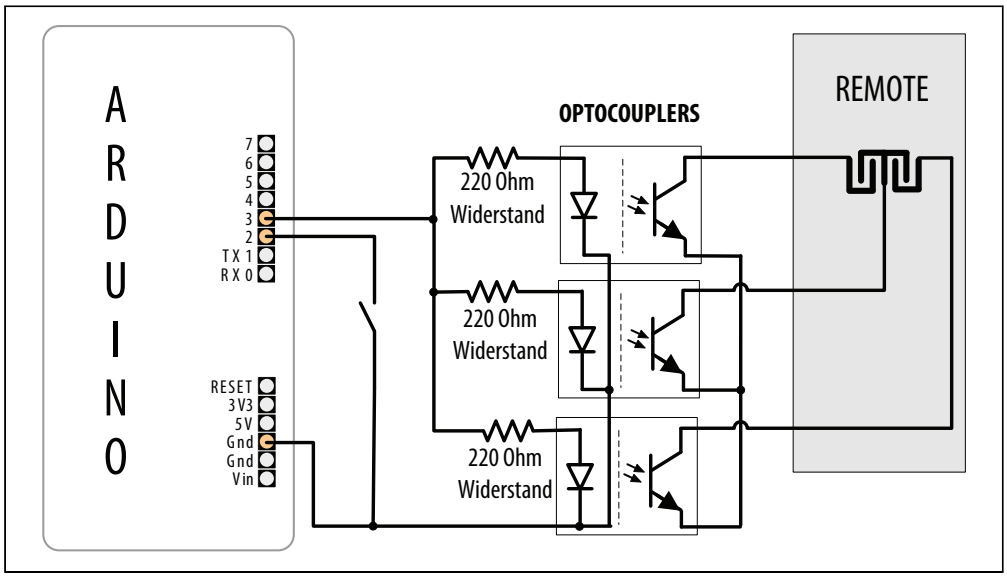


Abbildung 10-5: Mehrere Optokoppler an einer einzelnen Taste der Fernbedienung

Siehe auch

Ein anderer Ansatz zur Steuerung von Wechselströmen ist ein isoliertes Relais wie der PowerTailSwitch, der über Arduino-Pins direkt ein- und ausgeschaltet werden kann. Siehe *http://powerswitchtail.com/default.aspx*.

KAPITEL 11

Displays nutzen

11.0 Einführung

Eine Flüssigkristallanzeige (engl. Liquid Crystal Display, kurz LCD) ist eine einfache und kostengünstige Möglichkeit, Ihr Projekt mit einem Benutzerinterface auszustatten. Dieses Kapitel erklärt, wie man text- und grafikbasierte LCD-Panels mit dem Arduino verbindet und betreibt. Das mit Abstand populärste LCD ist das auf dem Hitachi HD44780 basierende Text-Panel. Es gibt zwei bis vier Zeilen Text mit 16 oder 20 Zeichen pro Zeile aus (es gibt auch Versionen mit 32 und 40 Zeichen, allerdings zu einem wesentlich höheren Preis). Eine Bibliothek zur Ansteuerung textbasierter LC-Displays wird mit dem Arduino mitgeliefert und Sie können Texte auf dem LCD ebenso einfach ausgeben wie über den seriellen Monitor (siehe Kapitel 4), da LCD und serieller Monitor die gleichen zugrundeliegenden `print`-Funktionen verwenden.

LCDs können mehr, als einfach nur Text darstellen. Wörter können gescrollt und hervorgehoben werden und Sie können spezielle Symbole und nicht-englische Zeichen ausgeben.

Sie können eigene Symbole und Blockgrafiken für das Text-LCD entwerfen, doch wenn Sie feine grafische Details benötigen, müssen Sie ein Grafikdisplay verwenden. Grafische LC-Displays (GLCD) kosten nur wenig mehr als Text-Displays und viele beliebte GLCD-Panel können neben Grafik auch bis zu acht Textzeilen mit 20 Zeichen darstellen.

LCDs benötigen mehr Arduino-Anschlüsse als die meisten anderen Rezepte in diesem Buch. Falsche Verbindungen sind das Hauptproblem bei LCDs, also nehmen Sie sich die Zeit für die richtige Verschaltung und überprüfen Sie alles sorgfältig. Ein kostengünstiges Multimeter, das Spannungen und Widerstände messen kann, ist eine große Hilfe, um den korrekten Anschluss zu überprüfen. Es kann einem einige Kopfschmerzen ersparen, wenn nichts auf dem Display erscheint. Sie brauchen nichts besonderes, selbst das einfachste Multimeter hilft Ihnen sicherzustellen, dass die richtigen Pins verbunden und die Spannungen korrekt sind.

Es gibt sogar ein Video-Tutorial und ein PDF, das die Verwendung eines Multimeters beschreibt (siehe *http://blog.makezine.com/archive/2007/01/multimeter_tutorial_make_1.html*).

Für Projekte, die eine größere Anzeige benötigen, als sie günstige LCD-Panels bieten, beschreibt Rezept 11.11, wie man einen Fernseher als Anzeige für den Arduino nutzen kann.

11.1 Ein Text-LCD anschließen und nutzen

Problem

Sie besitzen ein Text-LCD, das auf dem Industriestandard HD44780 (oder einem kompatiblen Controller) basiert, und wollen Text und Zahlen ausgeben.

Lösung

Die Arduino-Software umfasst die LiquidCrystal-Bibliothek zur Steuerung von LC-Displays mit HD44780-Chip.

Die meisten für den Arduino gedachten Text-LCDs sind mit dem Hitachi HD44780-Controller kompatibel. Wenn Sie sich nicht sicher sind, überprüfen Sie anhand des Datenblatts, ob er 44780-kompatibel ist.

Um die Anzeige ans Laufen zu bekommen, müssen Sie die Strom-, Daten- und Steuerpins anschließen. Verbinden Sie die Daten- und Steuerpins mit digitalen Ausgängen, schließen Sie ein Potentiometer für den Kontrast an und schließen Sie die Stromleitungen an. Wenn Ihr Display eine Hintergrundbeleuchtung hat, muss sie ebenfalls angeschlossen werden, üblicherweise über einen Widerstand.

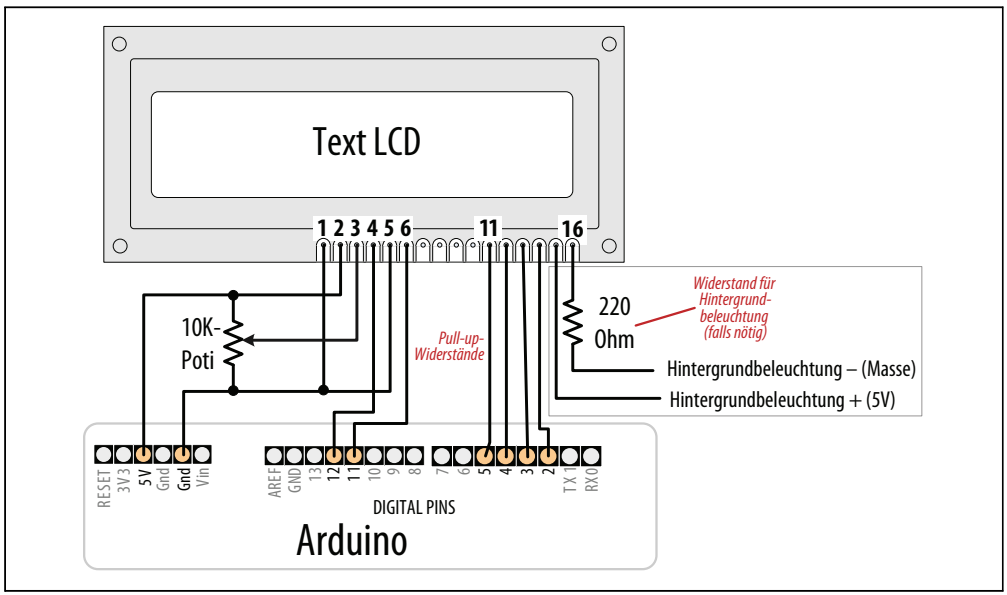


Abbildung 11-1: Anschluss eines Text-LCDs

Abbildung 11-1 zeigt die gängigsten LCD-Anschlüsse. Stellen Sie mit Hilfe des Datenblatts sicher, das Sie die richtigen Pins verwenden. Tabelle 11-1 zeigt die wichtigsten Anschlüsse, doch wenn Ihr LCD andere Pins verwendet, müssen Sie auf Kompatibilität mit dem Hitachi HD44780 achten – dieses Rezept funktioniert nur mit LCDs, die mit diesem Chip kompatibel sind. Das LCD verfügt über 16 Pins (bzw. 14 Pins ohne Hintergrundbeleuchtung) – identifizieren Sie Pin 1 auf dem Panel, da es an einer anderen Stelle liegen könnte, als in der Abbildung zu sehen.

Vielleicht fragen Sie sich, warum die LCD-Pins 7 bis 10 nicht angeschlossen sind. Der Datentransfer des LCDs kann entweder über vier oder über acht Pins erfolgen. Dieses Rezept arbeitet mit vier Pins, damit die anderen vier Arduino-Pins für andere Aufgaben frei bleiben. Theoretisch steigt bei acht Pins die Performance, ist aber nicht so hoch, dass sie den Verlust von vier Arduino-Pins wert wäre.

Tabelle 11-1: LCD-Anschlüsse

LCD-Pin	Funktion	Arduino-Pin
1	Masse: Gnd oder 0V oder Vss	Gnd
2	+5V oder Vdd	5V
3	Vo oder Kontrast	
4	RS	12
5	R/W	Gnd
6	E	11
7	D0	
8	D1	
9	D2	
10	D3	
11	D4	5
12	D5	4
13	D6	3
14	D7	2
15	A oder Anode	
16	K oder Kathode	

Sie müssen ein 10K-Potentiometer für den Kontrast an LCD-Pin 3 anschließen. Ohne die richtige Spannung an diesem Pin sehen Sie auf dem Display möglicherweise gar nichts. In Abbildung 11-1 ist eine Seite des Potis mit Gnd (Masse) verbunden, die andere Seite mit +5V vom Arduino und der Schieber mit LCD-Pin 3. Das LCD wird über Gnd und +5V vom Arduino an den LCD-Pins 1 und 2 mit Strom versorgt.

Viele LCD-Panels besitzen eine interne Lampe, die sog. *Hintergrundbeleuchtung*, um das Display zu erhellen. Auf dem Datenblatt sollte stehen, ob es eine Hintergrundbeleuchtung gibt und ob sie einen externen Widerstand benötigt – was häufig der Fall ist, damit die

Hintergrund-LEDs nicht durchbrennen. Falls Sie sich nicht sicher sind, schließen Sie einfach einen 220 Ohm-Widerstand an. Die Hintergrundbeleuchtung ist gepolt, also achten Sie darauf, dass Pin 15 mit +5V und Pin 16 mit Masse verbunden ist. (Der abgebildete Widerstand ist zwischen Pin 16 und Masse angeschlossen, Sie können ihn aber ebenso gut auch zwischen Pin 15 und +5V hängen.)

Überprüfen Sie die Verschaltung noch einmal, bevor Sie die Spannung einschalten, da Sie das LCD beschädigen können, wenn die Spannungsversorgung falsch angeschlossen ist. Um den mit Arduino gelieferten HelloWorld-Sketch auszuführen, klicken Sie in der IDE das Files-Menü an und navigieren zu Examples→Library→LiquidCrystal→HelloWorld.

Der folgende Code wurde etwas modifiziert, um neben »Hallo, Welt« auch Zahlen auszugeben. Passen Sie `numRows` und `numCols` an die Zeilen und Spalten Ihres LC-Displays an:

```
/*
 LiquidCrystal Library - Hallo, Welt

 Demonstriert die Verwendung eines 16 x 2 LC-Displays.
 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

#include <LiquidCrystal.h> // Bibliothek einbinden

//Konstanten für die Zeilen und Spalten des LCDs
const int numRows = 2;
const int numCols = 16;


// Bibliothek mit den Interface-Pins initialisieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
 lcd.begin(numCols, numRows);
 lcd.print("Hallo, Welt!"); // Meldung im LCD ausgeben.
}

void loop()
{
 // Cursor an Spalte 0, Zeile 1 positionieren
 // (Hinweis: Zeile 1 ist die zweite Zeile, da die Zählung bei 0 beginnt):
 lcd.setCursor(0, 1);
 // Seit Reset verstrichene Sekunden ausgeben.
 lcd.print(millis()/1000);
}
```

Führen Sie den Sketch aus. In der ersten Zeile der Anzeige sollte »Hallo, Welt« erscheinen. In der zweiten Zeile sollte eine Zahl erscheinen, die sich jede Sekunde erhöht.

Diskussion

Wenn kein Text erscheint und Sie ganz sicher sind, dass alle Anschlüsse korrekt sind, müssen Sie möglicherweise den Kontrast mit dem Poti einstellen. Wenn Sie den Poti auf eine Seite drehen (üblicherweise die Seite, die mit Masse verbunden ist), ist der maximale Kontrast eingestellt und Sie sollten kleine Blöcke sehen. Drehen Sie den Poti in die andere Richtung, sehen Sie wahrscheinlich gar nichts mehr. Die richtige Einstellung hängt von vielen Faktoren ab, einschließlich Blickwinkel und Temperatur – drehen Sie am Poti, bis Sie ein gutes Ergebnis erzielen.

Wenn Sie bei keiner Poti-Stellung Pixelblöcke sehen können, überprüfen Sie, ob die Anzeige über die richtigen Pins angesteuert wird.

Sobald Sie Text auf dem Display sehen, ist der Einsatz des LCDs in einem Sketch eine einfache Sache. Sie nutzen ähnliche Befehle wie bei der seriellen Ausgabe, die in Kapitel 4 behandelt wurde. Das nächste Rezept geht genauer auf die print-Befehle ein und erläutert, wie man die Textposition kontrolliert.

Siehe auch

Die LiquidCrystal-Referenz: *http://arduino.cc/en/Reference/LiquidCrystalPrint*.

Kapitel 4 enthält Details zu den `print`-Befehlen.

Das Datenblatt des Hitachi HD44780 LCD-Controllers ist die umfassende Referenz für die Low-Level-Funktionalität. Die Arduino-Bibliothek versteckt einen Großteil der Komplexität vor Ihnen, doch wenn Sie alles über die Fähigkeiten des Chips nachlesen wollen, können Sie sich das Datenblatt von *http://www.sparkfun.com/datasheets/LCD/HD44780.pdf* herunterladen.

Die LCD-Seite im Arduino Playground enthält Tipps zu Soft- und Hardware sowie weiterführende Links: *http://www.arduino.cc/playground/Code/LCD*.

11.2 Text formatieren

Problem

Sie wollen die Position des im LC-Display dargestellten Textes kontrollieren, z.B. um Werte an festgelegten Positionen auszugeben.

Lösung

Der folgende Sketch zählt zuerst von 9 bis 0 herunter und gibt dann eine Ziffernfolge in drei Spalten à vier Zeichen aus. Passen Sie `numRows` und `numCols` an die Anzahl der Zeilen und Spalten Ihres LCDs an:

```
/*
 LiquidCrystal Library - FormatText
*/
```

```
#include <LiquidCrystal.h> // Bibliothek einbinden


//Konstanten für Zeilen und Spalten des LCDs
const int numRows = 2;
const int numCols = 16;

int count;

// Bibliothek mit Interface-Pins initialisieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
 lcd.begin(numCols, numRows);
 lcd.print("Beginne in "); // Dieser String ist 11 Zeichen lang
 for(int i=9; i > 0; i--)     // Von 9 herunterzählen
 {
  // Erste Zeile ist 0
  lcd.setCursor(11,0); // Cursor an das Ende des Strings bewegen
  lcd.print(i);
  delay(1000);
 }
}

void loop()
{
 int columnWidth = 4;        //Spaltenbreite
 int displayColumns = 3;    //Anzahl der Spalten

 lcd.clear();
 for( int col=0; col < displayColumns; col++)
 {
  lcd.setCursor(col * columnWidth, 0);
  count = count+ 1;
  lcd.print(count);
 }
 delay(1000);
}
```

Diskussion

Die `lcd.print`- Funktionen sind `Serial.print` sehr ähnlich. Zusätzlich besitzt die LCD-Bibliothek Befehle zur Steuerung der Cursorposition (Zeile und Spalte, an denen der Text ausgegeben wird).

Die `lcd.print`-Anweisung gibt jedes neue Zeichen hinter dem vorigen aus. Über das Ende einer Zeile hinausgehender Text kann in der nächsten Zeile oder auch gar nicht ausgegeben werden. Mit dem Befehl `lcd.setCursor()` können Sie festlegen, an welcher Position der nächste `lcd.print`-Aufruf mit der Ausgabe beginnt. Sie legen die Spalte und Zeile fest (die obere linke Ecke ist 0,0). Sobald der Cursor positioniert ist, erfolgt die Ausgabe beim nächsten `lcd.print` an diesem Punkt und vorhandener Text wird überschrieben. Der

Sketch in diesem Rezept nutzt das, um eine Reihe von Zahlen an festen Positionen auszugeben.

Zum Beispiel stellt in `setup` die Zeile

```
lcd.setCursor(12,0); // Cursor an die 12. Stelle bewegen
lcd.print(i);
```

`lcd.setCursor(12,0)` sicher, dass jede Ziffer an der gleichen Position (12. Spalte, 1. Zeile) ausgegeben wird. Die Ziffern erscheinen also immer an der gleichen Stelle und werden nicht hintereinander ausgegeben.

Zeilen und Spalten beginnen bei 0, d.h., `setCursor(4,0)` positioniert den Cursor in der fünften Spalte der ersten Zeile. Das liegt daran, dass zwischen den Positionen 0 bis 4 fünf Zeichen liegen. Wenn das nicht klar ist, zählen Sie es einfach (bei 0 beginnend) an den Fingern ab.

Die folgenden Zeilen nutzen `setCursor`, um die Spalten mit `columnwidth` Leerzeichen aufzufüllen:

```
lcd.setCursor(col * columnWidth, 0);
count = count+ 1;
lcd.print(count);
lcd.clear();
```

`lcd.clear` löscht den Bildschirm und bewegt den Cursor zurück in die obere linke Ecke.

Nachfolgend eine `loop`-Variante, die für die Ausgabe alle Zeilen Ihre LCDs nutzt. Ersetzen Sie Ihren `loop`-Code durch folgende Zeilen (und tragen Sie für `numRows` und `numCols` die Zeilen und Spalten Ihres LCDs ein):

```
void loop()
{
int columnWidth = 4;
int displayColumns = 3;

 lcd.clear();
 for(int row=0; row < numRows; row++)
 {
  for( int col=0; col < displayColumns; col++)
  {
   lcd.setCursor(col * columnWidth, row);
   count = count+ 1;
   lcd.print(count);
  }
 }
 delay(1000);
}
```

Die erste `for`-Schleife geht die vorhandenen Zeilen durch, während die zweite die Spalten verarbeitet.

Um die Zahl der in eine LCD-Zeile passenden Zahlen zu berechnen (statt sie einfach fest einzutragen), ändern Sie die `displayColumns`-Zeile:

```
int displayColumns = 3;
```

wie folgt:

```
int displayColumns = numCols / columnWidth;
```

Siehe auch

Das Tutorial zur LiquidCrystal-Bibliothek: *http://arduino.cc/en/Reference/LiquidCrystal?from=Tutorial.LCDLibrary*

11.3 Cursor und Display ein- und ausschalten

Problem

Sie wollen den Cursor blinken lassen und das Display ein- oder ausschalten, z.B. um die Aufmerksamkeit auf einen bestimmten Bereich der Anzeige zu lenken.

Lösung

Dieser Sketch zeigt, wie Sie den Cursor (ein leuchtender Block an der Stelle, an der das nächste Zeichen ausgegeben wird) blinken lassen können. Er zeigt auch, wie man die Anzeige ein- und ausschalten kann, um Aufmerksamkeit zu erregen, indem man das gesamte Display blinken lässt:

```
/*
 blink
 */

// Bibliothek einbinden
#include <LiquidCrystal.h>

// Bibliothek mit den Interface-Pins initialsieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
 // Spalten und Zeilen des LCDs einstellen und
 lcd.begin(16, 2);
 // Eine Meldung auf dem LCD ausgeben.
 lcd.print("Hallo, Welt!");
}

void loop()
{
 lcd.setCursor(0, 1);

 lcd.print("Cursor blinkt");
 lcd.blink();
```

```
 delay(2000);

 lcd.noBlink();
 lcd.print(" noBlink");
 delay(2000);

 lcd.clear();

 lcd.print("Display aus ...");
 delay(1000);
 lcd.noDisplay();
 delay(2000);

 lcd.display(); // Display wieder einschalten

 lcd.setCursor(0, 0);
 lcd.print(" Display-Flash !");
 displayBlink(2, 250); // Zweimal blinken lassen
 displayBlink(2, 500); // Und nochmal, aber doppelt so lang

 lcd.clear();
}

void displayBlink(int blinks, int duration)
{
 while(blinks--)
 {
  lcd.noDisplay();
  delay(duration);
  lcd.display();
  delay(duration);
 }
}
```

Diskussion

Der Sketch ruft die Funktionen `blink` und `noBlink` auf, um das Blinken des Cursors ein- und auszuschalten.

Der Code, der das ganze Display blinken lässt, steht in der Funktion `displayBlink`. Die Funktion verwendet `lcd.display()` und `lcd.noDisplay()`, um die Anzeige ein- und auszuschalten (ohne den Text im internen Speicher zu löschen).

11.4 Text scrollen

Problem

Sie wollen Text scrollen. Zum Beispiel wollen Sie eine Laufschrift erzeugen, die mehr Zeichen darstellt, als in eine Zeile des LC-Displays passen.

Lösung

Dieser Sketch demonstriert `lcd.ScrollDisplayLeft` und `lcd.ScrollDisplayRight`.

Er scrollt eine Textzeile nach links, wenn das System geneigt ist, und nach rechts, wenn er nicht geneigt ist. Verbinden Sie eine Seite eines Neigungssensors mit Pin 7 und den anderen Pin mit Masse (wenn Sie mit Neigungssensoren nicht vertraut sind, sehen Sie sich Rezept 6.1 an):

```
/*
  Scroll
 * Der Sketch scrollt Text nach links, wenn er geneigt ist,
 * und nach rechts, wenn nicht.
 */

#include <LiquidCrystal.h>

// Bibliothek mit den Interface-Pins initialisieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int numRows = 2;
const int numCols = 16;

const int tiltPin = 7; // Pin für Neigungssensor

const char textString[] = "Zum Scrollen neigen";
const int textLen = sizeof(textString) -1; // Zahl der Zeichen
boolean isTilted = false;

void setup()
{
 // Spalten und Zeilen des LCDs festlegen
 lcd.begin(numCols, numRows);
 digitalWrite(tiltPin, HIGH); // Pullups für Neigungssensor aktivieren
 lcd.print(textString);
}

void loop()
{
 if(digitalRead(tiltPin) == LOW && isTilted == false )
 {
  // Geneigt, also Text nach links scrollen
  isTilted = true;
  for (int position = 0; position < textLen; position++)
  {
   lcd.scrollDisplayLeft();
   delay(150);
  }
 }
 if(digitalRead(tiltPin) == HIGH && isTilted == true )
 {
  // Nicht mehr geneigt, also Text nach rechts scrollen
  isTilted = false;
  for (int position = 0; position < textLen; position++)
  {
```

```
    lcd.scrollDisplayRight();
    delay(150);
   }
  }
}
```

Diskussion

Die erste Hälfte des `loop`-Codes behandelt den Übergang vom nicht geneigten in den geneigten Zustand. Der Code prüft, ob der Neigungsschalter geschlossen (`LOW`) oder offen (`HIGH`) ist. Ist er `LOW` und der aktuelle Zustand (der in `isTilted` steht) nicht geneigt, dann wird der Text nach links gescrollt. Die Verzögerung in der `for`-Schleife steuert die Geschwindigkeit des Scrollens. Passen Sie den Wert entsprechend an, wenn sich der Text zu schnell oder zu langsam bewegt.

Die zweite Hälfte des Codes verwendet die gleiche Logik, um den Übergang vom geneigten zum nicht geneigten Zustand zu verarbeiten.

Eine solche Scrolling-Fähigkeit ist besonders nützlich, wenn Sie mehr Text ausgeben müssen, als in eine LCD-Zeile passt.

Der folgende Sketch verwendet die Funktion `marquee` (engl. Laufschrift), die Text mit einer Länge von bis zu 32 Zeichen scrollen kann:

```
/*
 Marquee
 * Scrollt eine sehr lange Textzeile
 */

#include <LiquidCrystal.h>

// Bibliothek mit den Interface-Pins initialisieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int numRows = 2;
const int numCols = 16;


void setup()
{
 // Spalten und Zeilen des LCDs festlegen
 lcd.begin(numCols, numRows);
}

void loop()
{
 marquee("Eine viel zu lange Nachricht!");
 delay(1000);
 lcd.clear();
}

// Diese Version von marquee scrollt sehr lange Nachrichten von Hand
void marquee( char *text)
{
 int length = strlen(text); // Anzahl der Zeichen im Text
```

```
 if(length < numCols)
  lcd.print(text);
 else
 {
  int pos;
  for( pos = 0; pos < numCols; pos++)
    lcd.print(text[pos]);
  delay(1000); // Vor dem Scrollen etwas Zeit zum Lesen lassen
  pos=1;
  while(pos <= length - numCols)
  {
   lcd.setCursor(0,0);
   for( int i=0; i < numCols; i++)
    lcd.print(text[pos+i]);
   delay(300);
   pos = pos + 1;
  }
 }
}
```

Der LCD-Chip besitzt einen internen Speicher, der den Text aufnimmt. Dieser Speicher ist begrenzt ((32 Byte bei den meisten vierzeiligen Displays). Wenn Sie längere Meldungen ausgeben wollen, könnten die sich selbst überschreiben. Wenn Sie längere Nachrichten (z.B. einen Tweet) scrollen oder das Scrollen genauer steuern wollen, benötigen Sie eine andere Technik. Die folgende Funktion speichert den Text im Arduino-RAM und schickt nur Teile an die Anzeige, um den Scroll-Effek zu erzielen. Die Nachrichten können eine beliebige Länge haben, solange sie in den Arduino-Speicher passen:

```
void marquee( char *text)
{
 int length = strlen(text); // Anzahl der Buchstaben im Text
 if(length < numCols)
  lcd.print(text);
 else
 {
  int pos;
  for( pos = 0; pos < numCols; pos++)
    lcd.print(text[pos]);
  delay(1000); // ermöglicht es, dass die erste Zeile vor dem Scrollen gelesen werden kann
  pos=1;
  while(pos <= length - numCols)
  {
   lcd.setCursor(0,0);
   for( int i=0; i < numCols; i++)
    lcd.print(text[pos+i]);
   delay(300);
   pos = pos + 1;
  }
 }
}
```

11.5 Sonderzeichen darstellen

Problem

Sie wollen Sonderzeichen des LCD-Zeichenspeichers wie ° (Grad), ¢, ÷, π (Pi) ausgeben.

Lösung

Ermitteln Sie den Zeichencode des darzustellenden Zeichens in der Zeichenmuster-Tabelle des LCD-Datenblatts. Der folgende Sketch gibt einige gängige Symbole in `setup` aus. Er gibt dann alle darstellbaren Symbole in `loop` aus:

```
/*
 LiquidCrystal Library - Special Chars
*/

#include <LiquidCrystal.h>

//Zeilen und Spalten für Ihr LCD anpassen
const int numRows = 2;
const int numCols = 16;

// Definition einiger nützlicher Sonderzeichen
const byte degreeSymbol = B11011111;
const byte piSymbol   = B11110111;
const byte centsSymbol  = B11101100;
const byte sqrtSymbol  = B11101000;
const byte omegaSymbol  = B11110100;  // Das Symbol für Ohm

byte charCode = 32; // Erstes druckbares ASCII-Zeichen
int col;
int row;

// Bibliothek mit den Interface-Pins initialisieren
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()
{
 lcd.begin(numRows, numCols);

 showSymbol(degreeSymbol, "Grad");
 showSymbol (piSymbol, "Pi");
 showSymbol(centsSymbol, "Cent");
 showSymbol(sqrtSymbol, "Quadrat");
 showSymbol(omegaSymbol, "Ohm");
 lcd.clear();

}

void loop()
{
  lcd.print(charCode);
  calculatePosition();
   if(charCode == 255)
```

```
  {
    // Alle Zeichen ausgegeben, also ein wenig warten
    // und wieder von vorne anfangen
    delay(2000);
    lcd.clear();
    row = col = 0;
    charCode = 32;
  }
  charCode = charCode + 1;
}

void calculatePosition()
{
 col = col + 1;
 if( col == numCols)
 {
   col = 0;
   row = row + 1;
   if( row == numRows)
   {
     row = 0;
     delay(2000); // pause
     lcd.clear();
   }
   lcd.setCursor(col, row);
 }
}

// Sonderzeichen samt Beschreibung ausgeben
void showSymbol( byte symbol, char * description)
{
 lcd.clear();
 lcd.write(symbol);
 lcd.print(' '); // Leerzeichen vor Beschreibung einfügen
 lcd.print(description);
 delay(3000);
}
```

Diskussion

Eine Tabelle mit den verfügbaren Zeichenmustern finden Sie auf dem Datenblatt des LCD-Controller-Chips (auf S. 17 auf dem Datenblatt an *http://www.sparkfun.com/datasheets/LCD/HD44780.pdf*).

Ermitteln Sie das gewünschte Zeichen in der Tabelle. Der Code jedes Zeichens wird durch die Kombination der Binärwerte der Spalte und Zeile bestimmt (siehe Abbildung 11-2).

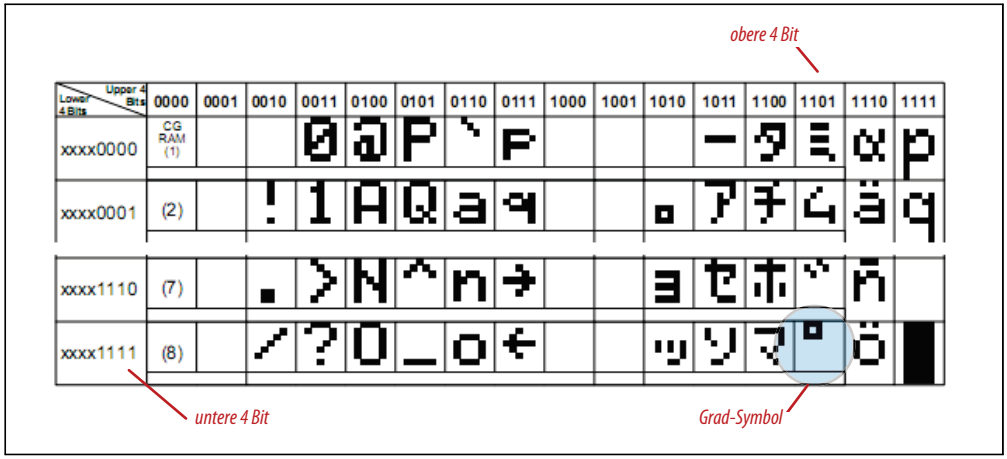


Abbildung 11-2: Zeichencode aus Datenblatt ermitteln

Zum Beispiel ist das Grad-Symbol (°) der drittletzte Eintrag in der unteren Zeile der Tabelle in Abbildung 11-2. Seine Spalte gibt die oberen vier Bits mit 1101 und die Zeile die unteren vier Bits mit 1111 an. Kombiniert man beide, erhält man den Code für das Symbol: B11011111. Sie können diesen Binärwert nutzen oder ihn in einen Hexwert (0xDF) oder Dezimalwert (223) umwandeln. Beachten Sie, dass Abbildung 11-2 nur 4 der insgesamt 16 Zeilen auf dem Datenblatt zeigt.

Die LCD-Anzeige kann natürlich auch jedes druckbare ASCII-Zeichen darstellen. Dazu nutzen Sie den entsprechenden ASCII-Wert in `lcd.print`.

Der Sketch nutzt eine Funktion namens `showSymbol`, um das Sonderzeichen und eine Beschreibung auszugeben:

```
void showSymbol( byte symbol, char * description)
```

(Wenn Sie eine kleine Auffrischung brauchen, wie man Zeichenketten verwendet und an Funktionen übergibt, sehen Sie sich Rezept 2.6 an.)

Siehe auch

Datenblatt zum Hitachi HD44780: *http://www.sparkfun.com/datasheets/LCD/HD44780.pdf*

11.6 Eigene Zeichen definieren

Problem

Sie wollen eigene Zeichen oder Symbole (Glyphen) definieren und anzeigen. Die gewünschten Symbole sind im LCD-Zeichenspeicher nicht vordefiniert.

Lösung

Wenn Sie den folgenden Code hochladen, erscheint abwechselnd ein lächelnder und ein schmollender Smiley:

```
/*
 custom_char Sketch
 Erzeugt ein animiertes Smiley mit selbstdefinierten Zeichen
*/

#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte happy[8] =
{
 B00000,
 B10001,
 B00000,
 B00000,
 B10001,
 B01110,
 B00000,
 B00000
};

byte saddy[8] =
{
 B00000,
 B10001,
 B00000,
 B00000,
 B01110,
 B10001,
 B00000,
 B00000
};

void setup() {
 lcd.createChar(0, happy);
 lcd.createChar(1, saddy);
 lcd.begin(16, 2);

}

void loop() {
 for (int i=0; i<2; i++)
 {
  lcd.setCursor(0,0);
  lcd.write(i);
  delay(500);
 }
}
```

Diskussion

Die LiquidCrystal-Bibliothek ermöglicht die Definition von bis zu acht eigenen Zeichen, die über die Zeichencodes 0 bis 7 ausgegeben werden können. Jedes Zeichen wird auf der Anzeige in einem Raster aus 5×8 Pixeln dargestellt. Sie definieren ein Zeichen in einem Array von acht Bytes. Jedes Byte definiert eine Zeile des Zeichens. Schreibt man sie als Binärzahl, steht eine 1 für ein eingeschaltetes und die 0 für ein ausgeschaltetes Pixel (alle Werte hinter dem fünften Bit werden ignoriert). Der Sketch definiert zwei Zeichen namens `happy` und `saddy` (siehe Abbildung 11-3).

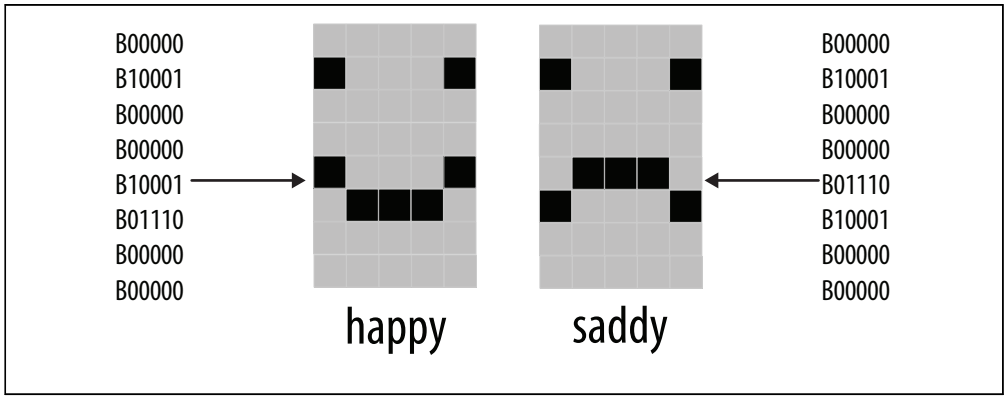


Abbildung 11-3: Definition eigener Zeichen

Die folgende Zeile in `setup` erzeugt das Zeichen aus den im `happy`-Array definierten Werten und weist es dem Zeichen 0 zu:

```
lcd.createChar(0, happy);
```

Um das selbstdefinierte Zeichen auszugeben, nutzen Sie die folgende Zeile:

```
lcd.write(0);
```

Beachten Sie den Unterschied beim Schreiben eines Zeichens mit und ohne Hochkomma. Die folgende Zeile gibt eine 0 aus, nicht das Smiley-Symbol:

```
lcd.write('0'); // gibt eine 0 aus
```

Der Code in der `for`-Schleife schaltet zwischen den Zeichen 0 und 1 hin und her, um eine Animation zu erzeugen.

11.7 Große Symbole darstellen

Problem

Sie wollen zwei oder mehr selbstdefinierte Zeichen kombinieren, um Symbole darzustellen, die größer sind als ein einzelnes Zeichen, z.B. Zahlen doppelter Höhe.

Lösung

Der folgende Sketch gibt Zahlen doppelter Größe über selbstdefinierte Zeichen aus:

```
/*
* customChars
*
* Dieser Sketch gibt große Ziffern aus
* Die bigDigit-Arrays wurden vom Arduino-Forum-Mitglied dcb inspiriert
*/

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

byte glyphs[5][8] = {
 { B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000 },
 { B00000,B00000,B00000,B00000,B00000,B00000,B11111,B11111 },
 { B11111,B11111,B00000,B00000,B00000,B00000,B11111,B11111 },
 { B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111 } ,
 { B00000,B00000,B00000,B00000,B00000,B01110,B01110,B01110 } };

const int digitWidth = 3; // Breite in großen Ziffer in Zeichen
                          // (Ohne Leerzeichen zwischen den Zeichen)

//Arrays zur Indexierung der selbstdefinierten Zeichen, aus denen die großen Ziffern bestehen
// Ziffern 0-4                                  0   1   2   3   4
const char bigDigitsTop[10][digitWidth]={ 3,0,3, 0,3,32, 2,2,3, 0,2,3, 3,1,3,
 // Ziffern 5-9                                  5   6   7   8   9
                                          3,2,2, 3,2,2, 0,0,3,  3,2,3, 3,2,3};

const char bigDigitsBot[10][ digitWidth]={ 3,1,3, 1,3,1,  3,1,1, 1,1,3, 32,32,3,
                                           1,1,3, 3,1,3, 32,32,3, 3,1,3, 1,1,3};

char buffer[12]; // Puffer zur Umwandlung einer Zahl in einen String
void setup ()
{
 lcd.begin(20,4);
 // Selbstdefinierte Zeichen erzeugen
 for(int i=0; i < 5; i++)
  lcd.createChar(i, glyphs[i]);  // 5 eigene Zeichen definieren
 // Countdown ausgeben
 for(int digit = 9; digit >= 0; digit--)
 {
  showDigit(digit, 2); // Ziffer ausgeben
  delay(1000);
 }
 lcd.clear();
}

void loop ()
{
 // Nun ausgeben, wie lange der Sketch läuft (in Sekunden)
 int number = millis() / 1000;
 showNumber( number, 0);
 delay(1000);
}
```

```
void showDigit(int digit, int position)
{
 lcd.setCursor(position * (digitWidth + 1), 0);
 for(int i=0; i < digitWidth; i++)
  lcd.write(bigDigitsTop[digit][i]);
 lcd.setCursor(position * (digitWidth + 1), 1);
 for(int i=0; i < digitWidth; i++)
  lcd.write(bigDigitsBot[digit][i]);
}
void showNumber(int value, int position)
{
 int index; // Index auf die auszugebende Ziffer, 0 ist die Ziffer ganz links
 itoa(value, buffer, 10); // Mehr zu itoa finden Sie in Rezept 2.8
 // Alle Ziffern nacheinander ausgeben
 for(index = 0; index < 10; index++) // Bis zu 10 Ziffern darstellen
 {
  char c = buffer[index];
  if( c == 0)  // Auf Null prüfen (nicht auf '0')
   return; // Das String-Ende-Zeichen ist die Null, siehe Kapitel 2
  c = c - 48; //ASCII-Wert in numerischen Wert umwandeln (siehe Rezept 2.9)
  showDigit(c, position + index);
 }
}
```

Diskussion

Die Zeichen eines LC-Displays haben eine feste Größe, aber man kann durch die Kombination von Zeichen größere Symbole darstellen. Dieses Rezept erzeugt mit der in Rezept 11.6 beschriebenen Technik fünf selbstdefinierte Zeichen. Diese Symbole (siehe Abbildung 11-4) können so kombiniert werden, dass sich mit ihnen große Ziffern (siehe Abbildung 11-5) darstellen lassen. Der Sketch zählt auf dem LCD in großen Ziffern von 9 bis 0 herunter. Er gibt dann in Sekunden aus, wie lange der Sketch schon läuft.

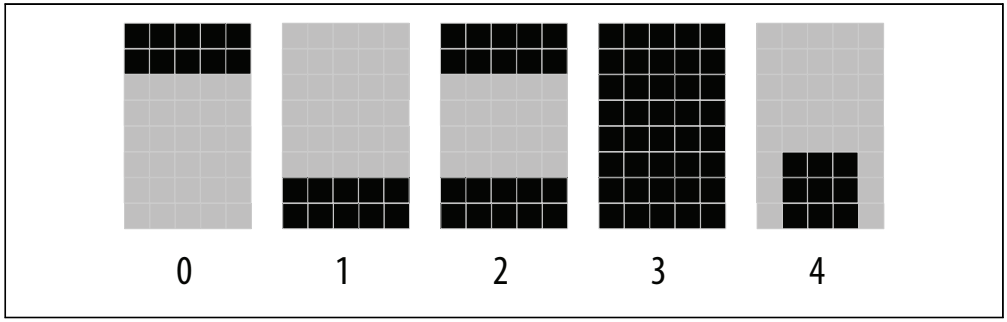


Abbildung 11-4: Selbstdefinierte Zeichen für große Ziffern

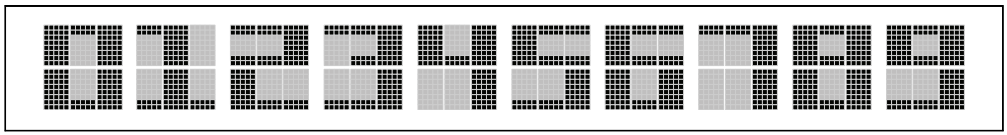


Abbildung 11-5: Aus selbstdefinierten Zeichen zusammengesetzte Ziffern

Das glyphs-Array definiert die Pixel für fünf selbstdefinierte Zeichen. Die zwei Dimensionen des Arrays stehen in den eckigen Klammern:

```
byte glyphs[5][8] = {
```

[5] ist die Zahl der Zeichen und [8] die Anzahl der Zeilen pro Zeichen. Jedes Element enthält Einsen und Nullen die festlegen, ob ein Pixel an dieser Stelle der Zeile an oder aus ist. Wenn Sie die Werte in glyph[0] (dem ersten Zeichen) mit Abbildung 11-2 vergleichen, können Sie sehen, dass die Einsen den dunklen Pixeln entsprechen:

```
{ B11111,B11111,B00000,B00000,B00000,B00000,B00000,B00000 },
```

Jede große Ziffer setzt sich aus sechs der selbstdefinierten Zeichen zusammen, drei für die obere und drei für die untere Hälfte. Die Arrays bigDigitsTop und bigDigitsBot definieren, welches selbstdefinierte Zeichen für die oberen und unteren Zeilen auf der LCD-Anzeige verwendet wird.

Siehe auch

Falls Sie wirklich große Ziffern benötigen, finden Sie in Kapitel 7 Informationen zu 7-Segment-LED-Anzeigen. 7-Segment-Anzeigen gibt es in Größen von etwa einem bis zu 5 Zentimetern und mehr. Sie benötigen wesentlich mehr Strom als LC-Displays und können Buchstaben und Symbole nicht besonders gut darstellen, sind aber eine gute Wahl, wenn Sie etwas groß darstellen wollen.

11.8 Kleine Pixel darstellen

Problem

Sie wollen Informationen mit einer feineren Auflösung als ein einzelnes Zeichen darstellen, z.B. um ein Balkendiagramm auszugeben.

Lösung

Rezept 11.7 beschreibt, wie man große Symbole darstellen kann, die aus mehr als einem Zeichen bestehen. Dieses Rezept macht das Gegenteil: Es definiert acht kleine Symbole, jedes einen Pixel höher als das vorherige (siehe Abbildung 11-6).

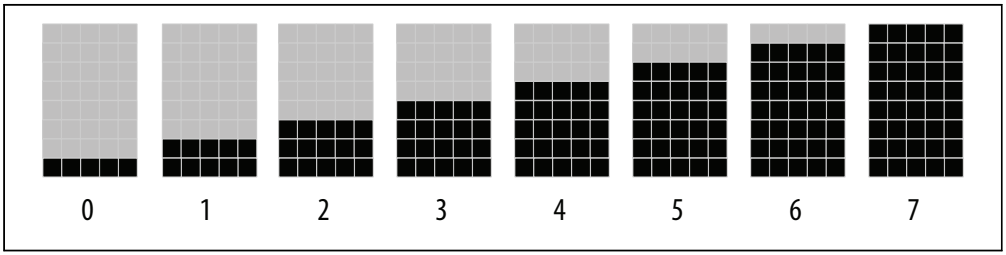


Abbildung 11-6: Acht selbstdefinierte Zeichen bilden vertikale Balken

Diese Symbole werden genutzt, um Balkendiagramme zu zeichnen. Wie das geht, zeigt der folgende Sketch:

```
/*
* customCharPixels
*/

#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

//Zeilen und Spalten für ihr LCD anpassen
const int numRows = 2;
const int numCols = 16;


// Bitarray definiert Pixel für 8 selbstdefinierte Zeichen
// Einsen und Nullen geben an, ob ein Pixel an oder aus ist

 byte glyphs[8][8] = {
   {B00000,B00000,B00000,B00000,B00000,B00000,B00000,B11111},   // 0
   {B00000,B00000,B00000,B00000,B00000,B00000,B11111,B11111},   // 1
   {B00000,B00000,B00000,B00000,B00000,B11111,B11111,B11111},   // 2
   {B00000,B00000,B00000,B00000,B11111,B11111,B11111,B11111},   // 3
   {B00000,B00000,B00000,B11111,B11111,B11111,B11111,B11111},   // 4
   {B00000,B00000,B11111,B11111,B11111,B11111,B11111,B11111},   // 5
   {B00000,B11111,B11111,B11111,B11111,B11111,B11111,B11111},   // 6
   {B11111,B11111,B11111,B11111,B11111,B11111,B11111,B11111}}; // 7

void setup ()
{
 lcd.begin(numCols, numRows);
 for(int i=0; i < 8; i++)
  lcd.createChar(i, glyphs[i]);   // Selbstdefinierte Zeichen erzeugen
 lcd.clear();
}


void loop ()
{
 for( byte i=0; i < 8; i++)
  lcd.write(i);  // Alle Balken darstellen
 delay(2000);
 lcd.clear();
}
```

Diskussion

Der Sketch erzeugt acht selbstdefinierte Zeichen, jedes ein Pixel höher als das vorige (siehe Abbildung 11-6). Sie werden nacheinander in der oberen Zeile des LCDs ausgegeben. Mit diesen »Balkendiagramm«-Zeichen können Sie Werte in Ihrem Sketch darstellen, die auf einem Bereich von 0 bis 7 abgebildet werden können. Das folgende Code-Fragment gibt beispielsweise einen Wert aus, der über den Analogeingang 0 eingelesen wurde:

```
int value = analogRead(0);
byte glyph = map(value, 0, 1023,0,8);// Proportionalen Wert zwischen 0 und 7 zurückgeben
lcd.print(glyph);
```

Für eine höhere Auflösung können Sie die Balken auch stapeln. Die Funktion `doubleHeightBars` im nachfolgenden Code gibt einen Wert zwischen 0 und 15 mit einer Auflösung von 16 Pixeln über zwei Zeilen der Anzeige aus:

```
void doubleHeightBars(int value, int column)
{
char upperGlyph;
char lowerGlyph;


 if(value < 8)
 {
   upperGlyph = ' '; // Kein Pixel an
   lowerGlyph = value;
 }
 else
 {
   upperGlyph = value - 8;
   lowerGlyph = 7; // Alle Pixel an
 }


 lcd.setCursor(column, 0); // Obere Hälfte ausgeben
 lcd.write(upperGlyph);
 lcd.setCursor(column, 1); // Untere Hälfe ausgeben
 lcd.write(lowerGlyph);
}
```

Die `doubleHeightBars`-Funktion kann wie folgt genutzt werden, um den Wert eines Analogeingangs auszugeben:

```
for( int i=0; i < 16; i++)
{
 int value = analogRead(0);
 value = map(value, 0, 1023,0,16);
 doubleHeightBars(value, i); // Wert zwischen 0 und 15 ausgeben
 delay(1000);  // Einmal pro Sekunde aktualisieren
}
```

Wenn Sie horizontale Balken brauchen, können Sie fünf Zeichen definieren (jedes ein Pixel breiter als das vorige) und eine ähnliche Logik wie bei den vertikalen Balken verwenden, um die Zeichen zu bestimmen, die ausgegeben werden sollen.

Ein komplexeres Beispiel dieser Technik finden Sie in einem Sketch, das eine bekannte Computersimulation, John Conways Spiel des Lebens (Game of Life) implementiert. Sie können den Sketch auf der Website zu diesem Buch (*http://shop.oreilly.com/product/0636920022244.do*) herunterladen.

11.9 Ein graphisches LC-Display anschließen und nutzen

Problem

Sie wollen Grafik und Text auf einem LCD mit einem KS0108 (oder kompatiblen) LCD-Treiber ausgeben.

Lösung

Diese Lösung verwendet die Arduino GLCD-Bibliothek zur Ansteuerung des Displays. Sie können Sie von *http://code.google.com/p/glcd-arduino/downloads/list* herunterladen (Hilfe bei der Installation von Bibliotheken finden Sie in).

Es gibt viele verschiedene GLCD-Controller. Stellen Sie sicher, dass Ihrer ein KS0108 oder ein kompatibler ist.

Die Anschlüsse von GLCDs sind nicht standardisiert, d.h., Sie müssen auf dem Datenblatt nachsehen, wie es richtig anzuschließen ist. Der fehlerhafte Anschluss der Signalleitungen ist die häufigste Fehlerursache und besondere Sorgfalt ist bei den Versorgungsanschlüssen vonnöten, da der falsche Anschluss das Panel beschädigen kann.

Die meisten GLCD-Panels benötigen einen externen variablen Widerstand, um die LCD-Betriebsspannung (Kontrast) einzustellen und eventuell einen weiteren (festen) Widerstand, um den Strom für die Hintergrundbeleuchtung zu beschränken. Das Datenblatt des Panels sollte alle Informationen zum Anschluss und die benötigten Komponenten enthalten.

Tabelle 11-2 zeigt den Standardanschluss eines KS0108-Panels an einen Arduino (oder Mega). Sie müssen auf dem Datenblatt Ihres Panels nachsehen, wo bei Ihrem Display die jeweiligen Funktionen liegen. Die Tabelle zeigt die drei gängigsten Panel-Layouts. Das erste (in der Tabelle mit »Panel A« bezeichnet), ist in Abbildung 11-7 zu sehen. Die Dokumentation der GLCD-Bibliothek enthält farbige Anschlussdiagramme der gängigsten Displays.

Tabelle 11-2: Anschluss eines KS0108-Panels an einen Arduino oder Mega

Arduino-Pins	Mega-Pins	GLCD-Funktion	Panel A	Panel B	Panel C	Kommentar
5V	5V	+5 volts	1	2	13	
Gnd	Gnd	Gnd	2	1	14	
–	–	Contrast in	3	3	12	Schleifer des Kontrast-Potis
8	22	D0	4	7	1	
9	23	D1	5	8	2	
10	24	D2	6	9	3	
11	25	D3	7	10	4	
4	26	D4	8	11	5	
5	27	D5	9	12	6	
6	28	D6	10	13	7	

Tabelle 11-2: Anschluss eines KS0108-Panels an einen Arduino oder Mega (Fortsetzung)

Arduino-Pins	Mega-Pins	GLCD-Funktion	Panel A	Panel B	Panel C	Kommentar
7	29	D7	11	14	8	
14 (Analog 0)	33	CSEL1	12	15	15	Chip-Select 1
15 (Analog 1)	34	CSEL2	13	16	16	Chip-Select 2
Reset		Reset	14	17	18	Mit Reset verbinden
16 (Analog 2)	35	R_W	15	5	10	Lesen/Schreiben (Read/Write)
17 (Analog 3)	36	D_I	16	4	11	Daten/Instruktionen (RS)
18 (Analog 4)	37	EN	17	6	9	Enable
–	–	Contrast out	18	18	17	10K oder 20K vorgegeben
–	–	Backlight +5	19	19	19	Siehe Datenblatt
Gnd	Gnd	Backlight Gnd	20	20	20	Siehe Datenblatt

Die Zahlen in den Arduino- und Mega-Spalten sind die Arduino- (oder Mega-) Pins, die in der mit der Bibliothek mitgelieferten Konfigurationsdatei verwendet werden. Sie können auch andere Pins verwenden, wenn sie bereits belegt sind. Wenn Sie die Anschlüsse ändern, müssen Sie auch die Zuweisungen in der Konfigurationsdatei anpassen und sich in der Bibliotheks-Dokumentation ansehen, wie man die Konfigurationsdatei editiert.

Der Anschluss des Panels entsprechend der Standardkonfiguration und die Ausführung des Sketches in diesem Rezept ermöglicht es Ihnen, alles auszutesten, bevor Sie die Konfiguration ändern. Eine der Verschaltung nicht entsprechende Konfiguration ist die häufigste Fehlerursache, weshalb bei einem Test mit minimalen Änderungen die Wahrscheinlichkeit steigt, dass es auf Anhieb funktioniert.

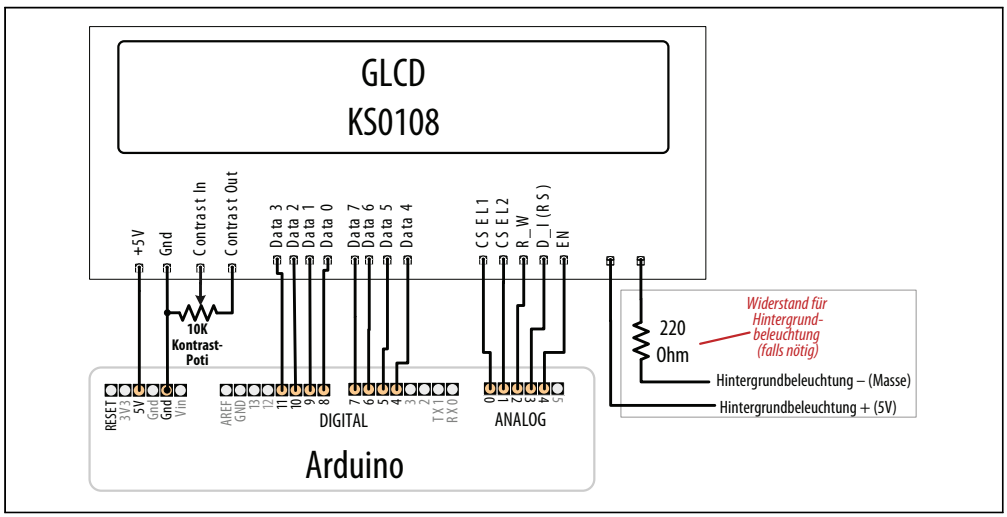


Abbildung 11-7: Anschluss eines GLCDs für Panels vom Typ A; Ihre Pinbelegung finden Sie auf dem Datenblatt

Der folgende Sketch gibt etwas Text und einige graphische Objekte aus:

```
/*
  glcd
 */

#include <glcd.h>

#include "fonts/allFonts.h"     // Zugriff auf alle mitgelieferten Schriften

int count = 0;

void setup()
{
 GLCD.Init(NON_INVERTED);       // Bibliothek initialsieren
 GLCD.ClearScreen();
 GLCD.SelectFont(System5x7);    // Systemfont in fester Breite wählen
 GLCD.print("Hallo, Welt");     // Text ausgeben
 delay(3000);
}


void loop()
{

  GLCD.ClearScreen();
  GLCD.DrawRect(0, 0, 64, 61, BLACK); // Rechteck auf der linken Seite der Anzeige
  // Abgerundetes Rechteck um den Textbereich
  GLCD.DrawRoundRect(68, 0, 58, 61, 5, BLACK);
  for(int i=0; i < 62; i += 4)
  {
   // Linien von oben links nach unten rechts zeichnen
   GLCD.DrawLine(1,1,63,i, BLACK);
  }
  GLCD.DrawCircle(32,31,30,BLACK);   // Kreis in der Mitte der linken Seite
  GLCD.FillRect(92,40,16,16, WHITE); // Textbereich löschen
  GLCD.CursorTo(5,5);                // Textcursor positionieren
  GLCD.PrintNumber(count);           // und eine Zahl ausgeben
  count = count + 1;
  delay(1000);
}
```

Diskussion

Die Bibliothek bietet eine Vielzahl grundlegender High-Level-Zeichenfunktionen, von denen einige in diesem Sketch demonstriert werden. Alle Funktionen sind in der Dokumentation der Bibliothek beschrieben.

Die Bildschirmkoordinaten für Text und Grafik beginnen in der linken oberen Ecke. Die am weitesten verbreiteten GLCD-Panels haben 128 × 64 Pixel und die Bibliothek arbeitet standardmäßig mit dieser Auflösung. Hat ihr Panel eine andere Auflösung, müssen Sie die Konfigurationsdatei der Bibliothek entsprechend korrigieren (momentan werden Panels mit bis zu 255 × 255 Punkten unterstützt).

GLCD erlaubt die Ausgabe von Text auf dem Bildschirm mit Befehlen, die den Arduino print-Befehlen für den seriellen Port ähneln. Darüber hinaus können Sie die Schriftart und -größe festlegen. Sie können auch einen Bereich der Anzeige festlegen, die als Textfenster verwendet werden soll. In diesem Bereich haben Sie dann ein »virtuelles Terminal«, in dem Text innerhalb der definierten Grenzen ausgeben und gescrollt wird. Der nachfolgende Code erzeugt zum Beispiel ein 32 Pixel großes Quadrat in der Mitte des Bildschirms:

```
gText myTextArea = gText(GLCD.CenterX-16, GLCD.CenterY -16, GLCD.CenterX +16,
GLCD.CenterY+16);
```

Mit Code wie dem folgenden können Sie eine Schriftart auswählen und im Textbereich ausgeben:

```
myTextArea.SelectFont(System5x7); // Systemfont für Textbereich wählen
name textTop
myTextArea.println("Los!");       // Eine Textzeile im Textbereich ausgeben.
```

Der mit der Bibliothek mitgelieferte Beispiel-Sketch zeigt, wie man mehrere Textbereiche zusammen mit Grafikelementen nutzen kann.

Diese Grafikanzeigen haben wesentlich mehr Anschlüsse als Text-LCDs und Sie müssen darauf achten, dass Ihr Panel korrekt angeschlossen ist.

Wenn keine Pixel auf dem Display erscheinen oder verstümmelt sind, machen Sie Folgendes:

- Überprüfen Sie die +5V- und Masse-Anschlüsse zwischen dem Arduino und dem GLCD-Panel.
- Stellen Sie sicher, dass alle Daten- und Befehlspins dem Datenblatt entsprechend verschaltet sind und mit der Konfiguration übereinstimmen. Das ist die häufigste Fehlerursache.
- Überprüfen Sie mit Hilfe des Datenblatts, dass die richtigen Timing-Werte in der Konfigurationsdatei eingestellt sind.
- Überprüfen Sie die Kontrast-Spannung (typischerweise zwischen –3 und –4 Volt) am Contrast-in-Pin des LCD-Panels. Gehen Sie den gesamten Wertebereich des Potis langsam durch, während der Sketch läuft. Manche Displays sind bei dieser Einstellung sehr empfindlich.
- Stellen Sie sicher, dass der Sketch korrekt kompiliert und auf den Arduino hochgeladen wurde.
- Führen Sie den GLCDdiags Diagnose-Sketch aus. Er steht im Menü über Examples→GLCD→GLCDdiags zur Verfügung.

Wenn die linke und rechte Seite des Bildes vertauscht ist, vertauschen Sie die CSEL1- und CSEL2-Anschlüsse (Sie können die Pins auch in der Konfigurationdatei vertauschen).

11.10 Bitmaps für graphische Displays

Problem

Sie wollen eigene graphische Images (Bitmaps) entwerfen und mit dem GLC-Display aus Rezept 11.9 einsetzen. Die Schriftdefinition und der Text soll im Programmspeicher abgelegt werden, um die RAM-Nutzung zu minimieren.

Lösung

Sie können die mit der Bibliothek mitgelieferten Bitmaps nutzen, oder eigene entwerfen. Bitmaps werden in Header-Dateien mit der Endung *.h* definiert. Zum Beispiel findet sich ein Arduino-Icon namens *ArduinoIcon.h* im *bitmap*-Ordner des GLCD-Bibliotheksverzeichnisses. Der Ordner enthält auch eine Datei namens *allBitmaps.h*, die Details zu allen mitgelieferten Bitmaps enthält. Sie können also die folgende Zeile einfügen, um alle mitgelieferten (oder neuen) Bitmaps verfügbar zu machen:

```
#include "bitmaps/allBitmaps.h" // Bindet alle mitgelieferten Bitmaps ein
```

Beachten Sie, dass das Einbinden aller Bitmaps keinen Speicher verbraucht, solange sie im Sketch nicht explizit mit `DrawBitmap` referenziert werden.

Um das Hinzufügen eigener Bitmaps zu ermöglichen, enthält die GLCD-Bibliothek ein Utility namens glcdMakeBitmap, das Dateien vom *.gif*, *.jpg*, *.bmp*, *.tga* oder *.png* in eine Header-Datei umwandelt, die von GLCD genutzt werden kann. Die Datei *glcdMakeBitmap.pde* ist ein Processing-Sketch, der in der Processing-Umgebung ausgeführt werden kann. Der Sketch liegt im Verzeichnis *bitmaps/utils/glcdMakeBitmap* directory. Weitere Informationen zu Processing finden Sie unter *http://processing.org/*.

Es gibt auch eine *.java-* (Java) Laufzeit-Datei (*glcdMakeBitmap.jar*) und eine *.java-*(Java-)Quelldatei (*glcdMakeBitmap.java*) im Verzeichnis *bitmaps/utils/Java*.

Führen Sie den Sketch aus, indem Sie ihn in Processing laden (oder die *.jar*-Datei anklicken). Ziehen Sie die umzuwandelnden Images dann einfach über das Fenster. Die Datei wird im *bitmaps*-Verzeichnis gespeichert und es wird automatisch ein Eintrage in *allBitMaps.h* eingefügt, d.h., das neue Image kann direkt im Sketch verwendet werden.

Um das zu demonstrieren, benennen Sie ein Image auf Ihrem Computer in *me.jpg* um. Starten Sie dann glcdMakeBitmap und ziehen Sie das Image in das erscheinende Fenster.

Kompilieren Sie den folgenden Sketch und laden Sie ihn hoch. Auf dem Display erscheint das mitgelieferte Arduino-Icon gefolgt von dem von Ihnen erzeugten Image:

```
/*
 * GLCDImage
 * In me.h definiertes Image ausgeben
 */

#include <glcd.h>

#include "bitmaps/allBitmaps.h"    // Alle Images im bitmap-Ordner einbinden
```

```
void setup()
{
 GLCD.Init();  // Bibliothek initialisieren
 GLCD.ClearScreen();
 GLCD.DrawBitmap(ArduinoIcon, 0,0); // Mitgelieferte Bitmap zeichnen
 delay(5000);
 GLCD.ClearScreen();
 GLCD.DrawBitmap(me, 0,0); // Ihre Bitmap zeichnen
}

void loop()
{

}
```

Die folgende Zeile zeichnet das in *ArduinoIcon.h* definierte Image, das mit der Bibliothek mitgeliefert wird:

```
GLCD.DrawBitmap(ArduinoIcon, 0,0); // Mitgelieferte Bitmap zeichnen
```

Nach einer Pause zeichnet die folgende Zeile das von Ihnen erzeugte Image aus der Datei *me.h*:

```
GLCD.DrawBitmap(me, 0,0);
```

Siehe auch

Weitere Informationen zur Erzeugung und Nutzung graphischer Images finden Sie in der Dokumentation der Bibliothek.

Die Dokumentation beschreibt auch, wie man eigene Schriften erzeugen kann.

11.11 Text auf dem Fernseher ausgeben

Problem

Sie wollen Text auf einem Fernseher oder einem Monitor mit Videoeingang ausgeben.

Lösung

Dieses Rezept nutzt ein Shield namens TellyMate, um Texte und Blockgrafiken auf einem Fernseher auszugeben. Das Shield wird am Arduino aufgesteckt und besitzt einen Ausgang, der mit dem Videoeingang eines Fernsehers verbunden werden kann.

Der folgende Sketch gibt alle Zeichen, die der TellyMate darstellen kann, auf einem Fernseher aus:

```
/*
 TellyMate
 Einfache Demo des TellyMate-Shield
*/
```

```
const byte ESC = 0x1B; // In TellyMate-Befehlen verwendetes ASCII-Escape-Zeichen

void setup()
{
 Serial.begin(57600); // 57k6 Baud ist die Standard-Geschwindigkeit des TellyMate
 clear(); // Bildschirm löschen
 Serial.print("  TellyMate-Zeichensatz"); // Etwas Text ausgeben
 delay(2000);
}

void loop()
{

 byte charCode = 32;   // Die Zeichen 0 bis 31 sind Steuerzeichen
 for(int row=0; row < 7; row++) // 7 Zeilen ausgeben
 {
  setCursor(2, row + 8); // Display zentrieren
  for(int col= 0; col < 32; col++) // 32 Zeichen pro Zeile
  {
    Serial.print(charCode);
    charCode = charCode + 1;
    delay(20);
  }
 }
 delay(5000);
 clear();
}

// TellyMate-Hilfsfunktionen

void clear( ) // Bildschirm löschen
{ // <ESC>E
 Serial.print(ESC);
 Serial.print('E');
}

void setCursor( int col, int row) // Cursor positionieren
{ // <ESC>Yrc
 Serial.print(ESC);
 Serial.print('Y' ) ;
 Serial.print((unsigned char)(32 + row)) ;
 Serial.print((unsigned char)(32 + col)) ;
}
```

Diskussion

Der Arduino steuert die TellyMate-Anzeige, indem er Befehle über den seriellen Port sendet.

TellyMate kommuniziert mit dem Arduino über den seriellen Port, d.h., Sie müssen das Shield abtrennen, um Sketches hochzuladen.

Abbildung 11-8 zeigt, welche Zeichen dargestellt werden können. Eine Tabelle mit den Werten aller Zeichen finden Sie unter *http://en.wikipedia.org/wiki/Code_page_437*.

Abbildung 11-8: TellyMate-Zeichensatz (Codeseite 437)

Die Zeichen 0 bis 31 werden als Steuerbefehle für den Bildschirm interpretiert, d.h., nur die Zeichen 32 bis 255 können dargestellt werden.

Der Sketch verwendet nicht-druckbare Zeichen, sogenannte *Escape-Codes*, um druckbare Zeichen von Befehlen zur Bildschirmsteuerung zu unterscheiden. Solche Steuercodes bestehen aus dem ESC-Zeichen (die Abkürzung für *Escape*, hex 0x1b), gefolgt von einem oder mehreren Zeichen, die die eigentliche Funktion bestimmen. Details zu allen Steuercodes finden Sie in der TellyMate-Dokumentation.

Der Sketch nutzt einige Hilfsfunktionen, die die zur Steuerung benötigen Zeichenfolgen senden, damit Sie sich auf die eigentlichen Aktivitäten des Sketches konzentrieren können.

Auf dem Bildschirm erscheint ein blinkender Cursor, den Sie mit einem Steuercode ausschalten können. Die Funktion `cursorHide` schaltet den Cursor aus:

```
void cursorHide()
{ // <ESC>f
 Serial.write(ESC) ;  // Das Escape-Zeichen
 Serial.print('f' ) ;  // gefolgt vom Buchstaben f schaltet den Cursor aus.
}
```

Um einen Rahmen um die Grenzen des Bildschirms zu zeichnen, fügen Sie die Funktionen `drawBox` und `showXY` am Ende des obigen Sketches hinzu. Damit der Sketch Sie auch nutzt, fügen Sie die folgende Zeile innerhalb der öffnenden Klammer der Schleife ein:

```
drawBox(1,0, 38, 24); // Bildschirm ist 38 Zeichen breit und 25 hoch
```

Die `drawBox`-Funktion gibt die vier Ecken sowie die oberen, unteren und seitlichen Linien mit Hilfe entsprechender Blockgrafik-Codes aus:

```
// Für den Rahmen verwendete Zeichen
// siehe http://en.wikipedia.org/wiki/Code_page_437
const byte boxUL = 201;
const byte boxUR = 187;
const byte boxLL = 200;
const byte boxLR = 188;
```

```
const byte HLINE = 205; // Horizontale Linie
const byte VLINE = 186; // Vertikale Linie

void drawBox( int startRow, int startCol, int width, int height)
{
 // Zeichne obere Zeile
 showXY(boxUL, startCol,startRow); // Obere linke Ecke
 for(int col = startCol + 1; col < startCol + width-1; col++)
   Serial.print(HLINE); // Die Linie
 Serial.print(boxUR); // Obere rechte Ecke

 // Rahmen links und rechts
 for(int row = startRow + 1; row < startRow + height -1; row++)
 {
   showXY(VLINE, startCol,row); // Linker Rand
   showXY(VLINE, startCol + width-1,row); // Rechter Rand
 }
 // Zeichne untere Zeile
 showXY(boxLL, 0, startRow+height-1); // Untere linke Ecke
 for(int col = startCol + 1; col < startCol + width-1; col++)
   Serial.write(HLINE);
 Serial.write(boxLR);

}
```

Eine von `drawBox` genutzte Hilfsfunktion namens `showXY` fasst die Cursor-Positionierung und die Ausgabe zusammen:

```
void showXY( char ch, int x, int y){
 // Zeichen an x- und y-Position ausgeben
 setCursor(x,y);
 Serial.write(ch);
}
```

Hier ein weiterer Sketch, der die Befehle zur Cursorsteuerung nutzt, um einen »Ball« über den Bildschirm springen zu lassen:

```
 /*
 TellyBounce
*/

// Bildschirmgrenzen definieren
const int HEIGHT = 25;   // Anzahl der Zeilen
const int WIDTH  = 38;   // Zeichen pro Zeile
const int LEFT   = 0;   // Daraus abgeleitete nützliche Konstanten
const int RIGHT  = WIDTH -1;
const int TOP    = 0;
const int BOTTOM = HEIGHT-1;

const byte BALL  = 'o';    // Zeichencode für den Ball
const byte ESC = 0x1B;     // Von TellyMate-Befehlen verwendetes ASCII-Escape-Zeichen

int ballX = WIDTH/2;       // x-Position des Balls
int ballY = HEIGHT/2;      // y-Position des Balls
int ballDirectionY = 1;    // x-Richtung des Balls
int ballDirectionX = 1;    // y-Richtung des Balls
```

```
// Dieses Intervall bewegt den Ball in weniger als 4 Sekunden über den 38-Zeichen-Bildschirm
long interval = 100;

void setup()
{
 Serial.begin(57600);  // 57k6 Baud ist die TellyMate-Standardgeschwindigkeit
 clear();              // Bildschirm löschen
 cursorHide();         // Cursor ausschalten
}

void loop()
{
 moveBall();
 delay(interval);
}

void moveBall() {
 // Erreicht der Ball den oberen oder unteren Rand, kehren wir die y-Richtung um
 if (ballY == BOTTOM || ballY == TOP)
  ballDirectionY = -ballDirectionY;

 // Erreicht der Ball den linken oder rechten Rand, kehren wir die x-Richtung um
 if ((ballX == LEFT) || (ballX == RIGHT))
  ballDirectionX = -ballDirectionX;

 // Alte Position des Balls löschen
 showXY(' ', ballX, ballY);

 // Position des Balls in beiden Richtung erhöhen
 ballX = ballX + ballDirectionX;
 ballY = ballY + ballDirectionY;

 // Ball an neuer Position ausgeben
 showXY(BALL, ballX, ballY);
}

// TellyMate-Hilfsfunktionen

void clear( ) // Bildschirm löschen
{ // <ESC>E
 Serial.write(ESC);
 Serial.write('E');
}

void setCursor( int col, int row) // Cursor positionieren
{ // <ESC>Yrc
 Serial.write(ESC);
 Serial.write('Y' ) ;
 Serial.write((unsigned char)(32 + row)) ;
 Serial.write((unsigned char)(32 + col)) ;
}

void cursorShow( )
{ // <ESC>e
 Serial.write(ESC) ;
 Serial.write('e') ;
}
```

```
void cursorHide()
{ // <ESC>f
 Serial.write(ESC) ;
 Serial.write('f' ) ;
}

void showXY( char ch, int x, int y){
 // Zeichen an x- und y-Position ausgeben
 setCursor(x,y);
 Serial.write(ch);
}
```

Siehe auch

Detaillierte Informationen zum TellyMate-Shield finden Sie unter *http://www.batsocks.co.uk/products/Shields/index_Shields.htm*.

Weitere Informationen zur Codeseite 437, einschließlich einer Zeichentabelle, finden Sie unter *http://en.wikipedia.org/wiki/Code_page_437*.

KAPITEL 12

Datum und Uhrzeit

12.0 Einführung

Die Arbeit mit Zeiten ist ein grundlegendes Element der Interaktivität von Computern. Dieses Kapitel behandelt die im Arduino fest eingebauten Funktionen und stellt zusätzliche Techniken zur Behandlung von Zeitverzögerungen, Zeitmessungen und Zeit- und Datumsangaben vor.

12.1 Zeitverzögerungen

Problem

Ihr Sketch soll eine bestimmte Zeitspanne pausieren. Dabei kann es sich um ein paar Millisekunden handeln oder um Sekunden, Minuten, Stunden oder Tage.

Lösung

Die Arduino-Funktion `delay` wird im gesamten Buch von vielen Sketches genutzt. `delay` hält den Sketch für die Zeit in Millisekunden an, die als Parameter übergeben wird. (1000 Millisekunden sind eine Sekunde.) Der folgende Sketch zeigt, wie Sie mit `delay` nahezu jede Pause hinbekommen:

```
/*
 * delay Sketch
 */

const long oneSecond = 1000;  // Eine Sekunde sind 1000 Millisekunden
const long oneMinute = oneSecond * 60; // Eine Minute
const long oneHour   = oneMinute * 60; // Eine Stunde
const long oneDay    = oneHour * 24;  // Ein Tag


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

```
void loop()
{
 Serial.println("Eine Millisekunde pausieren");
 delay(1);
 Serial.println("Eine Sekunde pausieren");
 delay(oneSecond);
 Serial.println("Eine Minute pausieren");
 delay(oneMinute);
 Serial.println("Eine Stunde pausieren");
 delay(oneHour);
 Serial.println("Einen Tag pausieren");
 delay(oneDay);
 Serial.println("Bereit für den Neustart");
}
```

Diskussion

Der Wertebereich der `delay`-Funktion reicht von einer tausendstel Sekunde bis zu etwa 25 Tagen bzw. etwas unter 50 Tagen, wenn Sie mit unsigned long arbeiten (in Kapitel 2 erfahren Sie mehr über die Variablentypen).

Die `delay`-Funktion hält die Ausführung des Sketches für die angegebene Zeitspanne an. Wenn Sie während dieser Zeit andere Aufgaben erledigen müssen, ist die `millis`-Funktion, wie in Rezept 12.2 erläutert, die bessere Wahl.

Für sehr kurze Zeitverzögerungen können Sie `delayMicroseconds` verwenden. Eine Millisekunde dauert 1000 Mikrosekunden und eine Million Mikrosekunden ist eine Sekunde. Die Zeitspannen von `delayMicroseconds` liegen zwischen einer Mikrosekunde und ca. 16 Millisekunden. Ist die Pause aber länger als ein paar tausend Mikrosekunden, sollten Sie `delay` verwenden:

```
delayMicroseconds(10); // 10 Mikrosekunde warten
```

`delay` und `delayMicroseconds` pausieren mindestens für die im Parameter angegebene Dauer, doch die Verzögerung kann auch etwas länger sein, wenn es innerhalb dieser Zeitspanne zu Interrupts kommt.

Siehe auch

Die Arduino-Referenz zu `delay`: *http://www.arduino.cc/en/Reference/Delay*

12.2 Laufzeiten messen mit millis

Problem

Sie wollen wissen, wie viel Zeit seit einem Ereignis vergangen ist. Zum Beispiel wollen Sie wissen, wie lange eine Taste gedrückt wurde.

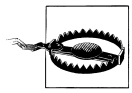
Lösung

Arduino besitzt eine Funktion namens `millis` (für Millisekunden), die im folgenden Sketch verwendet wird, um zu ermitteln, wie lange eine Taste gedrückt wurde (Rezept 5.2 beschreibt, wie der Taster anzuschließen ist):

```
/*
 millisDuration Sketch
 Gibt an, wie lange (in Millisekunden) eine Taste gedrückt wude
 */

const int switchPin = 2;           // Eingangspin

long startTime; // millis-Wert beim ersten Drücken der Taste
long duration;  // Variable für die Dauer

void setup()
{
 pinMode(switchPin, INPUT);
 digitalWrite(switchPin, HIGH); // Pullup-Widerstand aktivieren
 Serial.begin(9600);
}


void loop()
{
 if(digitalRead(switchPin) == LOW)
 {
   // Sobald die Taste gedrückt wurde
   startTime = millis();
   while(digitalRead(switchPin) == LOW)
     ; // Warten, solange die Taste gedrückt wird
   long duration = millis() - startTime;
   Serial.println(duration);
 }
}
```

Diskussion

Die `millis`-Funktion gibt in Millisekunden zurück, wie lange der aktuelle Sketch schon läuft.



Nach etwa 50 Tagen kommt es bei der `millis`-Funktion zu einem *Überlauf* (d.h., der Wert springt wieder auf 0 zurück). In 12.4 und 12.5 finden Sie Informationen darüber, wie Sie die Time-Bibliothek nutzen können, um Intervalle von Sekunden bis hin zu Jahren zu verarbeiten.

Indem Sie die Startzeit des Ereignisses festhalten, können Sie dessen Dauer ermitteln, wenn Sie die Startzeit von der aktuellen Zeit abziehen:

```
long duration = millis() - startTime;
```

Sie können eine eigene Verzögerungsfunktion mit `millis` entwickeln, die sich anderen Aufgaben widmet, während sie fortlaufend prüft, ob die festgelegte Zeitspanne abgelaufen ist. Ein entsprechendes Beispiel finden Sie im Sketch BlinkWithoutDelay, das bei der Arduino-Distribution mitgeliefert wird. Die folgenden Fragmente aus dem Sketch erläutern den Schleifencode:

```
void loop()
{
 // Hier steht der Code, der die ganze Zeit laufen muss...
```

Die nächste Zeile prüft, ob die gewünschte Zeitspanne verstrichen ist:

```
 if (millis() - previousMillis > interval)
 {
  // Festhalten, wann die LED zuletzt geblinkt hat
```

Ist die Zeitspanne verstrichen, wird der aktuelle `millis`-Wert in der Variablen `previousMillis` gespeichert:

```
  previousMillis = millis();

  // Ist die LED aus, schalten wir sie an, und umgekehrt
  if (ledState == LOW)
   ledState = HIGH;
  else
   ledState = LOW;

  // LED auf ledState setzen
  digitalWrite(ledPin, ledState);
 }
}
```

Hier nun eine Möglichkeit, diese Logik in eine Funktion namens `myDelay` zu packen, die den Code in `loop` pausieren lässt, während dieser Zeit aber eine andere Aufgabe erledigen kann. Sie können die Funktionalität für Ihre Anwendung anpassen, doch in diesem Beispiel leuchtet die LED fünfmal pro Sekunde auf, obwohl die print-Anweisung in `loop` mit 4-Sekunden-Intervallen ausgebremst wird:

```
// LED für festgelegte Dauer blinken lassen
const int ledPin =  13;      // LED-Pin

int ledState = LOW;         // ledState setzt LED
long previousMillis = 0;     // Letzter Update der LED

void setup()
{
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
}

void loop()
{
 Serial.println(millis() / 1000); // Verstrichene Zeit alle vier Sekunden ausgeben
 // Vier Sekunden warten (gleichzeitig aber eine LED schnell blinken lassen)
 myDelay(4000);
}
```

```
// duration ist Verzögerungszeit in Millisekunden
void myDelay(unsigned long duration)
{
 unsigned long start = millis();
 while (millis() - start <= duration)
 {
  blink(100);  // LED innerhalb der while-Schleife blinken lassen
 }
}


// interval gibt an, wie lange die LED an und aus sein soll
void blink(long interval)
{
 if (millis() - previousMillis > interval)
 {
  // Letztes Blinken der LED
  previousMillis = millis();
  // Ist die LED aus, schalten wir sie an, und umgekehrt
  if (ledState == LOW)
   ledState = HIGH;  else
   ledState = LOW;
  digitalWrite(ledPin, ledState);
 }
}
```

Sie können in der myDelay-Funktion Code für eine beliebige Aktion einfügen, die wiederholt ausgeführt werden soll, während die Funktion darauf wartet, dass die festgelegte Zeitspanne verstreicht.

Ein anderer Ansatz nutzt eine Bibliothek aus dem Arduino Playground namens TimedAction (*http://www.arduino.cc/playground/Code/TimedAction*):

```
#include <TimedAction.h>

//TimedAction-Klasse initialisieren, um den Status einer LED jede Sekunde zu ändern.
TimedAction timedAction = TimedAction(NO_PREDELAY,1000,blink);


const int ledPin =  13;    // LED-Pin
boolean ledState = LOW;


void setup()
{
 pinMode(ledPin,OUTPUT);
 digitalWrite(ledPin,ledState);
}

void loop()
{
 timedAction.check();
}

void blink()
```

```
{
 if (ledState == LOW)
  ledState = HIGH;
 else
  ledState = LOW;

 digitalWrite(ledPin,ledState);
}
```

Siehe auch

Die Arduino-Referenz für `millis`: *http://www.arduino.cc/en/Reference/Millis*

In 12.4 und 12.5 finden Sie Informationen darüber, wie Sie die Time-Bibliothek nutzen können, um Intervalle von Sekunden bis hin zu Jahren zu verarbeiten.

12.3 Die Dauer eines Impulses präziser messen

Problem

Sie wollen die Dauer eines Impulses im Mikrosekundenbereich genau bestimmen. Zum Beispiel wollen Sie die genaue Dauer eines `HIGH`- oder `LOW`-Impulses an einem Pin messen.

Lösung

Die `pulseIn`-Funktion gibt die Dauer (in Mikrosekunden) eines sich ändernden Signals an einem Digitalpin zurück. Der folgende Sketch gibt die Dauer der `HIGH`- und `LOW`-Impulse in Mikrosekunden aus, die von `analogWrite` erzeugt werden (beachten Sie den entsprechenden Abschnitt in Rezept 7.1 in Kapitel 7). Da die `analogWrite`-Impulse intern vom Arduino erzeugt werden, ist keine externe Verschaltung nötig:

```
/*
 PulseIn Sketch
 Dauer der HIGH- und LOW-Impulse von analogWrite ausgeben
 */


const int inputPin = 3;  // Analoger Ausgangspin
unsigned long val; // Enthält den Wert von pulseIn

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

 analogWrite(inputPin, 128);
 Serial.print("Schreibe 128 an Pin ");
 Serial.print(inputPin);
 printPulseWidth(inputPin);

 analogWrite(inputPin, 254);
 Serial.print("Schreibe 254 an Pin ");
```

```
 Serial.print(inputPin);
 printPulseWidth(inputPin);

}

void loop()
{
}

void printPulseWidth(int pin)
{
  val = pulseIn(pin, HIGH);
  Serial.print(": HIGH-Impulslänge = ");
  Serial.print(val);
  val = pulseIn(pin, LOW);
  Serial.print(", LOW-Impulslänge = ");
  Serial.println(val);
}
```

Diskussion

Auf dem seriellen Monitor erscheint:

```
Schreibe 128 an Pin 3: HIGH-Impulslänge = 989, LOW-Impulslänge = 997
Schreibe 254 an Pin 3: HIGH-Impulslänge = 1977, LOW-Impulslänge = 8
```

`pulseIn` misst, wie lange ein Impuls entweder `HIGH` oder `LOW` ist:

```
pulseIn(pin, HIGH); // Wie lange (in Mikrosekunden) ist Impuls HIGH
pulseIn(pin, LOW)  // Wie lange (in Mikrosekunden) ist Impuls HIGH
```

Die `pulseIn`-Funktion wartet darauf, dass der Impuls beginnt (bzw. auf einen Timeout, falls kein Impuls kommt). Standardmäßig liegt das Timeout bei einer Sekunde, aber Sie können das ändern, indem Sie die gewünschte Zeit in Mikrosekunden als dritten Parameter übergeben (denken Sie daran, dass 1000 Mikrosekunden eine Millisekunde sind):

```
pulseIn(pin, HIGH, 5000); // 5 Millisekunden auf Impuls warten
```

Der Timeout-Wert ist nur von Bedeutung, falls der Impuls nicht innerhalb der festgelegten Zeitspanne startet. Sobald der Beginn des Impulses erkannt wurde, beginnt die Funktion mit der Zeitmessung und kehrt erst zurück, wenn der Impuls endet.

`pulseIn` kann Werte zwischen etwa 10 Mikrosekunden bis drei Minuten messen, doch der Wert sehr langer Impulse ist nicht besonders genau.

Siehe auch

Die Arduino-Referenz zu `pulseIn`: *http://www.arduino.cc/en/Reference/PulseIn*

Rezept 6.4 zeigt, wie `pulseIn` die Impulsdauer eines Ultraschall-Abstandssensors misst.

Rezept 18.2 enthält Informationen zur Verwendung von Hardware-Interrupts.

12.4 Arduino als Uhr verwenden

Problem

Sie wollen die Uhrzeit (Stunden, Minuten und Sekunden) in einem Sketch ausgeben, ohne externe Hardware anschließen zu müssen.

Lösung

Der folgende Sketch verwendet die Time-Bibliothek zur Ausgabe der Uhrzeit. Sie kann von *http://www.arduino.cc/playground/Code/Time* heruntergeladen werden.

```
/*
 * Time Sketch
 *
 */

#include <Time.h>


void setup()
{
 Serial.begin(9600);
 setTime(12,0,0,1,1,11); // Zeit ist 12 Uhr mittags am 1.1.2011
}

void loop()
{
 digitalClockDisplay();
 delay(1000);
}

void digitalClockDisplay(){
 // Digitalanzeige der Zeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(month());
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

void printDigits(int digits){
 // Hilfsfunktion zur Uhrendarstellung: Gibt vorstehenden Doppelpunkt und führende 0 aus
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}
```

Diskussion

Die Time-Bibliothek ermöglicht es Ihnen, Datum und Uhrzeit nachzuhalten. Viele Arduino-Boards verwenden einen Quarzkristall für den Zeitgeber, der auf ein paar Sekunden pro Tag genau ist. Es gibt aber keine Batterie, mit deren Hilfe die Zeit beim Ausschalten erhalten bleibt. Aus diesem Grund beginnt die Zeitrechnung bei jedem Start des Sketches bei 0, d.h., Sie müssen Datum und Uhrzeit mit der Funktion `setTime` setzen. Der Sketch setzt Datum und Uhrzeit bei jedem Start auf den 1. Januar um 12 Uhr mittags.

Die Time-Bibliothek verwendet einen als Unix-Zeit (auch POSIX-Zeit) bezeichneten Standard. Die Werte geben die Zeit in Sekunden an, die seit dem 1.1.1970 verstrichen ist. Erfahrene C-Programmierer werden erkennen, dass das dem `time_t` entspricht, das in der ISO-Standard-C-Bibliothek zur Speicherung von Zeitwerten verwendet wird.

Natürlich ist es sinnvoller, Datum und Uhrzeit auf die aktuelle lokale Zeit einzustellen, statt einen festen Wert zu verwenden. Der folgende Sketch liest den numerischen Zeitwert (die Zahl der seit dem 1.1.1970 verstrichenen Sekunden) über den seriellen Port ein, um die Zeit zu setzen. Sie können einen Wert über den seriellen Monitor eingeben (die aktuelle Unix-Zeit findet sich auf vielen Webseiten, wenn man bei Google nach »Unix-Zeit umrechnen« sucht):

```
/*
 * TimeSerial Sketch
 * Time-Bibliothek über seriellen Port setzen
 *
 * Nachrichten bestehen aus dem Buchstaben T, gefolgt vom 10-stelligen Zeitwert
 * (in Sekunden seit dem 1.1.1970)
 * Geben Sie die folgende Zeile im seriellen Monitor ein,
 * um den 1.1.2011, 12 Uhr mittags einzustellen:
 * T1293883200
 *
 * Ein Processing-Beispiel-Sketch, der diese Nachrichten automatisch sendet,
 * ist in der Time-Bibliothek enthalten
 */

#include <Time.h>

#define TIME_MSG_LEN 11  // Zeitsynchronisation besteht aus dem HEADER, gefolgt von
                         // zehn ASCII-Ziffern
#define TIME_HEADER 'T'  // Header-Tag zur seriellen Zeitsynchronisation

void setup() {
 Serial.begin(9600);
 Serial.println("Warte auf Zeitsynchronisation");
}

void loop(){
 if(Serial.available() )
 {
  processSyncMessage();
 }
 if(timeStatus()!= timeNotSet)
```

```
 {
  // Wenn Datum/Uhrzeit gesetzt wurde
  digitalClockDisplay();
 }
 delay(1000);
}

void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(month());
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

void printDigits(int digits){
 // Hilfsfunktion zur Uhrendarstellung: Gibt vorstehenden Doppelpunkt und führende 0 aus
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}

void processSyncMessage() {
 // Wenn Zeitsynchronisation vom seriellen Port verfügbar ist, Datum/Uhrzeit aktualisieren und
true zurückgeben
 // Die Nachricht besteht aus einem Header und 10 ASCII-Ziffern
 while(Serial.available() >= TIME_MSG_LEN ){
  char c = Serial.read() ;
  Serial.print(c);
  if( c == TIME_HEADER ) {
   time_t pctime = 0;
   for(int i=0; i < TIME_MSG_LEN -1; i++){
    c = Serial.read();
    if( isDigit(c)) {
     pctime = (10 * pctime) + (c - '0') ; // Ziffern in eine Zahl umwandeln
    }
   }
   setTime(pctime);  // Uhr auf empfangenen Wert einstellen
  }
 }
}
```

Der Code zur Ausgabe von Uhrzeit und Datum ist mit dem obigen Code identisch, doch diesmal empfängt der Sketch Datum und Uhrzeit über den seriellen Port. Wenn Sie mit dem Empfang numerischer Daten über den seriellen Port nicht vertraut sind, sehen Sie sich die Diskussion in Rezept 4.3 an.

Ein Processing-Sketch namens SyncArduinoClock ist in den Beispielen der Time-Bibliothek enthalten (im Ordner *Time/Examples/Processing/SyncArduinoClock*). Dieser Proces-

sing-Sketch sendet bei einem Mausklick die aktuelle Zeit an den Arduino. Führen Sie SyncArduinoClock in Processing aus und stellen Sie sicher, dass der serielle Port mit dem Arduino verbunden ist (Kapitel 4 beschreibt, wie man einen Processing-Sketch ausführt, der mit dem Arduino kommuniziert). Die Nachricht `Warte auf Zeitsynchronisation` sollte im Processing-Textbereich (der schwarze Bereich am unteren Rand der Processing-IDE) erscheinen. Klicken Sie das Processing-Anwendungsfenster an (ein graues, 200 Pixel großes Quadrat), und im Textbereich sollten Datum und Uhrzeit erscheinen, wie sie vom Arduino-Sketch ausgegeben werden.

Sie können die Uhr auch über den seriellen Monitor stellen, wenn Sie die aktuelle Unix-Zeit kennen. *http://www.epochconverter.com/* ist eine der viele Webseiten, die die aktuelle Zeit in diesem Format bereitstellt. Kopieren Sie die zehnstellige Zahl, die als aktuelle Unix-Zeit angegeben wird, und fügen Sie sie in das Sendefenster des seriellen Monitors ein. Stellen Sie der Zahl den Buchstaben *T* voran und klicken Sie auf Send. Wenn Sie beispielsweise Folgendes eingeben:

```
T1282041639
```

sollte der Arduino jede Sekunde Datum und Zeit ausgeben:

```
10:40:49 17 8 2010
10:40:50 17 8 2010
10:40:51 17 8 2010
10:40:52 17 8 2010
10:40:53 17 8 2010
10:40:54 17 8 2010
. . .
```

Sie können die Zeit auch über Tasten oder andere Eingabegeräte wie Neigungssensoren, einen Joystick oder einen Drehgeber einstellen.

Der folgende Sketch verwendet zwei Taster, um die Zeiger der Uhr vor- und zurückzubewegen. Abbildung 12-1 zeigt die Verschaltung (falls Sie Hilfe bei den Tastern brauchen, sehen Sie sich Rezept 5.2 an):

```
/*
  AdjustClockTime Sketch
  Taster an Pins 2 und 3 justieren die Zeit
*/

#include <Time.h>

const int btnForward = 2; // Taster für Zeit vor
const int btnBack = 3;   // Taster für Zeit zurück

unsigned long prevtime; // Wann wurde die Uhr zuletzt ausgegeben

void setup()
{
 digitalWrite(btnForward, HIGH); // Interne Pullup-Widerstände aktivieren
 digitalWrite(btnBack, HIGH);
 setTime(12,0,0,1,1,11); // Wir beginnen mit dem 1.1.2011, 12 Uhr mittags
 Serial.begin(9600);
 Serial.println("Bereit");
```

```
}

void loop()
{
 prevtime = now();  // Zeit festhalten
 while( prevtime == now() )   // Bis zur nächsten Sekunde warten
 {
   // Wurde Taste gedrückt?
  if(checkSetTime())
    prevtime = now();  // Zeit wurde geändert, also Startzeit zurücksetzen
 }
 digitalClockDisplay();
}

// Prüft, ob Zeit korrigiert werden muss
// Gibt wahr zurück, wenn die Zeit geändert wurde
boolean checkSetTime()
{
int step;  // Um wie viele Sekunden bewegen wir uns (zurück, wenn negativ)
boolean isTimeAdjusted = false; // Wahr, wenn Zeit korrigiert wurde
 step = 1;  // Vorwärts
 while(digitalRead(btnForward)== LOW)
 {
  adjustTime(step);
  isTimeAdjusted = true; // Zeit wurde geändert
  step = step + 1; // Nächster Schritt ist größer
  digitalClockDisplay(); // Uhr aktualisieren
  delay(100);
 }
 step = -1;  // Rückwärts mit negativen Zahlen
 while(digitalRead(btnBack)== LOW)
 {
  adjustTime(step);
  isTimeAdjusted = true; // Zeit wurde geändert
  step = step - 1; // Nächster Schritt wird größer
  digitalClockDisplay(); // Uhr aktualisieren
  delay(100);
 }
 return isTimeAdjusted; // Zurückgeben, ob Zeit korrigiert wurde
}

void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(month());
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

void printDigits(int digits){
 // Hilfsfunktion zur Uhrendarstellung: Gibt vorstehenden Doppelpunkt und führende 0 aus
```

```
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}
```

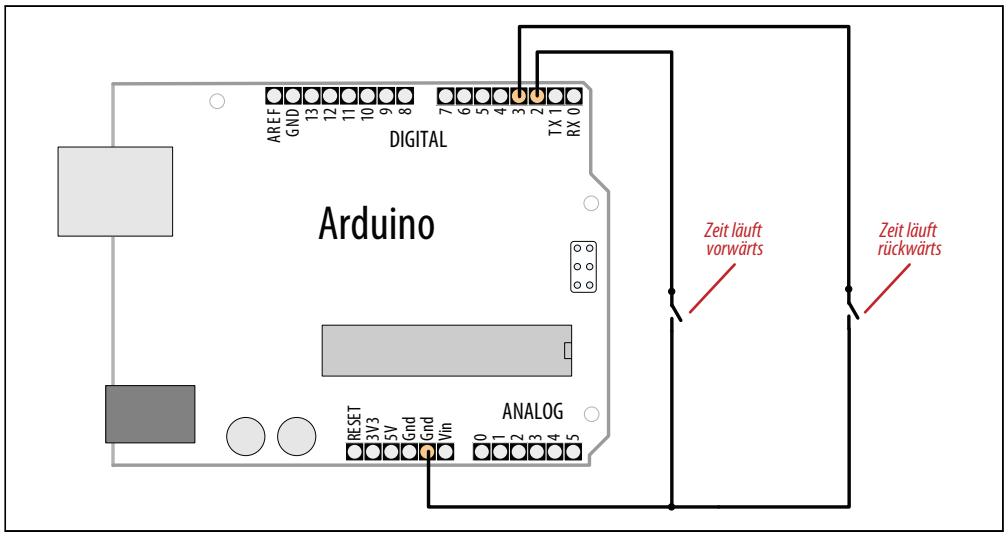


Abbildung 12-1: Zwei Taster zur Einstellung der Uhrzeit

Der Sketch nutzt die Funktionen `digitalClockDisplay` und `printDigits` aus Rezept 12.3, d.h., Sie müssen sie kopieren, bevor Sie den Sketch ausführen können.

Hier eine Variante des Sketches, der die Position eines variablen Widerstands nutzt, um die Richtung und die Geschwindigkeit der Korrektur zu bestimmen, wenn eine Taste gedrückt wird:

```
#include <Time.h>

const int potPin = 0;   // Poti für Richtung und Geschwindigkeit
const int buttonPin = 2; // Taster aktiviert Zeitkorrektur

unsigned long prevtime;  // Wann wurde die Uhrzeit zuletzt ausgegeben?

void setup()
{
 digitalWrite(buttonPin, HIGH); // Interne Pullup-Widerstände aktivieren
 setTime(12,0,0,1,1,11); // Wir beginnen mit dem 1.1.2011, 12 Uhr mittags
 Serial.begin(9600);
}

void loop()
{
 prevtime = now();  // Zeit festhalten
 while( prevtime == now() )   // Bis zur nächsten Sekunde warten
 {
   // Wurde Taste gedrückt?
```

```
  if(checkSetTime())
    prevtime = now();  // Zeit wurde geändert, also Startzeit zurücksetzen
 }
 digitalClockDisplay();
}

// Prüft, ob Zeit korrigiert werden muss
// Gibt wahr zurück, wenn die Zeit geändert wurde
boolean checkSetTime()
{
int value;  // Vom Poti eingelesener Wert
int step;  // Um wie viele Sekunden bewegen wir uns (zurück, wenn negativ)
boolean isTimeAdjusted = false;  // Wahr, wenn Zeit korrigiert wurde

 while(digitalRead(buttonPin)== LOW)
 {
  // here while button is pressed
   value = analogRead(potPin);  // Potiwert einlesen
   step = map(value, 0,1023, 10, -10);  // Wert auf gewünschten Wertebereich abbilden
   if( step != 0)
   {
     adjustTime(step);
     isTimeAdjusted = true; // Zeit wurde geändert
     digitalClockDisplay(); // Uhr aktualsieren
     delay(100);
   }
 }
 return isTimeAdjusted;
}
```

Der obige Sketch nutzt die Funktionen `digitalClockDisplay` und `printDigits` aus Rezept 12.3, d.h., Sie müssen sie kopieren, bevor Sie den Sketch ausführen können. Abbildung 12-2 zeigt, wie Poti und Taster angeschlossen sind.

Alle diese Beispiele geben die Uhrzeit über den seriellen Port aus, aber Sie können auch LEDs oder LCDs verwenden. Der Download für das Grafik-LCD aus Rezept 11.9 enthält Beispiel-Sketches, mit denen sich eine Analoguhr auf dem LCD anzeigen und einstellen lässt.

Die Time-Bibliothek umfasst einige Hilfsfunktionen, die eine Konvertierung aus und in verschiedene Zeitformate ermöglichen. So können Sie zum Beispiel herausfinden, wie viel Zeit seit Beginn dieses Tages schon vergangen ist und wie viel Zeit noch bis zum Ende dieses Tages bleibt.

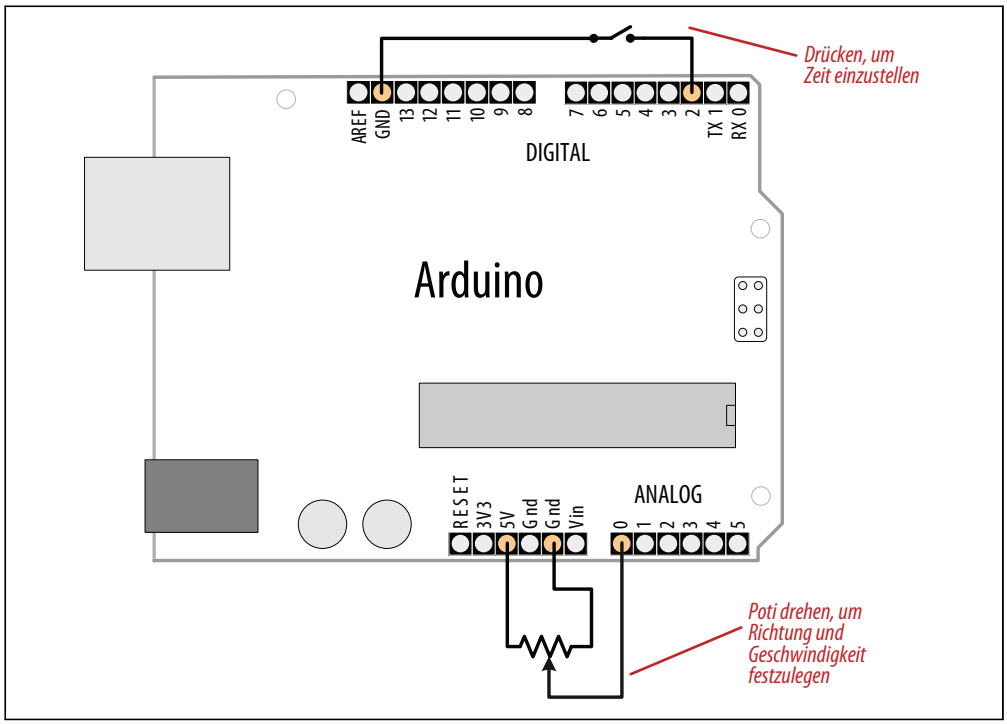


Abbildung 12-2: Poti zur Zeitkorrektur

Eine vollständige Liste finden Sie in *Time.h* im Ordner *libraries*. Weitere Details finden Sie in :

```
dayOfWeek( now() );       // Wochentag (Sonntag ist Tag 1)
elapsedSecsToday( now() );   // Verstrichene Sekunden seit
                // Beginn des Tages
nextMidnight( now() );     // Zeit bis zum Ende des Tages
elapsedSecsThisWeek( now() ); // Verstrichene Zeit seit
                // Beginn der Woche
```

Sie könnten Tage und Monate auch als Textstrings ausgeben. Hier eine Variante der Digitalanzeige, die Tag und Monat als Text ausgibt:

```
void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(dayStr(weekday())); // Wochentag ausgeben
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(monthShortStr(month())); // Monat (abgekürzt) ausgeben
 Serial.print(" ");
```

```
 Serial.print(year());
 Serial.println();
}
```

Siehe auch

Referenz der Arduino Time-Bibliothek: *http://www.arduino.cc/playground/Code/Time*

Wikipedia-Artikel zur Unix-Zeit: *http://de.wikipedia.org/wiki/Unixzeit*

http://www.epochconverter.com/ und *http://www.onlineconversion.com/unix_time.htm* sind zwei beliebte Tools zur Konvertierung der Unix- Zeit.

12.5 Einen Alarm einrichten, um regelmäßig eine Funktion aufzurufen

Problem

Sie wollen eine bestimmte Aktion an bestimmten Tagen und zu bestimmten Uhrzeiten ausführen.

Lösung

TimeAlarms ist eine Bibliothek, die im Download der Time-Bibliothek aus Rezept 12.4 mit enthalten ist (d.h., die Installation der Time-Bibliothek installiert auch TimeAlarms). TimeAlarms macht die Einrichtung zeit- und datumsgesteuerter Alarme einfach:

```
/*
 * TimeAlarmsExample Sketch
 *
 * Dieses Beispiel ruft Alarmfunktionen um 8:30 und 17:45 Uhr auf.
 * Simuliert das Einschalten von Lichtern am Abend und das Ausschalten am Morgen
 *
 * Ein Timer wird alle 15 Sekunden aufgerufen
 * Ein weiterer nur einmal nach 10 Sekunden
 *
 * Beim Start setzen wir die Zeit auf den 1.1.2010, 8:30 Uhr
 */

#include <Time.h>
#include <TimeAlarms.h>

void setup()
{
 Serial.begin(9600);
 Serial.println("TimeAlarms-Beispiel");
 Serial.println("Alarme werden taeglich um 8:30 und 17:45 Uhr ausgeloest");
 Serial.println("Ein Timer wird alle 15 Sekunden ausgeloest");
 Serial.println("Ein anderer nur einmal nach 10 Sekunden");
 Serial.println();
```

```
 setTime(8,29,40,1,1,10); // 8:29:40 Uhr am 1.1.2010

 Alarm.alarmRepeat(8,30,0, MorningAlarm);  // Jeden Tag um 8:30 Uhr
 Alarm.alarmRepeat(17,45,0,EveningAlarm);  // Jeden Tag um 17:45 Uhr

 Alarm.timerRepeat(15, RepeatTask);        // Timer alle 15 Sekunden
 Alarm.timerOnce(10, OnceOnlyTask);        // und einmal nach 10 Sekunden
}

void MorningAlarm()
{
 Serial.println("Alarm: - schalte Licht aus");
}

void EveningAlarm()
{
 Serial.println("Alarm: - schalte Licht ein");
}

void RepeatTask()
{
 Serial.println("15-Sekunden-Timer");
}

void OnceOnlyTask()
{
 Serial.println("Dieser Timer loest nur einmal aus");
}

void  loop()
{
 digitalClockDisplay();
 Alarm.delay(1000); // Zwischen Uhrenanzeige eine Sekunde warten
}

void digitalClockDisplay()
{
 // Digitalanzeige der Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.println();
}

// Hilfsfunktion zur Uhrendarstellung: Gibt
// vorstehenden Doppelpunkt und führende 0 aus
//
void printDigits(int digits)
{
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}
```

Diskussion

Sie können festlegen, dass Aufgaben zu einer bestimmten Tageszeit (wir nennen das *Alarme*) oder in bestimmten Zeitintervallen (sog. *Timer*) ausgeführt werden sollen. Jede dieser Aufgaben kann kontinuierlich oder einmalig ausgeführt werden.

Um einen Alarm zu definieren, der regelmäßig zu einer bestimmten Tageszeit ausgeführt wird, verwenden Sie Folgendes:

```
Alarm.alarmRepeat(8,30,0, MorningAlarm);
```

Das ruft die Funktion `MorningAlarm` jeden Tag um 8:30 Uhr morgens auf.

Soll ein Alarm nur einmal angestoßen werden, verwenden Sie die Methode `alarmOnce`:

```
Alarm.alarmOnce(8,30,0, MorningAlarm);
```

Damit wird die Funktion `MorningAlarm` nur einmal ausgeführt (wenn es wieder 8:30 Uhr ist) und danach nicht wieder.

Timer stoßen Aufgaben nach einem festgelegten Zeitintervall an, nicht zu einer bestimmten Tageszeit. Das Timer-Intervall kann in Sekunden oder in Stunden, Minuten und Sekunden angegeben werden:

```
Alarm.timerRepeat(15, Repeats);       // Timer-Job alle 15 Sekunden ausführen
```

Damit wird die Funktion `Repeats` in Ihrem Sketch alle 15 Sekunden ausgeführt.

Soll ein Timer nur einmal ausgeführt werden, verwenden Sie die Methode `timerOnce`:

```
Alarm.timerOnce(10, OnceOnly);        // Einmal nach 10 Sekunden
```

Das ruft die Funktion `onceOnly` im Sketch 10 Sekunden nach Erzeugung des Timers auf.

Ihr Code muss `Alarm.delay` regelmäßig aufrufen, da diese Funktion den Status aller eingetragenen Ereignisse prüft. Rufen Sie `Alarm.delay` nicht regelmäßig auf, wird auch kein Alarm ausgelöst. Sie können `Alarm.delay(0)` aufrufen, wenn der Scheduler sofort aufgerufen werden soll. Verwenden Sie immer `Alarm.delay` statt `delay`, wenn Sie in einem Sketch TimeAlarms verwenden.

Die TimeAlarms-Bibliothek benötigt die Time-Bibliothek – siehe Rezept 12.4. Es wird keine interne oder externe Hardware für die TimeAlarms-Bibliothek benötigt. Der Scheduler nutzt keine Interrupts, die aufgerufenen Verarbeitungsfunktionen sind genau wie alle anderen Funktion in Ihrem Sketch (Code in Interrupt-Handlern unterliegt Beschränkungen, die in Kapitel 17 diskutiert werden, sie gelten für TimeAlarms-Funktionen aber nicht).

Timer-Intervalle reichen von einer Sekunde bis zu mehreren Jahren. (Wenn Sie Timer-Intervalle unter einer Sekunde brauchen, ist die TimedAction-Bibliothek von Alexander Brevig die bessere Wahl; siehe *http://www.arduino.cc/playground/Code/TimedAction*.)

Die Ausführung von Aufgaben zu bestimmten Zeiten basiert auf der System-Uhr der Time-Bibliothek (Details finden Sie in Rezept 12.4). Wenn Sie die Systemzeit ändern (z.B.

über einen Aufruf von `setTime`), bleiben die Alarmzeiten unangetastet. Nutzen Sie beispielsweise `setTime`, um die Uhr eine Stunde vorzustellen, werden alle Alarme und Timer eine Stunde früher angestoßen. Ist es mit anderen Worten 13:00 Uhr und eine Aufgabe soll in zwei Stunden (um 15:00 Uhr) ausgeführt werden, und Sie stellen die Zeit auf 14:00 Uhr vor, dann wird der Task in einer Stunde ausgeführt. Stellen Sie die Systemzeit zurück – zum Beispiel auf 12:00 Uhr –, wird der Task in drei Stunden angestoßen (wenn es laut Systemuhr 15:00 Uhr ist). Wird die Zeit auf einen Wert gesetzt, der vor der Ausführungszeit des Tasks liegt, wird diese Aufgabe sofort angestoßen (d.h. bei nächsten Aufruf von `Alarm.delay`).

Das Verhalten von Alarmen ist offensichtlich: Aufgaben werden für einen bestimmten Zeitpunkt festgelegt und zu eben diesem Zeitpunkt ausgeführt. Der Effekt von Timern ist möglicherweise nicht ganz so offensichtlich. Wird ein Timer auf fünf Minuten eingestellt und die Uhr um eine Stunde zurückgesetzt, wird dieser Timer erst in einer Stunde und fünf Minuten ausgelöst (selbst wenn es sich um einen periodischen Timer handelt – die Wiederholung wird erst eingerichtet, nachdem er ausgelöst wurde).

Bis zu sechs Alarme und Timer können gleichzeitig ausgeführt werden. Sie können die Bibliothek aber so modifizieren, dass zusätzliche Tasks ausgeführt werden können. Rezept 16.3 zeigt, wie das geht.

`onceOnly`-Alarme und -Timer werden gelöscht, sobald sie angestoßen wurden, können aber so oft Sie wollen wieder neu eingetragen werden, solange insgesamt nicht mehr als sechs verwendet werden. Der folgende Code zeigt, wie ein `timerOnce`-Task erneut eingetragen wird:

```
Alarm.timerOnce(random(10), randomTimer); // Nach zufällig festgelegter
                        // Zeitspanne anstoßen

void randomTimer(){
 int period = random(2,10);        // Neue zufällige Zeitspanne
 Alarm.timerOnce(period, randomTimer); // Neuer zufälliger Timer
}
```

12.6 Eine Echtzeituhr nutzen

Problem

Sie wollen die von einer Echtzeituhr (real-time clock, RTC) bereitgestellte Zeit nutzen. Externe Boards sind üblicherweise batteriebetrieben, d.h., die Zeit ist auch dann korrekt, wenn der Arduino zurückgesetzt oder ausgeschaltet wird.

Lösung

Die einfachste Möglichkeit zur Nutzung einer Echtzeituhr bietet eine Zusatzbibliothek der Time-Bibliothek namens *DS1307RTC.h*. Dieses Rezept eignet sich für die weitverbreiteten DS1307- und DS1337-RTC- ICs:

```
/*
 * TimeRTC Sketch
 * Time-Bibliothek mit Echtzeituhr
 *
 */

#include <Time.h>
#include <Wire.h>
#include <DS1307RTC.h> // Einfache DS1307-Bibliothek, die die Zeit als time_t zurückgibt

void setup() {
 Serial.begin(9600);
  setSyncProvider(RTC.get);  // Diese Funktion liest die Zeit von der Echtzeituhr ein
 if(timeStatus()!= timeSet)
   Serial.println("Synchronisation mit Echtzeituhr fehlgeschlagen");
 else
   Serial.println("Systemzeit ueber Echtzeituhr gesetzt");
}

void loop()
{
  digitalClockDisplay();
  delay(1000);
}

void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(month());
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

// Hilfsfunktion zur Uhrendarstellung: Gibt
// vorstehenden Doppelpunkt und führende 0 aus
//
void printDigits(int digits){
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}
```

Die meisten RTC-Boards für den Arduino nutzen das I2C-Protokoll zur Kommunikation (mehr zu I2C erfahren Sie in Kapitel 13). Verbinden Sie den mit »SCL« (oder »Clock«) gekennzeichneten Anschluss mit dem Arduino-Analogpin 5 und »SDA« (oder »Data«) mit Analogpin 4, wie in Abbildung 12-3 zu sehen. (Die Analogpins 4 und 5 werden für I2C genutzt; siehe Kapitel 13). Achten Sie darauf, die +5V- und Masse-Pins korrekt anzuschließen.

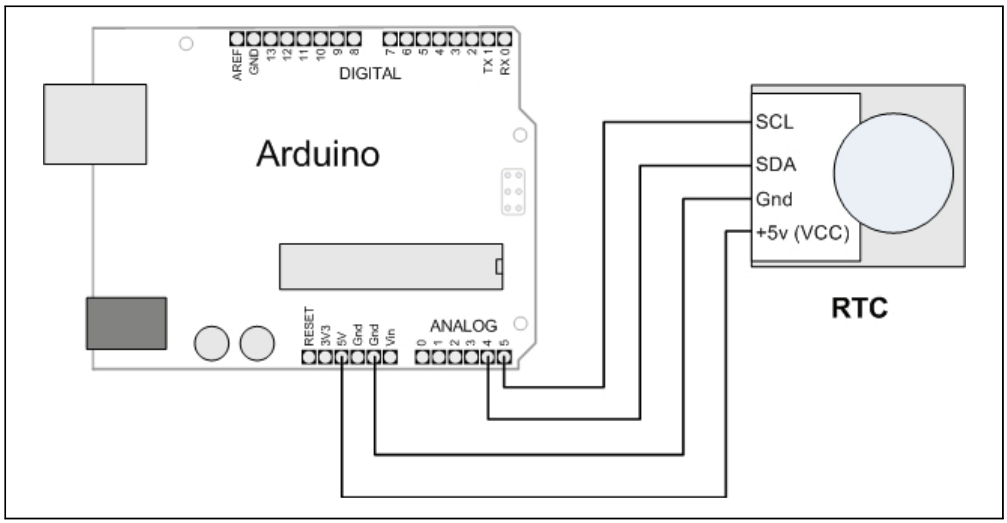


Abbildung 12-3: Anschluss einer Echtzeituhr

Diskussion

Der Code ähnelt den anderen Rezepten, die die Time-Bibliothek nutzen, erhält seine Daten aber über die Echtzeituhr und nicht über den seriellen Port oder einen fest kodierten Wert. Nur eine zusätzliche Zeile ist nötig:

```
setSyncProvider(RTC.get);  // Diese Funktion liest die Zeit von der Echtzeituhr ein
```

Die Funktion `setSyncProvider` teilt der Time-Bibliothek mit, wie sie Informationen zum Setzen (und Aktualisieren) der Zeit bekommt. `RTC.get` ist einer Methode der RTC-Bibliothek, die die aktuelle Zeit in dem Format (Unix-Zeit) zurückliefert, die die Time-Bibliothek nutzt.

Jedes Mal, wenn der Arduino startet, ruft die `setup`-Funktion `RTC.get` auf, um die Zeit über die RTC-Hardware zu setzen.

Bevor Sie die korrekte Zeit vom Modul einlesen können, müssen Sie sie aber erst einmal setzen. Hier ein Sketch, der es Ihnen erlaubt, die Zeit der RTC-Hardware einzustellen. Das ist nur nötig, wenn Sie die RTC zum ersten Mal an eine Batterie anschließen, die Batterie wechseln oder die Zeit ändern müssen:

```
/*
 * TimeRTCSet Sketch
 * Time-Bibliothek mit Echtzeituhr
 *
 * RTC wird entsprechend einer Nachricht vom seriellen Port gesetzt
 * Ein beispielhafter Processing-Sketch zum Setzen der Zeit ist im Download enthalten
 */

#include <Time.h>
#include <Wire.h>
```

```
#include <DS1307RTC.h> // Einfache DS1307-Bibliothek gibt Zeit als time_t zurück

void setup() {
 Serial.begin(9600);
 setSyncProvider(RTC.get);  // Liest Zeit von RTC ein
 if(timeStatus()!= timeSet)
   Serial.println("Synchronisation mit Echtzeituhr fehlgeschlagen");
 else
   Serial.println("Systemzeit über Echtzeituhr gesetzt");
}

void loop()
{
 if(Serial.available())
 {
   time_t t = processSyncMessage();
   if(t >0)
   {
     RTC.set(t);  // RTC und Systemzeit auf empfangenen Wert einstellen
     setTime(t);
   }
 }
  digitalClockDisplay();
  delay(1000);
}

void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(month());
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

// Hilfsfunktion zur Uhrendarstellung: Gibt
// vorstehenden Doppelpunkt und führende 0 aus
//
void printDigits(int digits){
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}

/* Nachrichten zur Zeitsynchronisation vom seriellen Port verarbeiten  */
#define TIME_MSG_LEN  11  // Zeitsynchronisation vom PC ist ein HEADER gefolgt von Unix time_t
                          // als zehn ASCII-Ziffern
#define TIME_HEADER  'T'  // Header-Tag für serielle Zeitsynchronisationsnachricht

time_t processSyncMessage() {
 // Gibt Zeit zurück, wenn eine gültige Sync-Nachricht über den seriellen Port empfangen wurde
```

```
// Die Zeit-Nachricht besteht au einem Header und zehn ASCII-Ziffern
while(Serial.available() >= TIME_MSG_LEN ){
 char c = Serial.read() ;
 Serial.print(c);
 if( c == TIME_HEADER ) {
  time_t pctime = 0;
  for(int i=0; i < TIME_MSG_LEN -1; i++){
   c = Serial.read();
   if( c >= '0' && c <= '9'){
    pctime = (10 * pctime) + (c - '0') ; // Ziffer in Zahl umwandeln
   }
  }
  return pctime;
 }
}
return 0;
}
```

Der Sketch ist nahezu identisch mit dem TimeSerial-Sketch in Rezept 12.4, der die Zeit über den seriellen Port setzt. Hier wird aber die folgende Funktion zum Setzen der RTC aufgerufen, wenn eine Zeit-Nachricht vom Computer empfangen wurde:

```
RTC.set(t);  // RTC und Systemzeit auf empfangenen Wert einstellen

setTime(t);
```

Der RTC-Chip nutzt I2C zur Kommunikation mit dem Arduino. I2C wird in Kapitel 13 erläutert; in Rezept 13.3 finden Sie weitere Details zur I2C-Kommunikation mit dem RTC-Chip.

Siehe auch

Datenblatt zum SparkFun BOB-00099: *http://store.gravitech.us/i2crecl.html*

KAPITEL 13

Kommunikation per I2C und SPI

13.0 Einführung

Die I2C- (Inter-Integrated Circuit) und SPI- (Serial Peripheral Interface) Standards wurden entwickelt, um auf einfache Weise digitale Informationen zwischen Sensoren und Mikrocontrollern wie dem Arduino auszutauschen. Arduino-Bibliotheken für I2C und SPI machen es einfach, diese beiden Protokolle zu verwenden.

Die Wahl zwischen I2C und SPI wird üblicherweise durch die Bauelemente bestimmt, die Sie anschließen wollen. Manche Bauelemente unterstützen beide Standards, doch üblicherweise unterstützt ein Bauelement oder Chip nur den einen oder den anderen.

I2C hat den Vorteil, dass für den Anschluss mit dem Arduino nur zwei Leitungen benötigt werden. Mehrere Bauelemente über die beiden Anschlüsse zu betreiben, ist recht einfach und Sie erhalten Bestätigungen, dass die Signale korrekt empfangen wurden. Die Nachteile sind, dass die Datenraten geringer sind als bei SPI und dass die Daten nur in jeweils eine Richtung fließen können, was die Datenrate noch weiter senkt, wenn eine bidirektionale Kommunikation notwendig ist. Darüber hinaus werden Pullup-Widerstände für die Anschlüsse benötigt, um eine zuverlässige Signalübertragung zu gewährleisten (mehr über Pullups erfahren Sie in der Einführung zu Kapitel 5).

Die Vorteile von SPI sind eine höhere Datenrate und separate Ein- und Ausgänge, so dass gleichzeitig gesendet und empfangen werden kann. Es benötigt nur eine zusätzliche Leitung pro Bauelement (zur Geräteauswahl), d.h., Sie benötigen mehr Anschlüsse, wenn Sie viele Bauelemente anbinden.

Die meisten Arduino-Projekte nutzen SPI-Einheiten für Anwendungen mit hohen Datenraten (z.B. Ethernet und Speicherkarten) und nur einer angeschlossenen Einheit. I2C wird eher für Sensoren verwendet, die nicht so viele Daten senden müssen.

Dieses Kapitel zeigt, wie man I2C und SPI nutzt, um gängige Bauelemente anzuschließen. Es zeigt auch, wie man für Multiboard-Anwendungen mehrere Arduino-Boards über I2C miteinander koppelt.

I2C

Die beide Anschlüsse des I2C-Busses heißen *SCL* und *SDA*. Sie sind bei einem Standard-Arduino-Board über Analogpin 5 für SCL, der das Taktsignal (Clock) liefert, und Analogpin 4 für SDL, der den Datentransfer übernimmt. Beim Mega verwenden Sie Digitalpin 20 für SDA und Pin 21 für SCL. Uno-Boards der Revision 3 besitzen zusätzliche Pins (siehe Rezept 1.2), die die Pins 4 und 5 duplizieren. Wenn Sie ein solches Board besitzen, können Sie sich die Pins aussuchen. Ein Gerät auf dem I2C-Bus wird als *Master* betrachtet. Seine Aufgabe besteht darin, den Informationsaustausch zwischen den anderen angeschlossenen Geräten (*Slaves*) zu koordinieren. Es kann nur einen Master geben und in den meisten Fällen ist der Arduino der Master, der die anderen Chips steuert. Abbildung 13-1 zeigt einen I2C-Master mit mehreren I2C- Slaves.

Mit Arduino 1.0 eingeführte Boards wie das Leonardo-Board duplizieren die SCL- und SDA-Anschlüsse an Pins neben dem AREF-Pin. Die neue Lage der Pins ermöglicht es zukünftigen Boards, die I2C-Anschlüsse immer an der gleichen Stelle zu belassen.

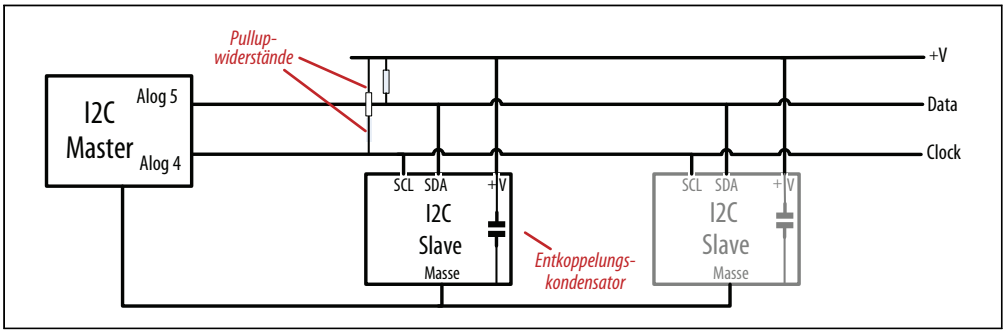


Abbildung 13-1: Ein I2C-Master mit ein oder mehr I2C-Slaves

I2C-Geräte benötigen zur Kommunikation eine gemeinsame Masse. Der Masse-Anschluss des Arduino muss mit der Masse jedes I2C-Gerätes verbunden sein.

Slaves werden über ihre Adresse identifiziert. Jeder Slave muss eine eindeutige Adresse besitzen. Einige I2C-Geräte haben feste Adressen (ein Beispiel ist der Nunchuck in Rezept 13.2), während Sie bei anderen die Adresse konfigurieren können, indem Sie bestimmte Pins auf HIGH oder LOW setzen (siehe Rezept 13.7) oder Initialisierungsbefehle senden.

Arduino nutzt für I2C-Adressen 7-Bit-Werte. Die Datenblätter einiger Geräte verwenden 8-Bit-Adressen. Ist das bei Ihnen der Fall, teilen Sie den Wert durch zwei, um den richtigen 7-Bit-Wert zu bestimmen.

I2C und SPI definieren nur, wie die Kommunikation zwischen den Geräten zu erfolgen hat. Die zu sendenden Nachrichten hängen vom jeweiligen Gerät (und was es macht) ab. Sie müssen auf dem Datenblatt Ihres Geräts nachsehen, welche Befehle für den Betrieb benötigt werden und welche Daten benötigt oder zurückgegeben werden.

Die Arduino Wire-Bibliothek versteckt die gesamte Low-Level-Funktionalität von I2C vor Ihnen und erlaubt das Senden einfacher Befehle zur Initialisierung und Kommunikation von Geräten. Rezept 13.1 ist eine einfache Einführung in diese Bibliothek und ihre Nutzung.

Wire-Code nach Arduino 1.0 migrieren

Die Arduino Wire-Bibliothek wurde in der Release 1.0 geändert. Sie müssen für ältere Releases geschriebene Sketches entsprechend modifizieren und unter 1.0 neu kompilieren. Die Methoden send und receive wurden zugunsten der Konsistenz mit anderen Bibliotheken umbenannt:

Ändern Sie `Wire.send()` in `Wire.write()`.
Ändern Sie `Wire.receive()` in `Wire.read()`.

Bei write müssen Sie nun den Variablentyp für literale Konstanten angeben. Ein Beispiel:

Ändern Sie `Wire.write(0x10)` in `Wire.write((byte)0x10)`.

3,3V-Geräte mit 5V-Boards nutzen

Viele I2C-Geräte sind für den 3,3V-Betrieb ausgelegt und können beschädigt werden, wenn man sie an ein 5V-Arduino-Board anschließt. Sie können so etwas wie das BOB-08745 Breakout-Board von SparkFun verwenden, das den Anschluss über einen Pegelwandler ermöglicht (siehe Abbildung 13-2). Das Pegelwandler-Board hat eine Seite für 3,3V (low-voltage, LV) und eins für 5 Volt (high-voltage, HV).

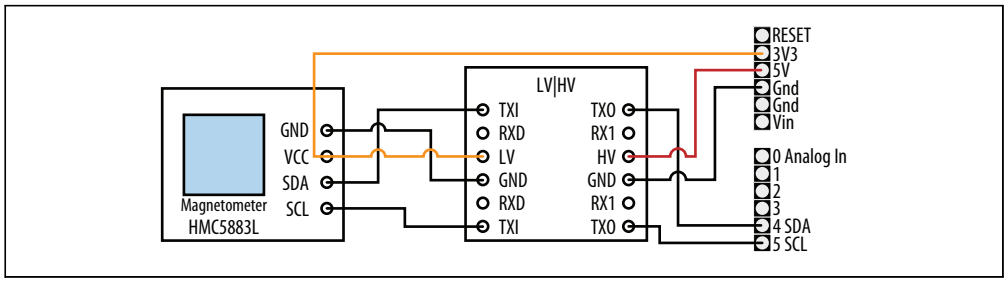


Abbildung 13-2: 3,3V-Gerät mit Pegelwandler betreiben

Für ein 3,3V-I2C-Gerät schließen Sie die LV-Seite wie folgt an:

- Oberen `TXI`-Pin mit I2C SDA-Pin
- Unteren `TXI`-Pin mit I2C SCL-Pin
- `LV`-Pin an I2C VCC und 3,3V-Spannungsversorgung
- `GND`-Pin an I2C-Masse

Die HV-Seite schließen Sie wie folgt an:

- Oberen TXO-Pin mit I2C SDA-Pin
- Unteren TXO-Pin mit I2C SCL-Pin
- HV-Pin mit Arduino 5V-Spannungsversorgung
- GND-Pin mit Arduino-Masse

Sie können mehrere I2C-Geräte mit einem einzelnen Pegelwandler anschließen, wie Abbildung 13-3 zeigt.

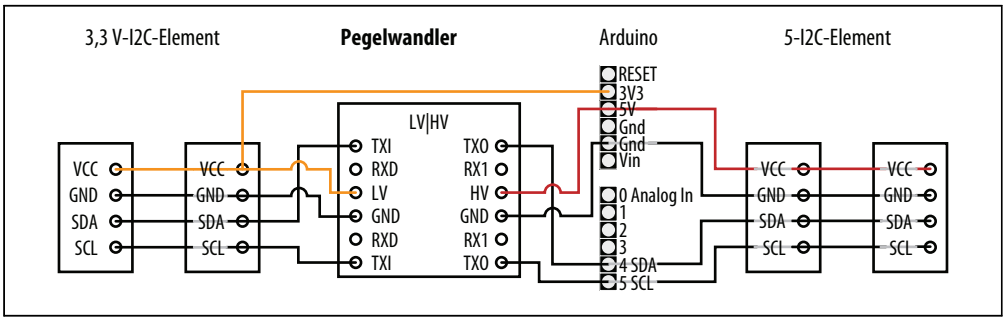


Abbildung 13-3: Anschluss mehrerer 3,3V- und 5V-I2C-Geräte

Beispiele für die Verwendung eines Pegelwandlers finden Sie in der Diskussion zum ITG-3200 in Rezept 6.15 und dem HMC5883 in Rezept 6.16.

SPI

Jüngere Arduino-Releases (seit Release 0019) enthalten eine Bibliothek, die die Kommunikation mit SPI-Geräten erlaubt. SPI besitzt separate Eingangs- (»MOSI«) und Ausgangsleitungen (»MISO«) sowie einen Taktanschluss (Clock). Diese drei Leitungen werden mit den entsprechenden Anschlüssen von ein oder mehr Slaves verbunden. Slaves werden über ein Signal an der Slave-Select-Leitung (SS) identifiziert. Abbildung 13-4 zeigt die Verschaltung bei SPI.

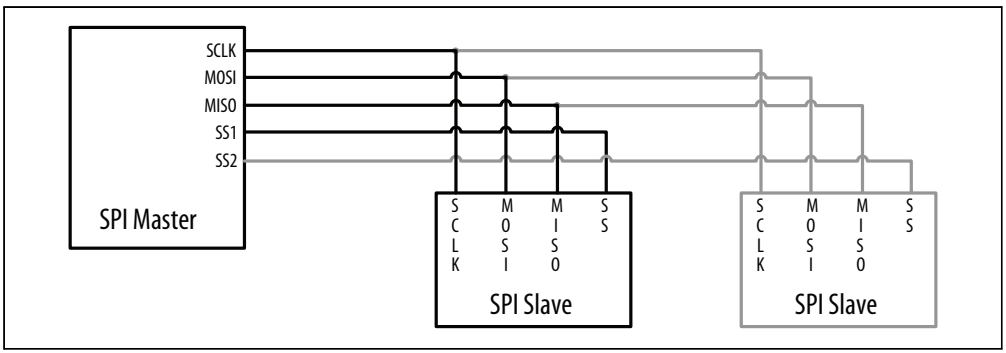


Abbildung 13-4: Signalanschlüsse für SPI-Master und -Slaves

Die für die SPI-Pins zu verwendenden Pin-Nummern sehen Sie in Tabelle 13-1.

Tabelle 13-1: Arduino-Digitalpins für SPI

SPI-Signal	Standard-Arduino-Board	Arduino-Mega
SCLK (clock)	13	52
MISO (data out)	12	50
MOSI (data in)	11	51
SS (slave select)	10	53

Siehe auch

I2C mit SPI vergleichende Application Note: *http://www.maxim-ic.com/app-notes/index.mvp/id/4024*

Referenz der Arduino Wire-Bibliothek: *http://www.arduino.cc/en/Reference/Wire*

Referenz der Arduino SPI-Bibliothek: *http://www.arduino.cc/playground/Code/Spi*

13.1 Steuerung einer RGB-LED mit dem BlinkM-Modul

Problem

Sie wollen I2C-fähige LEDs wie das BlinkM-Modul steuern.

Lösung

BlinkM ist ein vormontiertes Farb-LED-Modul, das einen einfachen Einstieg in I2C ermöglicht.

Verbinden Sie die BlinkM-Pins wie in Abbildung 13-5 gezeigt mit den Analogpins 2 bis 5.

Der folgende Sketch basiert auf Rezept 7.4, steuert aber nicht die Spannung der roten, grünen und blauen LED-Elemente, sondern sendet I2C-Befehle an das BlinkM-Modul. Diese Anweisungen legen die Rot-, Grün- und Blauanteile der zu erzeugenden Farbe fest. Die Funktion `hueToRGB` entspricht der aus Rezept 7.4 und ist an dieser Stelle nicht erneut abgedruckt. Kopieren Sie die Funktion vor der Kompilierung an das Ende des Skripts. (Auf der Website zu diesem Buch finden Sie den kompletten Sketch):

```
/*
* BlinkM Sketch
* Der Sketch geht kontinuierlich den Farbkreis durch
*/

#include <Wire.h>

const int address = 0; // Standard I2C-Adresse für BlinkM
```

```
int color = 0; // Wert zwischen 0 und 255 bestimmt den Farbton
byte R, G, B;  // Rot-, Grün- und Blauanteil

void setup()
{
 Wire.begin(); // I2C-Unterstützung initialisieren

 // Spannung für BlinkM einschalten
 pinMode(17, OUTPUT);   // Pin 17 (Analogausgang 3) versorgt BlinkM mit +5V
 digitalWrite(17, HIGH);
 pinMode(16, OUTPUT);   // Pin 16 (Analogausgang 2) ist Masse
 digitalWrite(16, LOW);
}


void loop()
{
 int brightness = 255; // 255 ist maximale Helligkeit
 hueToRGB(color, brightness);  // Farbton in RGB umwandeln
 // RGB-Werte an BlinkM schreiben

 Wire.beginTransmission(address);// I2C-Kommunikation mit BlinkM einleiten
 Wire.write('c');       // 'c' == Farbe wechseln
 Wire.write(R);        // Wert für Rot
 Wire.write(B);        // Wert für Blau
 Wire.write(G);        // Wert für Grün
 Wire.endTransmission();   // I2C-Bus freigeben

 color++;       // Farbwert erhöhen
 if (color > 255)
   color = 0;
   delay(10);
}
```

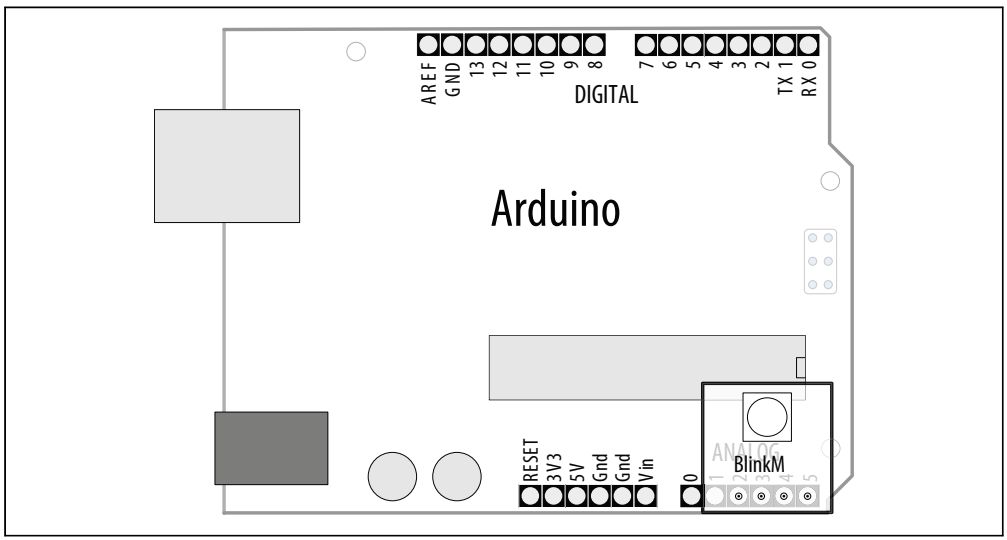


Abbildung 13-5: An Analogpins aufgestecktes BlinkM-Modul

Diskussion

Die Wire-Bibliothek wird wie folgt in den Sketch eingebunden:

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

Weitere Details zur Nutzung von Bibliotheken finden Sie in .

Der Code in `setup` initialisiert die Wire-Bibliothek, die Hardware im Arduino zur Ansteuerung von SCA und SDL an den Analogpins 4 und 5 und schaltet die Pins ein, die zur Spannungsversorgung des BlinkM-Moduls genutzt werden.

Der `loop`-Code ruft die Funktion `hueToRGB` auf, um die Rot-, Grün- und Blauwerte der Farbe zu berechnen.

Die R-, G- und B-Werte werden mit der folgenden Sequenz an BlinkM gesendet:

```
Wire.beginTransmission(address); // Beginn einer I2C-Nachricht für die BlinkM-Adresse
Wire.write('c');      // 'c' ist ein Befehl, der die nachfolgende Farbe einstellt
Wire.write(R);       // Rotanteil
Wire.write(B);       // Blauanteil
Wire.write(G);       // Grünanteil
Wire.endTransmission();  // I2C-Nachricht abgeschlossen
```

Die gesamteDatenübertragung an I2C-Geräte folgt diesem Muster: `beginTransmission`, eine Reihe von `write`-Nachrichten und `endTransmission`.

Versionen vor Arduino 1.0 verwenden `Wire.send` statt `Wire.write`.

I2C unterstützt bis zu 127 Geräte, die mit den Clock- und Datenpins verbunden sind. Die Adresse bestimmt dabei, welches Gerät reagiert. Die Standard-Adresse für BlinkM ist 0, kann aber geändert werden, indem man einen Befehl zur Adressänderung sendet – im BlinkM-Benutzerhandbuch finden Sie Informationen zu allen Befehlen.

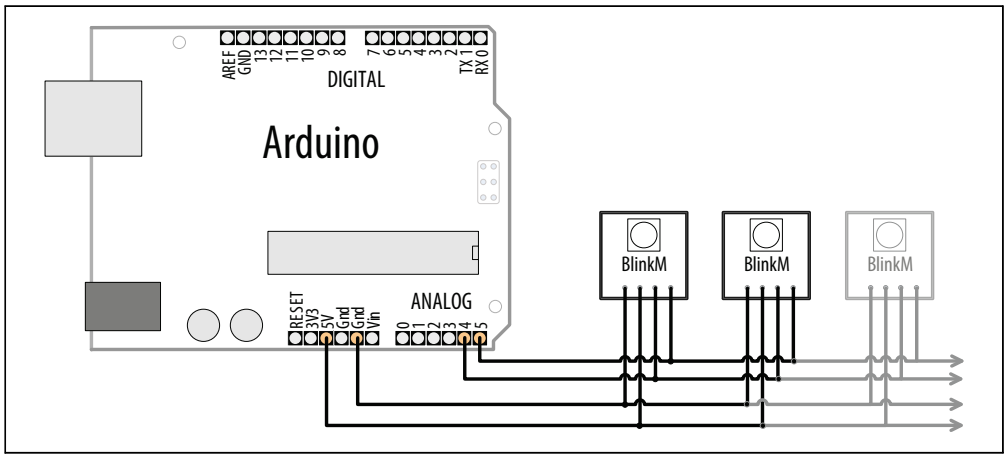


Abbildung 13-6: Anschluss mehrerer BlinkM-Module

Um mehrere BlinkMs anzuschließen, verbinden Sie alle Clock-Pins (»c« am BlinkM, Analogpin 5 am Arduino) und alle Datenpins (»d« am BlinkM, Analogpin 4 am Arduino) wie in Abbildung 13-6 zu sehen. Die Versorgungsanschlüsse sollten mit +5V und Masse des Arduino oder einer externen Spannungsversorgung verbunden sein (da die Analogpins nur wenige Module mit Strom versorgen können).

Jedes BlinkM zieht bis zu 60 mA, d.h., wenn Sie mehr als eine Handvoll verwenden, müssen Sie eine externe Spannungsversorgung nutzen.

Sie müssen jedem BlinkM eine andere I2C-Adresse zuweisen und Sie können den BlinkMTester-Sketch nutzen, der mit den BlinkM-Beispielen geliefert wird, die von *http://code.google.com/p/blinkm-projects/* heruntergeladen werden können.

Kompilieren Sie den BlinkMTester-Sketch und laden Sie ihn hoch. Stecken Sie ein BlinkM-Modul nach dem anderen auf den Arduino auf. (Schalten Sie die Spannung beim Auf- und Abstecken der Module aus.) Verwenden Sie den BlinkMTester Scan-Befehl `s`, um sich die aktuelle Adresse ausgeben zu lassen, und den Befehl `A`, um jedem Modul eine andere Adresse zuzuweisen.

BlinkMTester kommuniziert mit 19.200 Baud, d.h., Sie müssen diese Baudrate im seriellen Monitor einstellen, damit etwas Lesbares auf dem Display erscheint.

Sobald alle BlinkMs eine eindeutige Adresse haben, können Sie die `address`-Variable im obigen Sketch auf die Adresse des BlinkMs setzen, den Sie steuern wollen. Das nachfolgende Beispiel geht von den Adressen 9 bis 11 aus:

```
#include <Wire.h>

int addressA = 9;  // I2C-Adresse für BlinkM
int addressB = 10;
int addressC = 11;

int color = 0; // Wert 0 und 255 bestimmt den Farbton
byte R, G, B;  // Rot-, Grün- und Blauanteil

void setup()
{
 Wire.begin(); // I2C-Unterstützug initialisieren

 // Spannung für BlinkM einschalten
 pinMode(17, OUTPUT);   // Pin 17 (Analogausgang 4) versorgt BlinkM mit +5V
 digitalWrite(17, HIGH);
 pinMode(16, OUTPUT); // Pin 16 (Analogausgang 3) ist Masse
 digitalWrite(16, LOW);
}


void loop()
{
 int brightness = 255; // 255 ist maximale Helligkeit
```

```
 hueToRGB( color, brightness); // Farbton in RGB umwandeln
 // RGB-Werte an jedes BlinkM schreiben
 setColor(addressA, R,G,B);
 setColor(addressB, G,B,R);
 setColor(addressC, B,R,G);

 color++;      // Farbwert erhöhen
 if(color > 255)   // Für gültigen Wert sorgen
   color = 0;
    delay(10);
}

void setColor(int address, byte R, byte G, byte B)
{
 Wire.beginTransmission(address);// I2C-Kommunikation mit BlinkM einleiten
 Wire.write('c');          // 'c' == Farbe ändern
 Wire.write(R);           // Rotanteil
 Wire.write(B);           // Blauanteil
 Wire.write(G);           // Grünanteil
 Wire.endTransmission();     // I2C-Bus freigeben
}

// hueToRGB-Funktion aus obigem Sketch nutzen
```

Die Funktion `setColor` schreibt die übergebenen RGB-Werte an das BlinkM-Modul mit der angegebenen Adresse.

Der Code nutzt die bereits angesprochene `hueToRGB`-Funktion, um einen Integerwert in seine Rot-, Grün- und Blauanteile umzuwandeln.

Siehe auch

Das BlinkM-Benutzerhandbuch: *http://thingm.com/fileadmin/thingm/downloads/BlinkM_datasheet.pdf*

Arduino-Beispielsketches: *http://code.google.com/p/blinkm-projects/*

13.2 Den Wii Nunchuck-Beschleunigungsmesser nutzen

Problem

Sie wollen einen Wii Nunchuck als einfachen und kostengünstigen Beschleunigungsmesser nutzen. Der Nunchuck ist ein weitverbreiteter, kostengünstiger Spiele-Controller, bei dem sich die Orientierung des Controllers mit Hilfe des Gravitationseffekts messen lässt.

Lösung

Der Nunchuck verwendet einen proprietären Stecker. Soll der Nunchuck nicht wieder an die Wii angeschlossen werden, können Sie den Abschluss einfach abschneiden. Alternativ können Sie die Verbindungen (vorsichtig) über eine kleine Lochrasterplatte herstellen

oder einen entsprechenden Adapter von Todbot kaufen (*http://todbot.com/blog/2008/02/18/wiichuck-wii-nunchuck-adapter-available/*).

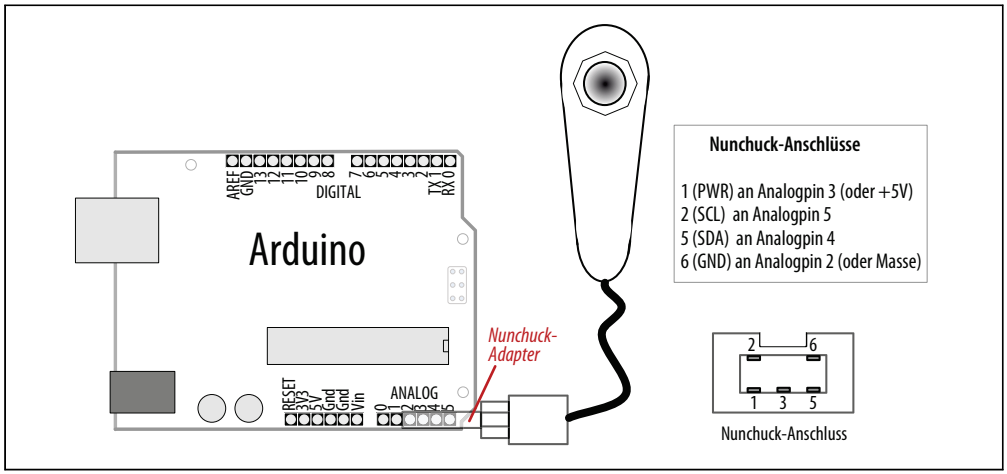


Abbildung 13-7: Anschluss eines Nunchuck

```
/*
 * nunchuck_lines Sketch
 * Sendet Daten an Processing, um Linien zu zeichnen, die der Nunchuck-Bewegung folgen
 */

#include <Wire.h> // Wire initialisieren

const int vccPin = A3;  // +V über Pin 17
const int gndPin = A2;  // Masse über Pin 16

const int dataLength = 6;      // Anzahl anzuforderner Bytes
static byte rawData[dataLength];  // Array zur Speicherung der Nunchuck-Daten

enum nunchuckItems { joyX, joyY, accelX, accelY, accelZ, btnZ, btnC };

void setup() {
 pinMode(gndPin, OUTPUT); // Spannungsversorgung initialisieren
 pinMode(vccPin, OUTPUT);
 digitalWrite(gndPin, LOW);
 digitalWrite(vccPin, HIGH);
 delay(100);  // Auf Stabilisierung warten

 Serial.begin(9600);
 nunchuckInit();
}

void loop(){
 nunchuckRead();
 int acceleration = getValue(accelX);
 if((acceleration >= 75) && (acceleration <= 185))
 {
  //map gibt Werte von 0 bis 63 für Werte von 75 bis 185 zurück
```

```
  byte x = map(acceleration, 75, 185, 0, 63);
  Serial.write(x);
 }
 delay(20); // Zeit zwischen Redraws in Millisekunden
}

void nunchuckInit(){
 Wire.begin();         // Mit I2C-Bus als Master verbinden
 Wire.beginTransmission(0x52);// Übertragung an Gerät 0x52
 Wire.write((byte)0x40);    // Speicheradresse senden
 Wire.write((byte)0x00);    // Eine Null senden
 Wire.endTransmission();    // Übertragung beenden
}

// Daten vom Nunchuck anfordern
static void nunchuckRequest(){
 Wire.beginTransmission(0x52);// Übertragung an Gerät 0x52
 Wire.write((byte)0x00);    // Ein Byte senden
 Wire.endTransmission();    // Übertragung beenden
}

// Daten vom Nunchuck empfangen. Gibt bei Erfolg
// 'wahr' zurück, anderenfalls 'falsch'
boolean nunchuckRead(){
 int cnt=0;
 Wire.requestFrom (0x52, dataLength); // Daten vom Nunchuck anfordern
 while (Wire.available ()) {
  rawData[cnt] = nunchuckDecode(Wire.read());
  cnt++;
 }
 nunchuckRequest();  // Nutzdaten anfordern
 if (cnt >= dataLength)
  return true;   // Erfolgreich, wenn alle 6 Bytes empfangen wurden,
 else
  return false;   //anderenfalls Fehler
}

// Daten in ein Format umwandeln, das die meisten wiimote-Treiber akzeptieren
static char nunchuckDecode (byte x) {
 return (x ^ 0x17) + 0x17;
}

int getValue(int item){
 if (item <= accelZ)
  return (int)rawData[item];
 else if (item == btnZ)
  return bitRead(rawData[5], 0) ? 0: 1;
 else if (item == btnC)
  return bitRead(rawData[5], 1) ? 0: 1;
}
```

Diskussion

I2C wird bei kommerziellen Produkten wie dem Nunchuck häufig zur Kommunikation zwischen den Geräten genutzt. Es gibt kein offizielles Datenblatt für das Gerät, aber die

Nunchuck-Signale wurden analysiert (Reverse Engineering), um die zur Kommunikation benötigten Befehle zu ermitteln.

Sie können den folgenden Processing-Sketch nutzen, um eine Linie auf dem Bildschirm zu zeichnen, die der Bewegung des Nunchuck folgt, wie in Abbildung 13-8 zu sehen (in Kapitel 4 erfahren Sie mehr darüber, wie Sie vom Arduino empfangene serielle Daten mit Processing verarbeiten können. Kapitel 4 zeigt auch, wie man Processing für den Arduino einrichtet und nutzt):

```
// Processing-Sketch zeichnet Linie, die den Nunchuck-Daten folgt

import processing.serial.*;

Serial myPort;  // Serial-Objekt erzeugen
public static final short portIndex = 1;

void setup()
{
 size(200, 200);
 // Verwendeten Port öffnen - siehe Kapitel 4
 myPort = new Serial(this,Serial.list()[portIndex], 9600);
}

void draw()
{
 if ( myPort.available() > 0) {  // Wenn Daten vorhanden sind,
  int y = myPort.read();      // einlesen und speichern
  background(255);          // Weißer Hintergrund
  line(0,63-y,127,y);       // Linie zeichnen
 }
}
```

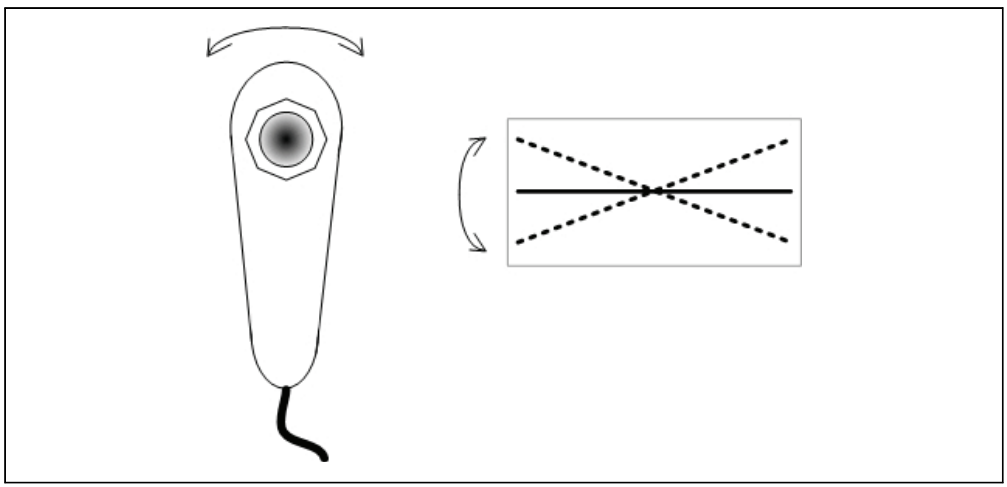


Abbildung 13-8: Nunchuck-Bewegung als geneigte Linie in Processing

Der Sketch bindet die Wire-Bibliothek für die I2C-Kommunikation ein und definiert die Pins, die zur Spannungsversorgung des Nunchuck genutzt werden:

```
#include <Wire.h> // Wire initialisieren

const int vccPin = A3;  // +V (vcc) an Pin 17
const int gndPin = A2;  // Masse an Pin 16
```

Wire.h ist die I2C-Bibliothek, die mit der Arduino-Release mitgeliefert wird. `A3` ist Analogpin 3 (Digitalpin 17), `A2` ist Analogpin 2 (Digitalpin 16). Diese Pins versorgen den Nunchuck mit Spannung.

```
enum nunchuckItems { joyX, joyY, accelX, accelY, accelZ, btnZ, btnC };
```

`enum` ist ein Konstrukt zur Erzeugung enumerierter Listen mit Konstanten, in diesem Fall einer Liste der vom Nunchuck zurückgelieferter Sensorwerte. Diese Konstanten werden genutzt, um Requests für einen der Nunchuck-Sensorwerte zu identifizieren.

`setup` initialisiert die zur Spannungsversorgung des Nunchuck verwendeten Pins, indem es `vccPin` auf `HIGH` und `gndPin` auf `LOW` setzt. Das ist nur notwendig, wenn der Nunchuck-Adapter die Spannungsversorgung übernimmt. Die Verwendung von Digitalpins als Spannungsquelle ist üblicherweise nicht zu empfehlen, solange man nicht sicher ist, dass das versorgte Gerät (wie der Nunchuck) nicht mehr Strom zieht als erlaubt (40 mA; siehe Kapitel 5).

Die Funktion `nunchuckInit` baut die I2C-Kommunikation mit dem Nunchuck auf.

Die I2C-Kommunikation beginnt mit `Wire.begin()`. In diesem Beispiel ist der Arduino als Master für die Initialisierung des Slaves verantwortlich, also des Nunchucks an Adresse 0x52.

Die folgende Zeile weist die Wire-Bibliothek an, das Senden einer Nachricht an das Gerät mit der Hexadezimal-Adresse 52 (0x52) einzuleiten:

```
beginTransmission(0x52);
```

Die I2C-Dokumentation gibt Adressen üblicherweise hexadezimal an. Es ist daher recht bequem, diese Notation auch in Ihrem Sketch zu verwenden.

`Wire.send` legt die übergebenen Werte in einem Puffer der Wire-Bibliothek ab, in dem die Daten zwischengespeichert werden, bis `Wire.endTransmission` aufgerufen wird, um die Daten tatsächlich zu senden.

`nunchuckRequest` und `nunchuckRead` werden genutzt, um Daten vom Nunchuck anzufordern und einzulesen.

Die Wire-Funktion `requestFrom` wird genutzt, um sechs Datenbytes von Gerät 0x52 (dem Nunchuck) einzulesen.

Der Nunchuck gibt seine Daten in sechs Bytes zurück:

Byte Nummer	Beschreibung
Byte 1	Analog-Joystick, Wert der x-Achse
Byte 2	Analog-Joystick, Wert er y-Achse
Byte 3	Beschleunigung x-Achse
Byte 4	Beschleunigung y-Achse
Byte 5	Beschleunigung z-Achse
Byte 6	Button-Zustände und niederwertige Bits der Beschleunigung

`Wire.available` funktioniert wie `Serial.available` (siehe Kapitel 4), gibt also an, wie viele Bytes über das I2C-Interface empfangen wurden. Sind Daten verfügbar, werden sie mit `Wire.read` eingelesen und mit `nunchuckDecode` dekodiert. Die Dekodierung ist nötig, um die gesendeten Werte in Zahlen umzuwandeln, die vom Sketch genutzt werden können. Diese Werte werden in einem Puffer namens `rawData` gespeichert. Ein Request fordert die nächsten sechs Datenbytes an und ist dann für den nächsten Aufruf bereit:

```
int acceleration = getValue(accelX);
```

Der Funktion `getValue` wird eine der Konstanten aus der enumerierten Sensorliste übergeben, in diesem Fall `accelX` für die Beschleunigung an der x-Achse.

Sie können zusätzliche Felder senden, indem Sie sie durch Kommata trennen (siehe Rezept 4.4). Hier eine entsprechend überarbeitete `loop`-Funktion:

```
void loop(){
 nunchuckRead();
 Serial.print("H,"); // Header
 for(int i=0; i < 3; i++)
 {
  Serial.print(getValue(accelX+ i), DEC);
  if( i > 2)
   Serial.write(',');
  else
   Serial.write('\n') ;
 }
 delay(20); // Zeit zwischen Redraws in Millisekunden
}
```

Siehe auch

Rezept 16.5 für eine Nunchuck-Bibliothek und die Diskussion in Rezept 4.4 für einen Processing-Sketch, der ein Echtzeit-Diagramm aller Nunchuck-Werte darstellt.

13.3 Anbindung einer externen Echtzeituhr

Problem

Sie wollen Datum/Uhrzeit einer externen Echtzeituhr (Real-Time Clock, RTC) nutzen.

Lösung

Die Lösung nutzt die Wire-Bibliothek, um auf eine Echtzeituhr zuzugreifen. Sie verwendet die gleiche Hardware wie in Rezept 12.6. Die Verschaltung finden Sie in Abbildung 12-3.

```
/*
* I2C_RTC Sketch
* Beispiel-Code für den Zugriff auf eine Echtzeituhr über die Wire-Bibliothek
*/

#include <Wire.h>

const byte DS1307_CTRL_ID = 0x68; // Adresse der DS1307-Echtzeituhr
const byte NumberOfFields = 7; // Zahl der Felder (Bytes), die
                 // von der Echtzeituhr angefordert werden
int Second ;
int Minute;
int Hour;
int Day;
int Wday;
int Month;
int Year;

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

void loop()
{
 Wire.beginTransmission(DS1307_CTRL_ID);
 Wire.write((byte)0x00);
 Wire.endTransmission();

 // 7 Datenfelder anfordern(Sek, Min, Std, WTag, Dat, Mon, Jhr)
 Wire.requestFrom(DS1307_CTRL_ID, NumberOfFields);

 Second = bcd2dec(Wire.read() & 0x7f);
 Minute = bcd2dec(Wire.read() );
 Hour  = bcd2dec(Wire.read() & 0x3f); // Maske erwartet 24-Stunden-Format
 Wday  = bcd2dec(Wire.read() );
 Day   = bcd2dec(Wire.read() );
 Month  = bcd2dec(Wire.read() );
 Year  = bcd2dec(Wire.read() );
 Year  = Year + 2000; // RTC-Jahr 0 ist Jahr 2000

 digitalClockDisplay(); // Datum/Uhrzeit ausgeben
 delay(1000);
}
```

```
// BCD (Binär kodierte Dezimalzahl) in Dezimal umwandeln
byte bcd2dec(byte num)
{
 return ((num/16 * 10) + (num % 16));
}

void digitalClockDisplay(){
 // Digitalanzeige von Datum/Uhrzeit
 Serial.print(Hour);
 printDigits(Minute);
 printDigits(Second);
 Serial.print(" ");
 Serial.print(Day);
 Serial.print(" ");
 Serial.print(Month);
 Serial.print(" ");
 Serial.print(Year);
 Serial.println();
}

// Hilfsfunktion zur Uhrendarstellung: Gibt
// vorstehenden Doppelpunkt und führende 0 aus
//
void printDigits(int digits){
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}
```

Die `requestFrom`-Methode der Wire-Bibliothek wird verwendet, um sieben Zeitfelder von der Uhr anzufordern (`DS1307_CTRL_ID` ist die Adresse der Uhr):

```
Wire.requestFrom(DS1307_CTRL_ID, NumberOfFields);
```

Die Werte für Datum und Uhrzeit werden mit sieben Aufrufen der `Wire.receive`-Methode abgerufen:

Die vom Modul zurückgelieferten Werte sind binär kodierte Dezimalzahlen (BCD). Wir nutzen daher die Funktion `bcd2dec`, um die Werte beim Empfang umzuwandeln. (BCD speichert Dezimalwerte in vier Datenbits.)

Siehe auch

Rezept 12.6 zeigt, wie man die Uhr setzt.

13.4 Externen EEPROM-Speicher anbinden

Problem

Sie benötigen mehr Permanentspeicher, als der Arduino bereitstellt, und wollen einen externen Speicherchip nutzen, um die Kapazität zu erhöhen.

Lösung

Dieses Rezept nutzt das I2C-fähige serielle EEPROM 24LC128 von Microchip Technology. Abbildung 13-9 zeigt die Verschaltung.

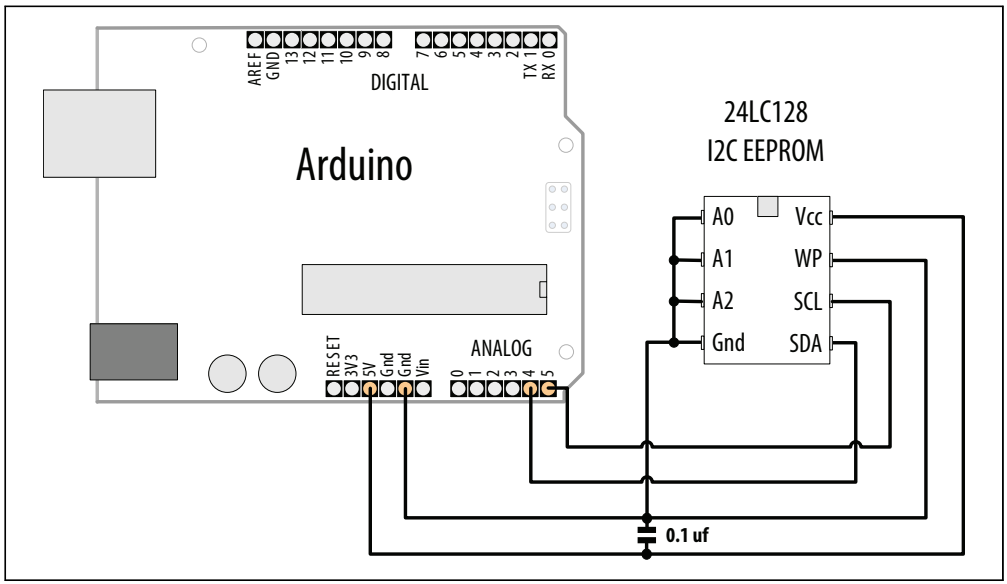


Abbildung 13-9: Anschluss eines I2C-EEPROMs

Das Rezept stellt eine vergleichbare Funktionalität bereit wie die Arduino EEPROM-Bibliothek (siehe Rezept 17.1), verwendet aber ein über I2C angebundenes externes EEPROM, um eine wesentlich höhere Speicherkapazität zur Verfügung zu stellen:

```
/*
 * I2C EEPROM Sketch
 * Version für 24LC128
 */
#include <Wire.h>

const byte EEPROM_ID = 0x50;    // I2C-Adresse für 24LC128-EEPROM

// Erstes sichtbares ASCII-Zeichen ('!') hat den Wert 33
int thisByte = 33;

void setup()
{
 Serial.begin(9600);
 Wire.begin();

 Serial.println("Schreibe 1024 Byte an EEPROM");
 for (int i=0; i < 1024; i++)
 {
  I2CEEPROM_Write(i, thisByte);
  // Weiter mit nächstem Zeichen
  thisByte++;
```

```
  if (thisByte == 126)   // Sie könnten auch "if (thisByte == '~')" verwenden
   thisByte = 33;    // Von vorn anfangen
 }

 Serial.println("Lese 1024 Byte von EEPROM");
 int thisByte = 33;
 for (int i=0; i < 1024; i++)
 {
  char c = I2CEEPROM_Read(i);
  if( c != thisByte)
  {
   Serial.println("Lesefehler");
   break;
  }
  else
  {
   Serial.print(c);
  }
  thisByte++;
  if(thisByte == 126)
  {
   Serial.println();
   thisByte = 33;    // In neuer Zeile von vorn anfangen
  }
 }
 Serial.println();
}

void loop()
{

}

// Diese Funktion entspricht EEPROM.write()
void I2CEEPROM_Write( unsigned int address, byte data )
{
 Wire.beginTransmission(EEPROM_ID);
 Wire.write((int)highByte(address) );
 Wire.write((int)lowByte(address) );
 Wire.write(data);
 Wire.endTransmission();
 delay(5); // Warten, dass I2C-EEPROM den Schreibzyklus abschließt
}

// Diese Funktion entspricht EEPROM.read()
byte I2CEEPROM_Read(unsigned int address )
{
 byte data;
 Wire.beginTransmission(EEPROM_ID);
 Wire.write((int)highByte(address) );
 Wire.write((int)lowByte(address) );
 Wire.endTransmission();
 Wire.requestFrom(EEPROM_ID,(byte)1);
 while(Wire.available() == 0) // Auf Daten warten
  ;
```

```
 data = Wire.read();
 return data;
}
```

Diskussion

Dieses Rezept verwendet den 24LC128, der 128K Bit Speicher hat. Es gibt aber vergleichbare Chips mit höheren und niedrigeren Kapazitäten (der Mikrochip-Link im Siehe-auch-Abschnitt enthält einen entsprechenden Querverweis). Die Adresse des Chips wird über die drei mit A0 bis A2 gekennzeichneten Pins festgelegt und liegt zwischen 0x50 und 0x57, wie in Tabelle 13-2 zu sehen.

Tabelle 13-2: Adressen für 24LC128

A0	A1	A2	Adresse
Gnd	Gnd	Gnd	0x50
+5V	Gnd	Gnd	0x51
Masse	+5V	Masse	0x52
+5V	+5V	Masse	0x53
Masse	Masse	+5V	0x54
+5V	Masse	+5V	0x55
+5V	+5V	Masse	0x56
+5V	+5V	+5V	0x57

Die Verwendung der Wire-Bibliothek entspricht 13.1 und 13.2. In diesen Rezepten können Sie nachlesen, wie die Initialisierung und die Anforderung der Daten von einem I2C-Gerät erfolgt.

Die EEPROM-spezifischen Lese- und Schreiboperationen finden sich in den Funktionen `i2cEEPROM_Write` und `i2cEEPROM_Read`. Die Operationen beginnen mit einem `Wire.beginTransmission` an die I2C-Adresse des Geräts. Dem folgt ein 2-Byte-Wert für die Speicherzelle der Lese-/Schreiboperation. Bei der Schreibfunktion folgt auf die Adresse der zu schreibende Wert – in diesem Beispiel wird an die Speicherzelle ein Byte geschrieben.

Die Leseoperation sendet die Speicherzelle an das EEPROM und dann ein `Wire.requestFrom(EEPROM_ID,(byte)1);`. Das liefert ein Datenbyte von der gerade gesetzten Speicheradresse zurück.

Wenn Sie die Schreibgeschwindigkeit erhöhen wollen, können Sie die Verzögerung von 5ms durch eine Statusprüfung ersetzen, die ermittelt, wann das EEPROM bereit ist, ein weiteres Byte zu schreiben. Siehe hierzu die »Acknowledge Polling«-Technik, die im Abschnitt 7 des Datenblatts beschrieben wird. Daten können nicht nur einzeln, sondern auch in 64 Byte großen »Seiten« geschrieben werden. Details finden Sie in Abschnitt 6 des Datenblatts.

Der Chip merkt sich die angegebene Adresse und bewegt sich bei jeder Lese- oder Schreiboperation zur nächsten Speicherzelle. Wenn Sie mehr als ein Byte einlesen müssen, legen Sie einfach die Startadresse fest und können dann wiederholt Daten anfordern und empfangen.

Die Wire-Bibliothek kann bis zu 32 Byte in einem einzigen Request lesen oder schreiben. Wenn Sie versuchen, mehr einzulesen oder zu schreiben, können Bytes verloren gehen.

Mit dem Pin WP (Write Protect) können Sie den Schreibschutz aktivieren. Er ist hier mit Masse verbunden, damit der Arduino in den Speicher schreiben kann. Wird er an 5V angeschlossen, werden Schreiboperationen unterbunden. Auf diese Weise können Sie persistente Daten in den Speicher schreiben und dann vor versehentlichem Überschreiben schützen.

Siehe auch

Datenblatt zum 24LC128: *http://ww1.microchip.com/downloads/en/devicedoc/21191n.pdf*

Wenn Sie die Schreibgeschwindigkeit erhöhen wollen, können Sie die Verzögerung von 5ms durch eine Statusprüfung ersetzen, die ermittelt, wann das EEPROM bereit ist, ein weiteres Byte zu schreiben. Siehe hierzu die »Acknowledge Polling«-Technik, die im Abschnitt 7 des Datenblatts erläutert wird.

Einen Querverweis auf vergleichbare I2C-EEPROMs mit unterschiedlichen Kapazitäten finden Sie in *http://ww1.microchip.com/downloads/en/DeviceDoc/21621d.pdf*.

Es gibt ein Shield, das Temperatursensor, EEPROM und 7-Segment-Anzeige kombiniert: *http://store.gravitech.us/7segmentshield.html*.

13.5 Temperatur per Digital-Thermometer messen

Problem

Sie wollen die Temperatur messen, vielleicht sogar mit mehr als einem Thermometer, um die Werte an verschiedenen Stellen abgreifen zu können.

Lösung

Das Rezept verwendet den Temperatursensor TMP75 von Texas Instruments. Sie schließen einen einzelnen TMP75 wie in Abbildung 13-10 zu sehen an:

```
/*
 * I2C_Temperature Sketch
 * I2C access the TMP75 digital Thermometer
 */
```

```
#include <Wire.h>

const byte TMP75_ID = 0x49; // Adresse des TMP75
const byte NumberOfFields = 2; // Anzahl anzufordernder Felder (Bytes)

// Höherwertiges Byte der Temperatur (vorzeichenbehafteter Wert in Grad Celsius)
char tempHighByte;
// Niederwertiges Byte der Temperatur  (die Nachkommastellen)
char tempLowByte;

float temperature;  // Temperatur im Fließkomma-Format


void setup() {
 Serial.begin(9600);
 Wire.begin();

 Wire.beginTransmission(TMP75_ID);
 Wire.write(1);       // 1 ist das Konfigurationsregister
 // Standardkonfiguration einstellen, Datenblatt beschreibt die Bedeutung der Konfig-Bits
 Wire.write((byte)0);
 Wire.endTransmission();

 Wire.beginTransmission(TMP75_ID);
 Wire.write((byte)0);  // Zeigerregister auf 0 setzen (die 12-Bit-Temperatur)
 Wire.endTransmission();

}

void loop()
{
  Wire.requestFrom(TMP75_ID, NumberOfFields);
  tempHighByte = Wire.read();
  tempLowByte = Wire.read();
  Serial.print("Integer-Temperatur ist ");

  Serial.print(tempHighByte, DEC);
  Serial.print(",");

  // Die unteren 4 Bit von LowByte enthalten die fractional-Temperatur
  int t = word( tempHighByte, tempLowByte) / 16 ;
  temperature = t / 16.0; // In Fließkommazahl umwandeln
  Serial.println(temperature);
  delay(1000);
}
```

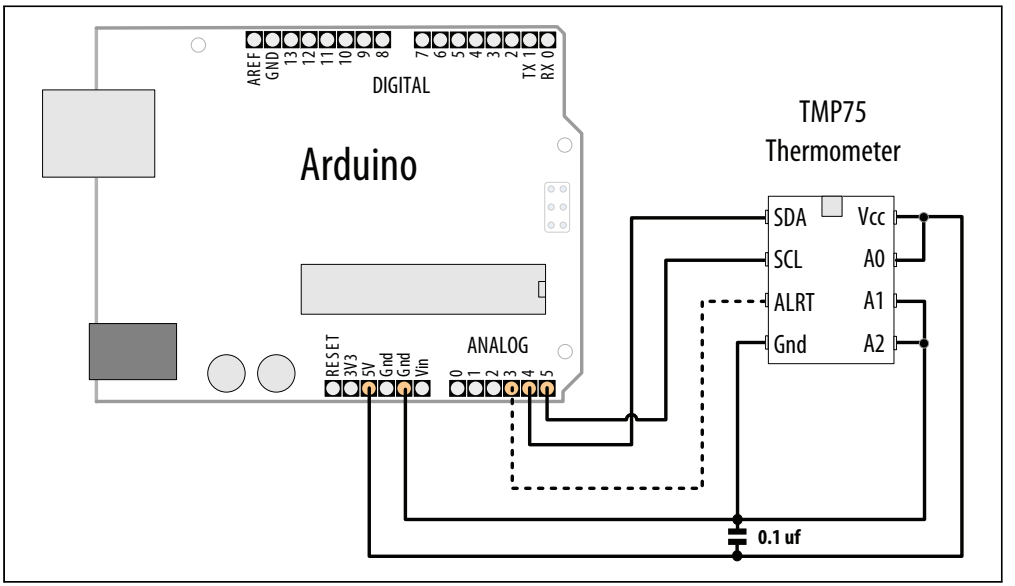


Abbildung 13-10: TMP75 I2C-Thermometer

Diskussion

Wie bei allen I2C-Bauelemente in diesem Kapitel erfolgt die Kommunikation über die beiden SCL- und SDA-Pins. Auch Spannung und Masse müssen angeschlossen werden, um das Bauelement mit Strom zu versorgen.

`Setup` sendet Daten zur Konfiguration des normalen Betriebs – es gibt eine Reihe von Optionen für spezialisierte Anwendungen (Interrupts, Energiesparmodus etc.), aber wir verwenden hier den normalen Modus mit einer Auflösung von 0,5°C.

Um die Temperatur einzulesen, fordert der Arduino (als Master) im `loop`-Code vom Slave (mit der Adresse `TMP75_ID`) zwei Datenbytes an:

```
Wire.requestFrom(TMP75_ID, NumberOfFields);
```

`Wire.read` ruft die beiden Bytes ab (auf dem Datenblatt wird detailliert beschrieben, wie Daten vom Gerät angefordert werden können):

```
tempHighByte = Wire.read();
tempLowByte = Wire.read();
```

Das erste Byte ist der Integerwert der Temperatur in Grad Celsius. Das zweite Byte enthält vier signifikante Bits mit der Temperatur.

Die beiden Bytes werden in ein 16-Bit-Wort umgewandelt (siehe Kapitel 3) und dann verschoben, um einen 12-Bit-Wert zu bilden. Da die ersten vier Bits die Temperatur darstellen, wird der Wert erneut um vier Bit verschoben, um den Fließkommawert zu ermitteln.

Der TMP75 kann mit acht unterschiedlichen Adressen konfiguriert werden, wodurch sich bis zu acht Geräte am gleichen Bus betreiben lassen (siehe Abbildung 13-11). Der Sketch verwendet die I2C-Adresse 0x49 (der TMP75-Adress-Pin A ist mit +5V verbunden, A1 und A2 mit Masse). Tabelle 13-3 zeigt die Anschlüsse für die acht Adressen.

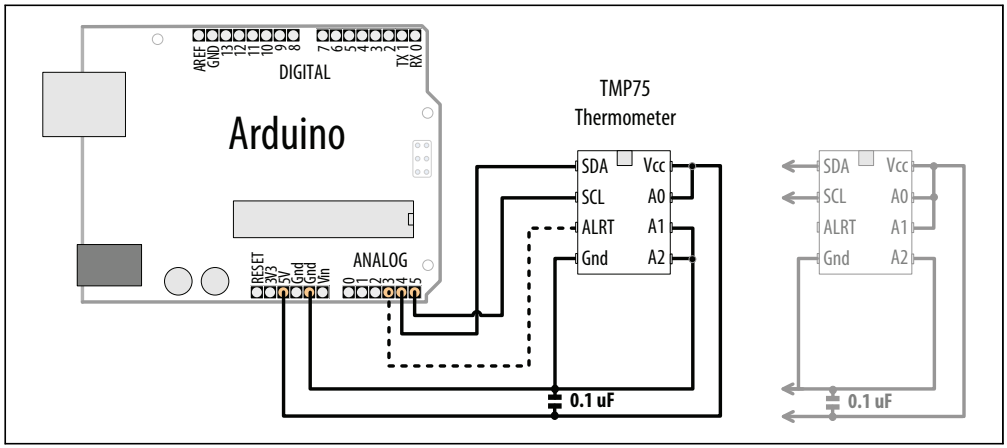


Abbildung 13-11: Paralleler Anschluss mehrerer Geräte mit verschiedenen Adressen über SDA und SCL

Tabelle 13-3: Address values for TMP75

A0	A1	A2	Adresse
Masse	Masse	Masse	0x48
+5V	Masse	Masse	0x49
Masse	+5V	Masse	0x4A
+5V	+5V	Masse	0x4B
Masse	Masse	+5V	0x4C
+5V	Masse	+5V	0x4D
+5V	+5V	Masse	0x4E
+5V	+5V	+5V	0x4F

Werden mehrere I2C-Geräte angeschlossen, verbindet man alle SDA- und alle SCL-Leitungen miteinander. Jedes Gerät wird mit der Spannungsversorgung verbunden und nutzt einen 0,1µF-Parallelkondensator. Die Masseleitungen müssen miteinander verbunden sein, selbst wenn separate Spannungsversorgungen (z.B. Batterien) verwendet werden.

Der folgende Sketch gibt die Temperatur zweier Geräte aus, die benachbarte Adressen (beginnend bei 0x49) verwenden:

```
#include <Wire.h>

const byte TMP75_ID = 0x49; // Adresse des ersten TMP75

const byte NumberOfFields  = 2; // Anzahl anzufordernder Felder (Bytes)
const byte NumberOfDevices = 2; // Anzahl TMP75s
```

```
char tempHighByte; // Höherwertiges Byte der Temperatur
                   // (vorzeichenbehafteter Wert in Grad
                   // Celsius)

char tempLowByte;  // Niederwertiges Byte der Temperatur
                   // (die Nachkommastellen)

float temperature; // Temperatur im Fließkomma-Format

void setup() {
 Serial.begin(9600);
 Wire.begin();

 for (int i=0; i < NumberOfDevices; i++)
 {
  Wire.beginTransmission(TMP75_ID+i);
  Wire.write(1);
  // Standardkonfiguration einstellen, Datenblatt beschreibt die Bedeutung der Konfig-Bits
  Wire.write((byte)0);
  Wire.endTransmission();

  Wire.beginTransmission(TMP75_ID+i);
  Wire.write((byte)0);  // Zeigerregister auf 0 setzen (die 12-Bit-Temperatur)
  Wire.endTransmission();
 }
}

void loop()
{
 for (int i=0; i < NumberOfDevices; i++)
 {
  byte id = TMP75_ID + i;  // Adressen liegen nebeneinander
  Wire.requestFrom(id, NumberOfFields);
  tempHighByte = Wire.read();
  tempLowByte = Wire.read();
  Serial.print(id,HEX); // Geräteadresse ausgeben
  Serial.print(": Integer-Temperatur ist ");
  Serial.print(tempHighByte, DEC);
  Serial.print(",");

  // Die unteren 4 Bit von LowByte enthalten die fractional-Temperatur
  int t = word( tempHighByte, tempLowByte) / 16 ;
  temperature = t / 16.0; // In Fließkomma umwandeln
  Serial.println(temperature);
 }
 delay(1000);
}
```

Sie können weitere Geräte hinzufügen, wenn Sie die Zahl der Geräte in `NumberOfDevices` anpassen und aufeinanderfolgende Adressen verwenden (die in diesem Beispiel bei 0x49 beginnen).

Der Alert-Anschluss (Pin 3) kann so programmiert werden, dass er ein Signal liefert, wenn die Temperatur einen Schwellwert erreicht. Details zu diesem Feature finden Sie auf dem Datenblatt.

Siehe auch

Das TMP75-Datenblatt: *http://focus.ti.com/docs/prod/folders/print/tmp75.html*

In Rezept 3.15 erfahren Sie mehr über die `word`-Funktion.

13.6 Vier 7-Segment-LEDs mit nur zwei Leitungen steuern

Problem

Sie wollen eine mehrstellige 7-Segment-Anzeige nutzen und müssen die Zahl der benötigten Arduino-Pins minimieren.

Lösung

Dieses Rezept nutzt das Gravitech 7-Segment-Display-Shield, das einen SAA1064 I2C-nach-7-Segment-Treiber von Philips verwendet (siehe Abbildung 13-12).

Der folgende einfache Sketch schaltet nacheinander jedes Segment aller Anzeigen ein:

```
/*
 * I2C_7Segment Sketch
 */

#include <Wire.h>

const byte LedDrive = 0x38;  // I2C-Adresse der 7-Segment-Anzeige

int segment,decade;

void setup() {
 Serial.begin(9600);
 Wire.begin();      // An I2C-Bus anbinden

 Wire.beginTransmission(LedDrive);
 Wire.write((byte)0);
 Wire.write(B01000111); // Ziffern 1 bis 4 mit maximalem Treiberstrom nutzen
 Wire.endTransmission();
}

void loop()
{
 for (segment = 0; segment < 8; segment++)
 {
  Wire.beginTransmission(LedDrive);
  Wire.write(1);
  for (decade = 0 ; decade < 4; decade++)
  {
   byte bitValue = bit(segment);
   Wire.write(bitValue);
  }
```

```
    Wire.endTransmission();
    delay (250);
  }
}
```

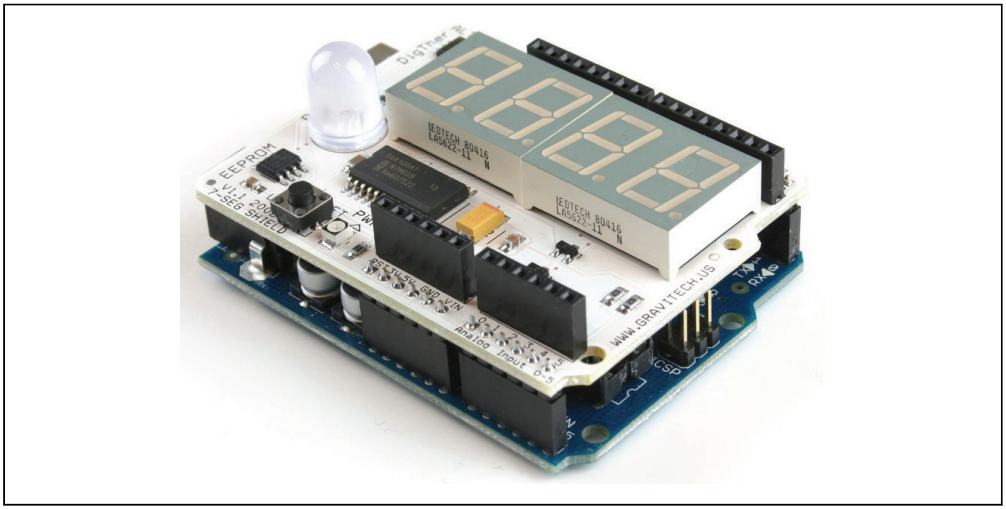


Abbildung 13-12: Gravitech I2C-Shield

Diskussion

Der SAA1064-Chip (an Adresse 0x38) wird in `setup` initialisiert. Der verwendete Wert konfiguriert den Chip so, dass alle vier Anzeigen mit maximalem Strom angesteuert werden (Details zur Konfiguration finden Sie im Datenblatt-Abschnitt zu den Steuerbits).

Der `loop`-Code aktiviert nacheinander jedes Segment aller Anzeigen. Der Befehl `Wire.send(1);` teilt dem Chip mit, dass das nächste empfangene Byte die erste Anzeige und nachfolgende Bytes die nachfolgenden Anzeigen ansteuern.

Zu Beginn wird der Wert `1` viermal gesendet und der Chip aktiviert das A-Segment (oben) aller vier Anzeigen. (In Kapitel 2 erfahren Sie mehr über die `bit`-Funktion.)

Der Wert von `segment` wird in der `for`-Schleife inkrementiert, wodurch `bitValue` so verschoben wird, dass das nächste LED-Segment eingeschaltet wird.

Jede Bit-Position entspricht einem Segment der Anzeige. Die Werte dieser Bit-Positionen lassen sich so kombinieren, dass mehr als ein Segment eingeschaltet wird.

Der folgende Sketch zählt von 0 bis 9999. Im Array `lookup[10]` finden Sie die Werte, die benötigt werden, um die Ziffern 0 bis 9 in einem Segment anzuzeigen:

```
#include <Wire.h>

const byte LedDrive = 0x38;  // I2C-Adresse der 7-Segment-Anzeige

// Lookup-Array mit den für die jeweilige Ziffer zu aktivierenden Segmenten
```

```
const int lookup[10] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

int count;

void setup()
{
 Wire.begin();     // Mit I2C-Bus verbinden (Adresse für Master optional)
}

void loop()
{
 Wire.beginTransmission(LedDrive);
 Wire.write((byte)0);
 Wire.write(B01000111); // 7-Segment-Treiber initialisieren - siehe Datenblatt
 Wire.endTransmission();

 // Zahlen von 0 bis 9999 ausgeben
 for (count = 0; count <= 9999; count++)
 {
  displayNumber(count);
  delay(10);
 }
}

// Bis zu vier Ziffern auf einer 7-Segment-I2C-Anzeige ausgeben
void displayNumber( int number)
{
 number = constrain(number, 0, 9999);
 Wire.beginTransmission(LedDrive);
 Wire.write(1);
 for(int i =0; i < 4; i++)
 {
  byte digit = number % 10;
  {
   Wire.write(lookup[digit]);
  }
  number = number / 10;
 }
 Wire.endTransmission();
}
```

Der Funktion `displayNumber` wird die auszugebende Zahl übergeben. Der für jedes Segment zu sendende Wert in der `for`-Schleife wird in zwei Schritten verarbeitet. Zuerst wird die Ziffer bestimmt, indem man den Rest ermittelt, nachdem die Zahl durch 10 dividiert wurde. Dieser Wert (eine Ziffer zwischen 0 und 9) wird genutzt, um das Bitmuster aus dem `lookup[]`-Array abzurufen, in dem die zur Darstellung der Ziffer benötigten Segmente stehen.

Jede nachfolgende Ziffer wird bestimmt, indem man die Zahl durch 10 teilt und dann den Rest ermittelt. Sobald der Rest 0 ist, wurden alle Ziffern gesendet.

Sie können *führende Nullen* (unnötige Nullen vor den Ziffern) unterdrücken, indem Sie `displayNumber` wie folgt anpassen:

```
// Bis zu vier Ziffern auf einer 7-Segment-I2C-Anzeige ausgeben
void displayNumber( int number)
```

```
{
 number = constrain(number, 0, 9999);
 Wire.beginTransmission(LedDrive);
 Wire.write(1);
 for(int i =0; i < 4; i++)
 {
  byte digit = number % 10;
  // Hier wird auf führende Nullen geprüft
  if ((number == 0) && (i > 0)) {
    Wire.write((byte)0); // Alle Segmente ausschalten, um führende Nullen zu unterdrücken
  }
  else {
    Wire.write(lookup[digit]);
  }
  number = number / 10;
 }
 Wire.endTransmission();
}
```

Die folgende Anweisung prüft, ob der Wert 0 ist, und stellt sicher, dass es sich nicht um die erste (niederwertigste) Ziffer handelt:

```
if ((number == 0) && (i > 0))
  Wire.write((byte)0); // Alle Segmente ausschalten, um führende Nullen zu unterdrücken
```

Ist das der Fall, wird eine 0 gesendet und alle Segmente für die Ziffer werden ausgeschaltet. Damit werden führende Nullen unterdrückt, aber eine einzelne Null ausgegeben, wenn dieser Wert an die Funktion übergeben wird.

Der Ausdruck `(byte)0` wird in der `Wire.write`-Anweisung benötigt, damit der Compiler weiß, dass die Null ein Bytewert sein soll. Lassen Sie das weg, erhalten Sie die Fehlermeldung »call of overloaded 'write(int)' is ambiguous«, was bedeutet, dass der Compiler sich nicht entscheiden kann, welche write-Methode er aufrufen soll,

Siehe auch

SAA1064-Datenblatt: *http://www.nxp.com/documents/data_sheet/SAA1064_CNV.pdf*

Es gibt ein Shield, das Temperatursensor, EEPROM und 7-Segment-Anzeige kombiniert: *http://store.gravitech.us/7segmentshield.html*.

13.7 Einen I2C-Port-Expander integrieren

Problem

Sie wollen mehr Ein-/Ausgabeports nutzen, als Ihr Board zur Verfügung stellt.

Lösung

Sie können einen externen Port-Expander wie den PCF8574A nutzen, der acht Ein-/Ausgangspins besitzt, die über I2C angesteuert werden können. Der folgende Sketch erzeugt eine Balkenanzeige mit acht LEDs. Abbildung 13-13 zeigt die Verschaltung.

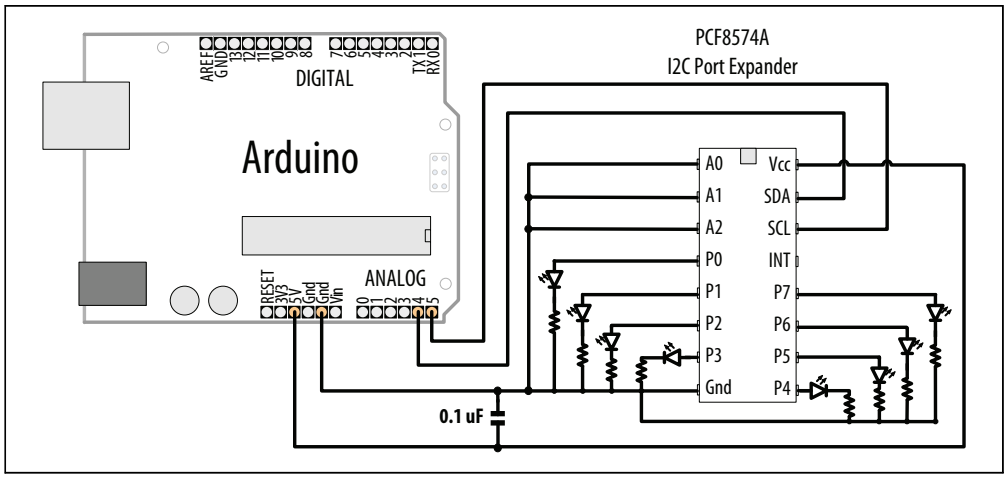


Abbildung 13-13: PCF8574A Port-Expander steuert acht LEDs

Der Sketch hat die gleiche Funktionalität wie in Rezept 7.5, verwendet aber zur Ansteuerung der LEDs einen I2C-Port-Expander, so dass nur zwei Pins benötigt werden:

```
/*
* I2C_7segment
* Nutzt I2C-Port zur Steuerung einer Balkenanzeige
* Aktiviert eine Reihe von LEDs proportional zum Wert eines Analog-Senors
* Siehe Rezept 7.5
*/

#include <Wire.h>

//Adresse für PCF8574. Anschluss wie in Abbildung 13-13
const int address = 0x38;
const int NbrLEDs = 8;

const int analogInPin = 0; // Analogeingang für
                // variablen Widerstand

int sensorValue = 0;     // Vom Sensor eingelesener Wert
int ledLevel = 0;       // In LED-Balkenanzeige umgewandelter Wert
int ledBits = 0;       //  Bits für jede LED werden auf 1 gesetzt, um die LED einzuschalten

void setup()
{
 Wire.begin(); // I2C initialisieren
 Serial.begin(9600);
}
```

```
void loop() {
 sensorValue = analogRead(analogInPin);        // Analogwert einlesen
 ledLevel = map(sensorValue, 0, 1023, 0, NbrLEDs); // Auf Anzahl LEDs abbilden
 for (int led = 0; led < NbrLEDs; led++)
 {
  if (led < ledLevel ) {
   bitWrite(ledBits,led, HIGH);   // LED unter Pegel einschalten
  }
  else {
   bitWrite(ledBits,led, LOW);   // LED über Pegel ausschalten
  }
  // Wert an I2C senden
  Wire.beginTransmission(address);
  Wire.write(ledBits);
  Wire.endTransmission();
 }
 delay(100);
}
```

Diskussion

Die Widerstände müssen 220 Ohm oder mehr haben (in Kapitel 7 wird beschrieben, wie man Widerstände wählt).

Der PCF8574A kann LEDs nur mit geringeren Strömen als der Arduino treiben. Wenn Sie höhere Ströme brauchen (Details finden Sie auf dem Datenblatt), finden Sie in Rezept 13.8 ein geeigneteres Bauelement.

Sie können die Adresse über die Adresspins A0, A1 und A2 ändern (siehe Tabelle 13-4).

Tabelle 13-4: Address values for PCF8574A

A0	A1	A2	Address
Masse	Masse	Masse	0x38
+5V	Masse	Masse	0x39
Masse	+5V	Masse	0x3A
+5V	+5V	Masse	0x3B
Masse	Masse	+5V	0x3C
+5V	Masse	+5V	0x3D
+5V	+5V	Masse	0x3E
+5V	+5V	+5V	0x3F

Sie können den Port-Expander auch als Eingang nutzen. Ein Byte lesen Sie wie folgt ein:

```
Wire.requestFrom(address, 1);
if(Wire.available())
{
 data = Wire.receive();
 Serial.println(data,BIN);
}
```

Siehe auch

PCF8574-Datenblatt: *http://www.nxp.com/documents/data_sheet/PCF8574.pdf*

13.8 Mehrstellige 7-Segment-Anzeigen über SPI ansteuern

Problem

Sie wollen 7-Segment-Anzeigen ansteuern, ohne zu viele Pins nutzen zu müssen.

Lösung

Dieses Rezept bietet eine Funktionalität wie Rezept 7.12, benötigt aber nur drei Ausgangspins. Der Text beschreibt die SPI-Befehle, die zur Kommunikation mit dem MAX7221 genutzt werden (Abbildung 13-14 zeigt die Verschaltung):

```
/*
 * SPI_Max7221_0019
 */

#include <SPI.h>

const int slaveSelect = 10; // Zur Aktivierung es aktiven Slaves verwendeter Pin

const int numberOfDigits = 2; // Anzahl angeschlossener Ziffern
const int maxCount = 99;

int count = 0;

void setup()
{
 SPI.begin();  // SPI initialisieren
 pinMode(slaveSelect, OUTPUT);
 digitalWrite(slaveSelect,LOW);  // Slave wählen
 // 7221 auf Anzeige von 7-Segment-Daten vorbereiten - siehe Datenblatt
 sendCommand(12,1);  // Normaler Modus (voreingestellt ist Shutdown-Modus);
 sendCommand(15,0);  // Display-Test aus
 sendCommand(10,8);  // Mittlere Helligkeit (Wertebereich ist 0-15)
 sendCommand(11,numberOfDigits);  // Zahl der Anzeigen festlegen
 sendCommand(9,255); // Dekodierungsart; wir verwenden Standard-7-Segment-Anzeigen
 digitalWrite(slaveSelect,HIGH);  // Slave deaktivieren
}

void loop()
{
 displayNumber(count);
 count = count + 1;
 if (count > maxCount)
  count = 0;
 delay(100);
}

// Ausgabe von bis zu vier Ziffern auf 7-Segment-Anzeige
```

```
void displayNumber( int number)
{
 for (int i = 0; i < numberOfDigits; i++)
 {
  byte character = number % 10;  // Wert der Ziffer ganz rechts ermitteln
  // Segmentnummer als Befehl senden; erstes Segment ist Befehl1
  sendCommand(numberOfDigits-i, character);
  number = number / 10;
 }
}

void sendCommand( int command, int value)
{
 digitalWrite(slaveSelect,LOW); // Chip-Select ist aktiv Low
 // 2-Byte-Datentransfer zum 7221
 SPI.transfer(command);
 SPI.transfer(value);
 digitalWrite(slaveSelect,HIGH); // Chip freigeben, Übertragungsende
}
```

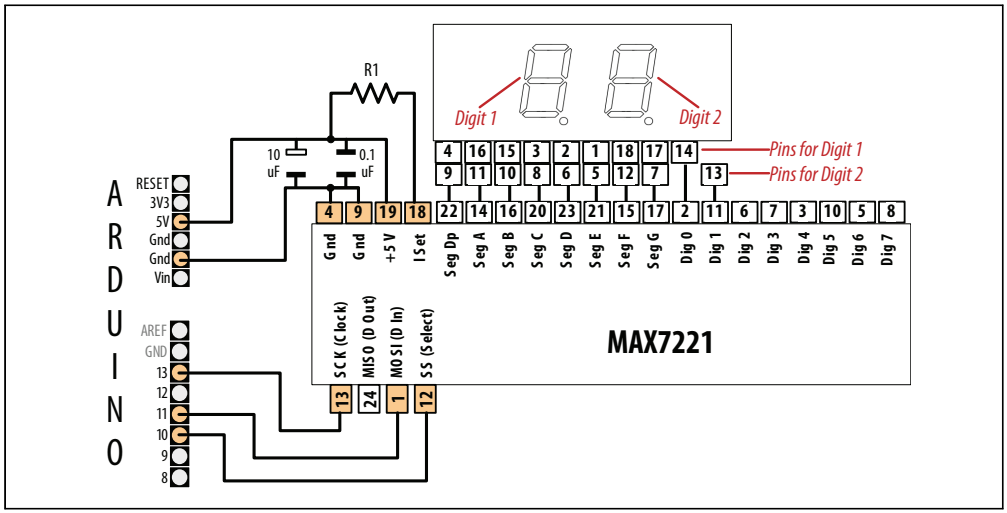


Abbildung 13-14: Anschluss eines MAX7221 mit Lite-On LTD-6440G

Diskussion

Der MAX7221 benötigt LEDs mit gemeinsamer Kathode. Das Pinout in Abbildung 13-14 ist für die Lite-On LTD-6440G, eine zweistellige 7-Segment-LED. Die Segmente jeder Ziffer müssen miteinander verbunden werden. Zum Beispiel liegt der Dezimalpunkt für die erste Ziffer an Pin 4 und für die zweite Ziffer an Pin 9. Wie in der Abbildung zu sehen, sind die Pins 4 und 9 miteinander verbunden und an den MAX7221-Pin 22 angeschlossen.

Der MAX7221 kann bis zu acht Segmente ansteuern (oder eine 8 × 8-Matrix). Die Steuerung erfolgt über Befehle, die festlegen, welches LED-Segment eingeschaltet werden soll.

Nach der Initialisierung der Bibliothek wird der SPI-Code in der Funktion `sendCommand` zusammengefasst. SPI verwendet den mit dem Chip verbundenen Select-Slave-Anschluss und der Chip wird aktiviert, indem man diesen Pin auf `LOW` setzt. Alle SPI-Befehle werden dann von diesem Chip empfangen, bis der Pin wieder auf `HIGH` gesetzt wird. `SPI.transfer` ist die Bibliotheksfunktion zum Senden einer SPI-Nachricht. Sie besteht aus zwei Teilen: einem numerischen Code, der angibt, welches Register die Nachricht empfangen soll, gefolgt von den eigentlichen Daten. Details zu jedem SPI-Gerät finden Sie auf dem Datenblatt.

`Setup` initialisiert den 7221, indem er Befehle zum Aufwachen (der Chip startet in einem Stromsparmodus) sendet, die Helligkeit anpasst, die Anzahl der Ziffern festlegt und die Dekodierung für 7-Segment-Anzeigen aktiviert. Jeder Befehl besteht aus einer Befehls-ID (die auf dem Datenblatt *Register* genannt wird) und einem Wert für diesen Befehl.

Zum Beispiel dient Befehl (Register) 10 der Helligkeit, d.h., der folgende Befehl stellt eine mittlere Helligkeit ein (der Wertebereich liegt zwischen 0 und 15):

```
sendCommand(10,8); // Mittlere Helligkeit einstellen
```

Die Befehle 1 bis 8 werden zur Steuerung der Anzeigen verwendet. Der folgende Code aktiviert die Segmente, die die Ziffer 5 in der ersten (ganz linken) Anzeige darstellen. Beachten Sie, dass die Anzeigennummern auf dem Datenblatt (und in Abbildung 13-14) bei 0 beginnen, d.h., Sie müssen daran denken, dass Sie die Anzeige 0 mit dem Befehl 1 steuern, Anzeige1 mit dem Befehl 2 und so weiter:

```
sendCommand(1, 5); // 5 in der ersten Anzeige ausgeben
```

Sie können führende Nullen unterdrücken, indem Sie zwei Codezeilen in `displayNumber` einfügen und 0xf an den 7221 senden, um alle Segmente zu löschen, wenn der Wert tatsächlich 0 ist:

```
void displayNumber( int number)
{
 for (int i = 0; i < numberOfDigits; i++)
 {
  byte character = number % 10;
```

Die beiden nächsten Zeilen unterdrücken führende Nullen:

```
  if ((number == 0) && (i > 0))
    character = 0xf; // 7221-Segmente löschen
  sendCommand(numberOfDigits-i, character);
  number = number / 10;
 }
}
```

13.9 Kommunikation zwischen zwei oder mehr Arduino-Boards

Problem

Sie wollen, dass zwei oder mehr Arduino-Boards zusammenarbeiten. Vielleicht wollen Sie die E/A-Kapazität erhöhen oder mehr Daten verarbeiten, als ein einzelnes Board bewältigen kann. Sie können I2C nutzen, um Daten zwischen den Boards zu übergeben und so die Last zu verteilen.

Lösung

Die beiden Sketches in diesem Rezept zeigen, wie man I2C als Kommunikationskanal zwischen zwei oder mehr Arduino-Boards verwenden kann. Die Verschaltung sehen Sie in Abbildung 13-15.

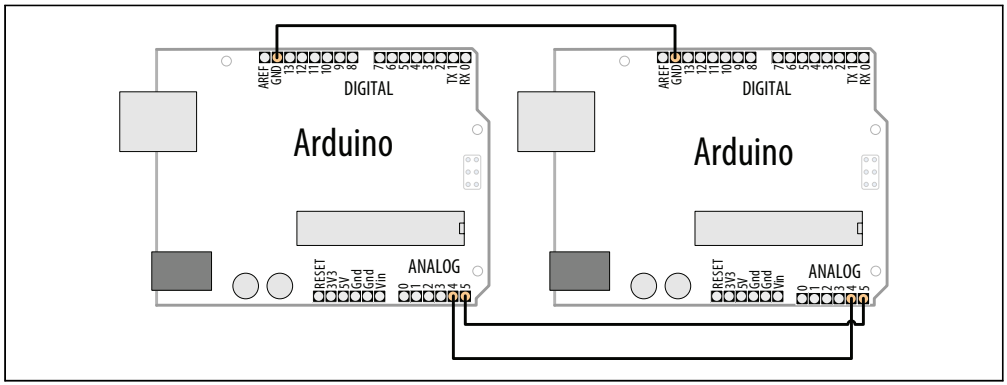


Abbildung 13-15: Arduino als I2C-Master und -Slave

Der Master sendet über den seriellen Port eingegangene Zeichen über I2C an einen Arduino-Slave:

```
/*
 * I2C_Master
 * Serielle Daten an einen I2C-Slave weitergeben
 */

#include <Wire.h>

const int address = 4; // Zu nutzende Adresse

void setup()
{
  Wire.begin();
}
```

```
void loop()
{
 char c;
 if(Serial.available() > 0 )
 {
   // Daten
   Wire.beginTransmission(address); // An Gerät senden
   Wire.write(c);
   Wire.endTransmission();
 }
}
```

Der Slave gibt die über I2C empfangenen Zeichen über seinen seriellen Port aus:

```
/*
 * I2C_Slave
 * Überwacht I2C-Requests und gibt sie über den seriellen Port aus
 */

#include <Wire.h>

const int address = 4;  // Zur Kommunikation verwendete Adresse

void setup()
{
 Serial.begin(9600);
 Wire.begin(address); // I2C-Bus über diese Adresse anbinden
 Wire.onReceive(receiveEvent); // Event-Handler für Requests registrieren
}

void loop()
{
 // Leer - die gesamte Verarbeitung erfolgt in receiveEvent
}

void receiveEvent(int howMany)
{
 while(Wire.available() > 0)
 {
  char c = Wire.read(); // Byte als Zeichen empfangen
  Serial.write(c); // und ausgeben
 }
}
```

Diskussion

Dieses Kapitel konzentriert sich auf den Arduino als I2C-Master, der auf verschiedene I2C-Slaves zugreift. Hier fungiert ein zweiter Arduino als I2C-Slave, der auf Requests von einem anderen Arduino reagiert. In Kapitel 4 behandelte Techniken zum Senden von Datenbytes können auch hier angewandt werden. Arduino 1.0 hat eine print-Funktion in die Wire-Biblothek integriert, d.h., Sie können Daten nun auch mit der print-Methode senden.

Der folgende Sketch sendet seine Ausgabe über I2C mit `Wire.println`. Mit dem oben vorgestellten I2C-Slave-Sketch kann der Master Daten ausgeben, ohne den eigenen seriellen Port nutzen zu müssen (der serielle Port des Slaves wird zur Ausgabe genutzt):

```
/*
 * I2C_Master
 * Sendet über print Sensordaten an einen I2C-Slave
 */

#include <Wire.h>

const int address = 4;   // Zur Kommunikation verwendete Adresse
const int sensorPin = 0; // Analogpin für Sensor
int val;          // Variable für Sensorwert

void setup()
{
  Wire.begin();
}

void loop()
{
  val = analogRead(sensorPin);    // Spannung am Poti einlesen
                                  // (Wert zwischen 0 und 1023)
  Wire.beginTransmission(address); // An Slave senden
  Wire.println(val);
  Wire.endTransmission();
  delay(1000);
}
```

Siehe auch

Kapitel 4 enthält weitere Informationen zur Verwendung der Arduino print-Funktion.

KAPITEL 14

Drahtlose Kommunikation

14.0 Einführung

Die Fähigkeit des Arduino zur Interaktion mit der Umgebung ist wundervoll, doch manchmal möchte man mit dem Arduino aus der Ferne kommunizieren, ohne Drähte und den Aufwand einer vollständigen TCP/IP-Verbindung. Dieses Kapitel behandelt verschiedene einfache Drahtlos-Module für Anwendungen, bei denen geringe Kosten die Hauptforderung sind. Die meisten Rezepte konzentrieren sich auf die vielseitigen XBee-Module.

XBee stellt eine flexible Drahtlos-Lösung für den Arduino dar, doch diese Flexibilität kann auch verwirrend ein. Dieses Kapitel enthält Beispiele, die vom einfachen »Drahtlos-Ersatz für den seriellen Port« bis hin zu Mesh-Netzwerken reichen, die mehrere Boards mit mehreren Sensoren verbinden.

Eine Reihe verschiedener XBee-Module stehen zur Verfügung. Die beliebtesten sind das XBee 802.15.4 (auch als XBee Serie 1 bekannt) und das XBee ZB Serie 2. Die Serie 1 ist einfacher zu nutzen als die Serie 2, unterstützt aber keine Mesh-Netzwerke. Siehe *http://www.digi.com/support/kbase/kbaseresultdetl.jsp?id=2213*.

14.1 Nachrichten über Low-Cost-Drahtlos-Module senden

Problem

Sie wollen Daten zwischen zwei Arduino-Boards über einfache, kostengünstige Drahtlos-Module senden.

Lösung

Dieses Rezept verwendet einfache Sende- und Empfangsmodule wie die SparkFun 315 MHz: WRL-10535 und WRL-10533, oder 434 MHz: WRL-10534 und WRL-10532.

Schließen Sie den Sender wie in Abbildung 14-1 und den Empfänger wie in Abbildung 14-2 an. Bei einigen Modulen heißt der Spannungsanschluss VDD statt Vcc.

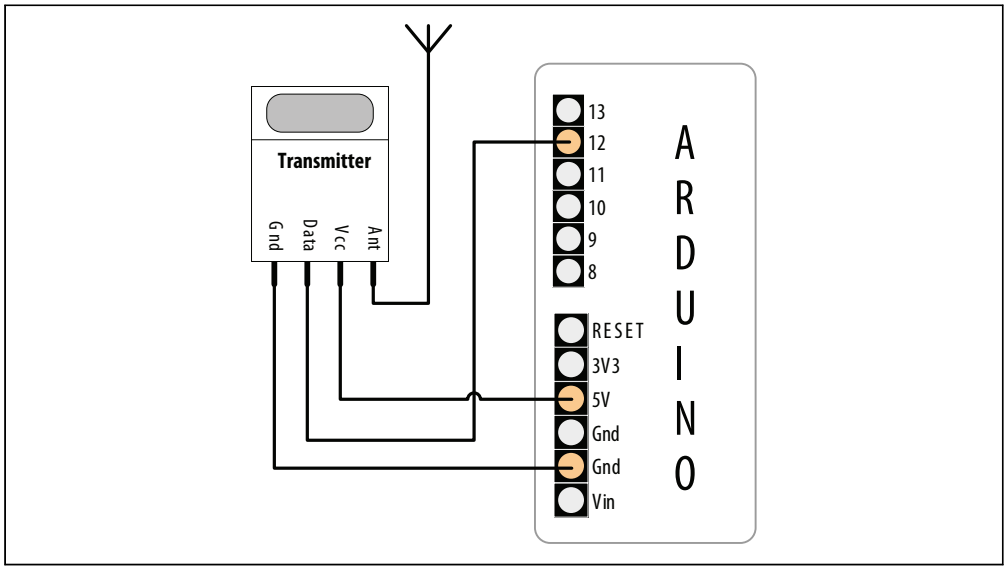


Abbildung 14-1: Einfacher Drahtlos-Sender mit VirtualWire

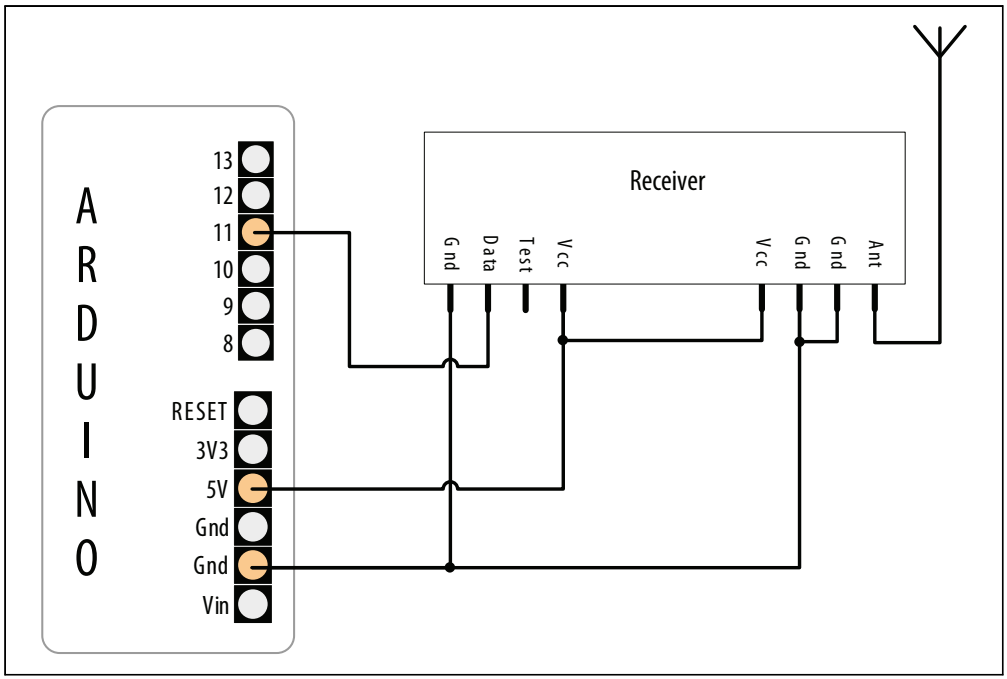


Abbildung 14-2: Einfacher Drahtlos-Empfänger mit VirtualWire

Der Sender-Sketch überträgt eine einfache Textnachricht an den Empfänger-Sketch, der diesen Text über den seriellen Monitor ausgibt. Sender- und Empfänger-Sketch verwenden die VirtualWire-Bibliothek von Mike McCauley, die eine Schnittstelle zur Drahtlos-Hardware bereitstellt. Die Bibliothek kann von *http://www.open.com.au/mikem/arduino/VirtualWire-1.5.zip* heruntergeladen werden:

```
/*
 SimpleSend
 Dieser Sketch sendet eine kurze Textnachricht mit Hilfe der VirtualWire-Bibliothek
 Verbinden Sie den Transmitter-Datenpin mit Arduino-Pin 12
*/

#include <VirtualWire.h>

void setup()
{
  // IO und ISR initialisieren
  vw_setup(2000);       // Bits pro Sekunde
}

void loop()
{
  send("Hallo");
  delay(1000);
}

void send (char *message)
{
 vw_send((uint8_t *)message, strlen(message));
 vw_wait_tx(); // Warten, bis gesamte Nachricht gesendet wurde
}
```

Der Empfänger-Sketch verwendet ebenfalls die VirtualWire-Bibliothek:

```
/*
 SimpleReceive
 Dieser Sketch gibt Textstrings aus, die über VirtualWire empfangen wurden
 Verbinden Sie den Empfänger-Datenpin mit dem Arduino-Pin 11
*/
#include <VirtualWire.h>

byte message[VW_MAX_MESSAGE_LEN];   // Puffer für eingehende Nachrichten
byte msgLength = VW_MAX_MESSAGE_LEN; // Länge der Nachricht


void setup()
{
  Serial.begin(9600);
  Serial.println("Bereit");

  // IO und ISR initialisieren
  vw_setup(2000);        // Bits pro Sekunde
  vw_rx_start();         // Empfänger starten
}
```

```
void loop()
{
  if (vw_get_message(message, &msgLength)) // Non-blocking
  {
    Serial.print("Empfangen: ");
  for (int i = 0; i < msgLength; i++)
  {
    Serial.write(message[i]);
  }
  Serial.println();
  }
}
```

Diskussion

Die VirtualWire-Bibliothek verwendet standardmäßig Pin 12 zum Senden und Pin 11 zum Empfangen. Wenn Sie andere Pins verwenden wollen, schauen Sie in die Dokumentation (siehe Link am Ende des Rezepts). `Setup` initialisiert die Bibliothek. Der `loop`-Code ruft einfach eine `send`-Funktion auf, die wiederum die Bibliotheksfunktion `vw_send` aufruft und dann wartet, bis die Nachricht übertragen wurde.

Der Empfänger initialisiert die Empfangslogik der Bibliothek und wartet dann in `loop` auf Nachrichten. `vw_get_message` gibt `true` zurück, wenn eine Nachricht vorhanden ist. Ist das der Fall, wird jedes Zeichen der Nachricht über den seriellen Monitor ausgegeben.

Die VirtualWire-Bibliothek packt mehrere Bytes zu Paketen zusammen, d.h., zum Senden binärer Daten müssen Sie nur die Adresse der Daten und die Zahl zu sendender Bytes übergeben.

Der folgende Sender-Sketch ähnelt dem obigen Sender-Sketch, füllt den Nachrichten-Puffer aber mit Binärwerten von Analogeingängen, die über `analogRead` eingelesen wurden. Die Größe des Puffers entspricht der Anzahl der zu sendenden Integerwerte mal der Anzahl der Bytes in einem Integerwert (die sechs Analogwerte benötigen 12 Bytes, da jedes `int` aus zwei Bytes besteht):

```
/*
 SendBinary
 Sendet digitale und analoge Pin-Werte als Binärdaten per VirtualWire
 Siehe SendBinary in Kapitel 4
*/

#include <VirtualWire.h>

const int numberOfAnalogPins = 6; // Anzahl einzulesender Analogpins

int data[numberOfAnalogPins]; // Der Datenpuffer

const int dataBytes = numberOfAnalogPins * sizeof(int); // Anzahl der Bytes
                                  // im Datenpuffer
```

```
void setup()
{
  // IO und ISR initialisieren
  vw_setup(2000);       // Bits pro Sekunde
}

void loop()
{
 int values = 0;
 for(int i=0; i <= numberOfAnalogPins; i++)
 {
  // Analogport einlesen
  data[i] = analogRead(i); // Wert im Datenpuffer speichern
 }
 send((byte*)data, dataBytes);
 delay(1000); // Einmal pro Sekunde senden
}


void send (byte *data, int nbrOfBytes)
{
 vw_send(data, nbrOfBytes);
 vw_wait_tx(); // Warten, bis gesamte Nachricht gesendet wurde
}
```

Mit dem Operator `sizeof` wird die Größe eines `int` in Byte bestimmt.

Die Empfänger-Seite wartet auf eingehende Nachrichten, stellt sicher, dass sie die richtige Länge haben und wandelt den Puffer wieder in sechs Integerwerte um, die über den seriellen Monitor ausgegeben werden:

```
#include <VirtualWire.h>

const int numberOfAnalogPins = 6; // Anzahl zu empfangender Integerwerte
int data[numberOfAnalogPins];  // Der Datenpuffer

// Anzahl der Bytes im Datenpuffer
const int dataBytes = numberOfAnalogPins * sizeof(int);

byte msgLength = dataBytes;


void setup()
{
 Serial.begin(9600);
 Serial.println("Bereit");
```

```
 // IO und ISR initialisieren
 vw_set_ptt_inverted(true); // Für DR3100 erforderlich
 vw_setup(2000);       // Bits pro Sekunde

 vw_rx_start();        // Empfänger starten
}

void loop()
{
 if (vw_get_message((byte*)data, &msgLength)) // Non-blocking
 {
  Serial.println("Empfangen: ");
  if(msgLength == dataBytes)
  {
   for (int i = 0; i < numberOfAnalogPins; i++)
   {
    Serial.print("Pin ");
    Serial.print(i);
    Serial.print("=");
    Serial.println(data[i]);
   }
  }
  else
  {
   Serial.print("Falsche Nachrichtlänge: ");
   Serial.println(msgLength);
  }
  Serial.println();
 }
}
```

Der serielle Monitor zeigt die Analogwerte des sendenden Arduinos an:

```
Empfangen:
Pin 0=1023
Pin 1=100
Pin 2=227
Pin 3=303
Pin 4=331
Pin 5=358
```

Beachten Sie, dass die maximale Puffergröße für VirtualWire bei 30 Bytes liegt (die Konstante `VW_MAX_MESSAGE_LEN` ist in der Header-Datei der Bibliothek definiert).

Die Reichweite liegt, je nach Versorgungsspannung und Antenne, bei etwa 100 Metern. Hindernisse zwischen Sender und Empfänger reduzieren die Reichweite.

Es gibt keine Garantie dafür, dass die Nachrichten zugestellt werden. Wenn Sie außerhalb der Reichweite sind oder wenn es starke Interferenzen gibt, können Nachrichten verloren gehen. Wenn Sie einen Mechanismus benötigen, der die Zustellung garantiert, ist die ZigBee-API (die in den noch folgenden Rezepten genutzt wird) die bessere Wahl. Doch die günstigen Module funktionieren gut, wenn es um Aufgaben wie die Ausgabe von Sensorwerten geht – jede Nachricht enthält den aktuellen Sensorwert und falls Nachrichten verloren gehen, werden sie durch nachfolgende Nachrichten ersetzt.

Siehe auch

Ein technisches Dokument zur VirtualWire-Bibliothek kann von *http://www.open.com.au/mikem/arduino/VirtualWire.pdf* heruntergeladen werden.

Datenblätter zu den Sende- und Empfangsmodulen finden Sie unter *http://www.sparkfun.com/datasheets/Wireless/General/MO-SAWR.pdf* und *http://www.sparkfun.com/datasheets/Wireless/General/MO-RX3400.pdf*.

14.2 Den Arduino mit einem ZigBee- oder 802.15.4-Netzwerk verbinden

Problem

Sie wollen Ihren Arduino mit einem ZigBee- oder 802.15.4-Netzwerk verbinden.

802.15.4 ist ein IEEE-Funknetz-Standard, der in Produkten wie den preiswerten XBee-Modulen von Digi International eingesetzt wird. *ZigBee* ist eine Unternehmens-Allianz und auch der Name eines Standards, der von dieser Allianz gepflegt wird. ZigBee ist eine Obermenge von IEEE 802.15.4 und in vielen Produkten implementiert, einschließlich einiger XBee-Module von Digi.

Nur XBee-Module, die als ZigBee-kompatiblel gelistet sind, etwa die XBee ZB-Module, sind garantiert ZigBee-konform. Davon abgesehen können Sie einen Teil der Features (IEEE 802.15.4) von ZigBee auch mit Modulen der älteren XBee Serie 1 verwenden. Tatsächlich funktionieren alle hier vorgestellten Rezepte mit Serie-1-Modulen.

Fehlersuche beim XBee

Wenn Sie Probleme haben, Ihre XBees zum Sprechen zu bringen, überprüfen Sie, ob sie den gleichen Firmware-Typ verwenden (z.B. XB24-ZB unter dem Modem: XBee-Einstellung wie in Abbildung 14-5) und ob die aktuellste Firmware-Version genutzt wird (der Versions-Wert in Abbildung 14-5). Umfassende Tipps zur XBee-Fehlersuche finden Sie in Robert Faludis »Common XBee Mistakes« auf *http://www.faludi.com/projects/common-xbee-mistakes/*. Umfassende Informationen zum Umgang mit XBees finden Sie im Buch *Building Wireless Sensor Networks* von O'Reilly (suchen Sie danach auf www.oreilly.de).

Lösung

Besorgen Sie sich zwei oder mehr XBee-Module, konfigurieren Sie sie (wie in Rezept 14.3 beschrieben) für die Kommunikation und verbinden Sie sie mit (mindestens) einem Arduino. Sie können weitere XBee-Module mit anderen Arduinos, einem Computer oder einem Analog-Sensor verbinden (siehe Rezept 14.4).

Wenn Sie den Arduino mit dem XBee verbinden und den nachfolgenden Sketch ausführen, gibt der Arduino jede Nachricht aus, die er vom XBee empfängt:

```
/*
 XBeeEcho
 Alles ausgeben, was über die seriellen Port eingeht
 */

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

void loop()
{
 while (Serial.available() ) {
  Serial.write(Serial.read()); // Empfangene Daten ausgeben
 }
}
```

Abbildung 14-3 zeigt den Anschluss eines Adafruit XBee-Adapters an den Arduino. Beachten Sie, dass die RX-Leitung des Arduino mit dem TX-Anschluss des XBee verbunden ist und umgekehrt.

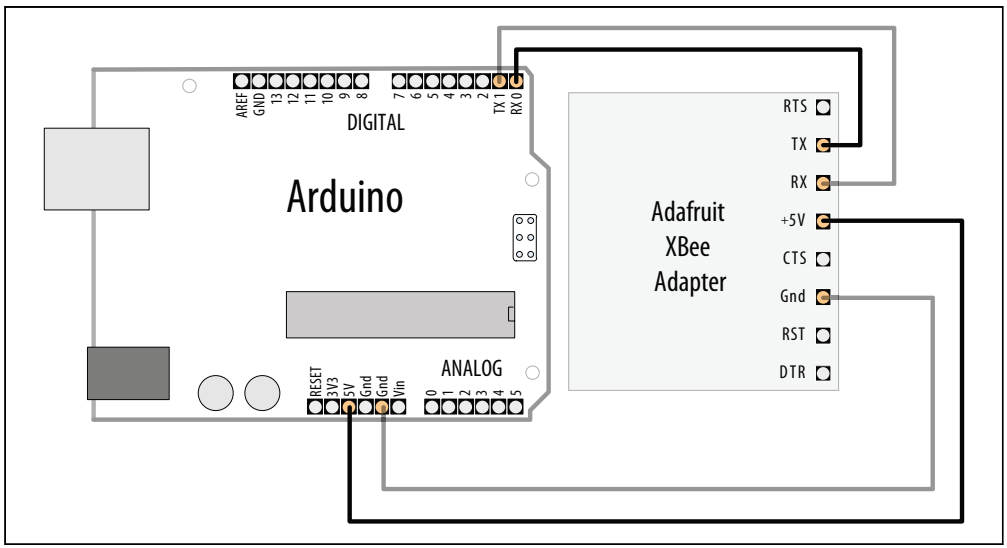
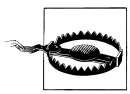
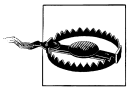


Abbildung 14-3: Anschluss eines Arduino an einen XBee per Adafruit XBee-Adapter



Wenn Sie einen anderen Adapter ohne eigenen Spannungsregler verwenden, wird die Spannung direkt an den XBee weitergegeben. In diesem Fall müssen Sie den 3V3-Pin des Arduino mit der Spannungsversorgung des Adapters verbinden. Anderenfalls riskieren Sie ein Durchbrennen Ihres XBee.

Sind die XBees mit einem Computer und/oder einem Arduino verbunden und konfiguriert, können Sie Nachrichten hin und her senden.



Sie müssen den Arduino vom XBee trennen, bevor Sie ihn programmieren. Das liegt daran, dass der Arduino die Pins 0 und 1 für die Programmierung nutzt. Die Signale kommen sich dann mit allem in die Quere (z.B. dem XBee), was an diese Pins angeschlossen ist.

Diskussion

Zur Konfiguration Ihrer XBees stecken Sie sie in einen XBee-Adapter wie dem Adafruit XBee-Adapter-Kit und einen USB-nach-TTL-Seriell-Adapter wie den TTL-232R, um den Adapter mit einem Computer zu verbinden.

Sie sollten mindestens zwei Adapter (und bei Bedarf zwei Kabel) kaufen, um zwei XBees gleichzeitig an den Computer anschließen zu können. Die gleichen Adapter können auch genutzt werden, um ein XBee mit dem Arduino zu verbinden.

Sie können auch einen All-In-One- XBee-USB-Adapter wie den Parallax XBee USB-Adapter oder den SparkFun XBee Explorer USB verwenden.

Abbildung 14-4 zeigt den Adafruit XBee-Adapter und den SparkFun XBee Explorer USB mit aufgesteckten Series 2 XBee-Modulen.

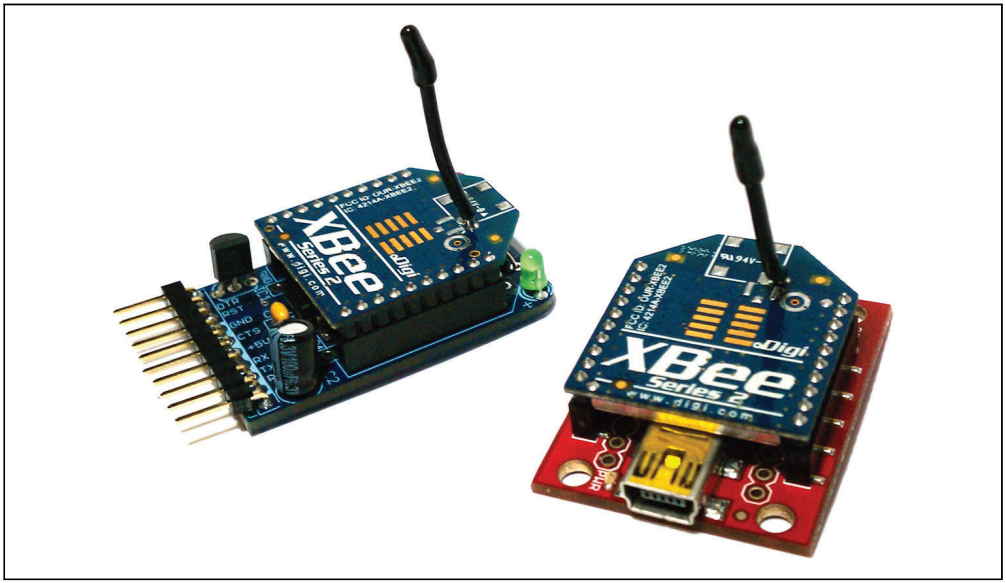


Abbildung 14-4: Zwei XBees, einer am Adafruit-, und der andere am SparkFun-Adapter

Series 2-Konfiguration

Zur Erstkonfiguration von Series 2 XBees müssen Sie Ihre XBees mit einem Windows-Computer verbinden (das Konfigurations-Utility ist für Mac und Linux nicht verfügbar). Schließen Sie erst einmal nur einen an einem USB-Port an. Der TTL-232R und der Parallax XBee USB-Adapter verwenden beide den gleichen USB-nach-Seriell-Treiber wie der Arduino auch. Sie müssen also keine zusätzlichen Treiber installieren.

1. Öffnen Sie den Gerätemanager (drücken Sie Windows-R, geben Sie **`devmgmt.msc`** ein und drücken Sie Enter), öffnen Sie den Ports-Bereich (COM & LPT) und halten Sie die Nummer des USB-Ports fest, mit dem der XBee verbunden ist. Wenn das nicht ganz klar ist, ziehen Sie den Stecker und stecken ihn wieder ein.) Beenden Sie den Gerätemanager.
2. Starten Sie die X-CTU-Anwendung (*http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=0&tp=5&tp2=0*), wählen Sie den oben ermittelten Port aus und drücken Sie Test/Query, um sicherzustellen, dass X-CTU den XBee erkennt. (Wenn nicht, sehen Sie sich das Support-Dokument unter *http://www.digi.com/support/kbase/kbaseresultdetl.jsp?id=2103* an.)
3. Wechseln Sie zum Reiter Modem Configuration und klicken Sie auf Read. X-CTU ermittelt das verwendete XBee-Modell sowie dessen aktuelle Konfiguration.
4. Unter Function Set wählen Sie ZIGBEE COORDINATOR AT (*nicht* API).
5. Klicken Sie das Versions-Menü an und wählen Sie die höchste verfügbare Firmware-Version aus.
6. Klicken Sie auf Show Defaults.
7. Ändern Sie die PAN ID-Einstellung von 0 auf 1234 (oder jeden anderen Hexadezimalwert; die PAN ID muss aber für alle Geräte im gleichen Netzwerk identisch sein), wie in Abbildung 14-5 zu sehen.
8. Klicken Sie auf Write.
9. Klicken Sie den Terminal-Reiter an.
10. Lassen Sie nun X-CTU laufen und den XBee eingesteckt. Schließen Sie den zweiten XBee an einem anderen USB-Port an. Wiederholen Sie die obigen Schritte (im zweiten Schritt starten Sie eine zweite Instanz von X-CTU), wählen im vierten Schritt aber statt ZIGBEE COORDINATOR AT die Option ZIGBEE ROUTER AT. Bei diesem XBee müssen Sie Channel Verification (JV) auch auf 1 setzen. Damit bestätigt er die Verwendung des richtigen Kanals, was die Verbindung mit dem Koordinator zuverlässiger macht.

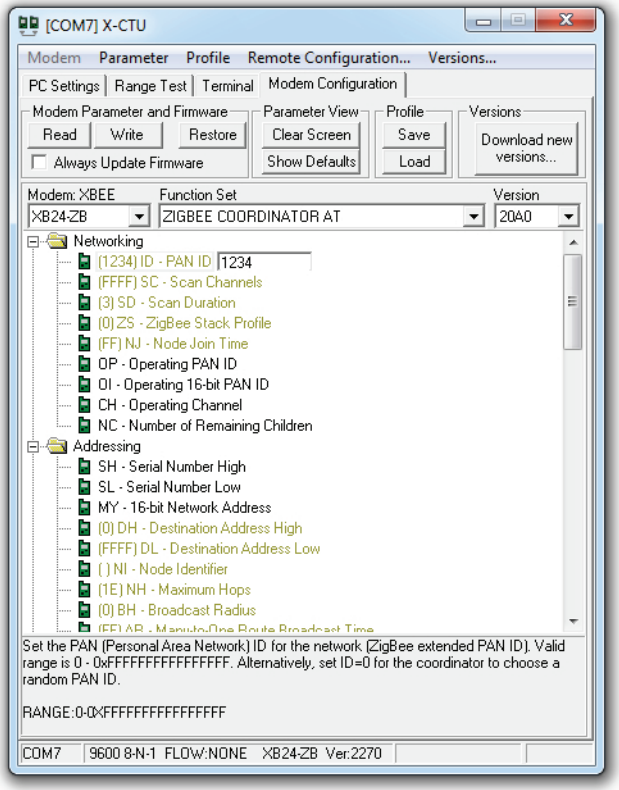


Abbildung 14-5: Konfiguration des XBee

Wenn Sie zwei Windows-Rechner haben, können Sie jeden XBee an einen separaten Rechner anschließen.

Beide XBees sind angeschlossen und in beiden X-CTU-Instanzen ist der Terminal-Reiter aktiv. Geben Sie nun etwas in eines der Terminal-Fenster ein. Was Sie in einem XBee eingeben, erscheint auf dem Terminal des anderen. Sie haben Ihr erstes einfaches XBee Personal Area Network (PAN) eingerichtet. Nun können Sie die XBees mit zwei Arduino-Boards verbinden und den Sketch aus Rezept 14.3 ausführen.

Konfiguration der Serie 1

Für Serie-1-XBees können Sie einen Mac oder einen PC mit Linux oder Windows nutzen. Wenn Sie die Firmware der XBees aktualisieren wollen, müssen Sie allerdings das X-CTU-Utility aus Rezept 14.3 verwenden.

Ermitteln Sie den seriellen Port Ihres XBee, wie es in »Den seriellen Port ermitteln« auf Seite 468 beschreiben wird. Verbinden Sie diesen Port mit dem Terminal-Programm. Die Verbindung zum XBee stellen Sie mit CoolTerm (Windows oder Mac) wie folgt her:

1. Starten Sie CoolTerm.

Sie können CoolTerm für Windows und Mac von *http://freeware.the-meiers.org/* herunterladen. PuTTY ist für Windows und Linux von *http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html* verfügbar. PuTTY für Linux können Sie möglicherweise auch über den Paketmanager Ihres Linux-Systems installieren. Bei Ubuntu ist PuTTY beispielsweise über das Universe-Repository mit `apt-get install putty` verfügbar.

2. Klicken Sie den Options-Button in der Symbolleiste an.
3. Wählen Sie den seriellen USB-Port aus (z.B. usbserial-A700eYw1 bei einem Mac oder COM8 bei eine PC). Stellen Sie sicher, das 9600 Baud, 8 Datenbits, keine Parität und 1 Stop-Bit eingestellt sind (das entspricht der Voreinstellung).
4. Aktivieren Sie Local Echo.
5. Klicken Sie auf OK.
6. Klicken Sie den Save-Button in der Symbolleiste an und speichern Sie die Session-Einstellungen.
7. Bei weiteren Sessions überspringen Sie die Schritte 2 bis 6, klicken stattdessen auf Open und laden die gesicherten Einstellungen.
8. Klicken Sie den Connect-Button in der Symbolleiste an.

Den seriellen Port ermitteln

Um den seriellen Port zu ermitteln, der Ihrem XBee unter Windows zugewiesen wurde, sehen Sie sich den ersten Schritt in Rezept 14.3 an. Um ihn unter Mac OS X zu ermitteln, öffnen Sie ein Mac OS X-Terminal-Fenster (in */Applications/Utilities*) und geben den folgenden Befehl ein: **`ls /dev/tty.usbserial-*`**. Unter Linux öffnen Sie ein xterm oder ein vergleichbares Terminal-Fenster und geben **`ls /dev/ttyUSB*`** ein.

Erscheint hier mehr als ein Eintrag, klemmen Sie alle seriellen USB-Geräte bis auf den XBee ab und geben den Befehl erneut ein. Sie sollten jetzt nur noch einen Eintrag sehen.

Das Ergebnis sieht bei einem Mac etwa so aus:

```
/dev/tty.usbserial-A700eYw1
```

Und bei Linux etwa so:

```
/dev/ttyUSB0
```

Das ist der Dateiname für den seriellen USB-Port Ihres XBee.

Um die Verbindung zu Ihrem XBee über PuTTY (Windows oder Linux) herzustellen, machen Sie Folgendes:

1. Starten Sie PuTTY.
2. Klicken Sie unter Connection Type auf Serial.
3. Tragen Sie den Namen des seriellen Ports im Feld Serial Line ein (z.B. /dev/ttyUSB0 unter Linux oder COM7 unter Windows). Stellen Sie sicher, dass die Geschwindigkeit (Speed) auf 9600 gesetzt ist (das ist die Voreinstellung).
4. Auf der linken Seite des Fenster klicken Sie unter Category auf Terminal.
5. Unter Local Echo wählen Sie Force On.
6. Unter »Set various terminal options« wählen Sie Implicit LF in Every CR.
7. Auf der linken Seite des Fensters klicken Sie unter Category auf Session.
8. Geben Sie der Session einen Namen, z.B. »XBee 1«, und klicken Sie auf Save.
9. Bei zukünftigen Sessions überspringen Sie die Schritte 2 bis 8 und klicken die gespeicherte Session doppelt an.

Nachdem Sie die Verbindung hergestellt haben, konfigurieren Sie den ersten XBee mit den folgenden AT-Befehlen. Sie müssen **+++** eingeben (kein `Return` oder `Enter`) und eine Sekunde warten, um den XBee auf sich aufmerksam zu machen (er antwortet mit »OK«):

```
ATMY1234
ATDL5678
ATDH0
ATID0
ATWR
```

Lassen Sie das Terminal-Fenster geöffnet, um weitere Befehle eingeben zu können. Stecken Sie nun den zweiten XBee ein und stellen Sie wie oben beschrieben wieder eine Verbindung mit PuTTY oder CoolTerm her (um ein neues PuTTY-Fenster zu öffnen, starten Sie das Programm einfach noch einmal; ein neues CoolTerm-Fenster öffnen Sie mit File→New). Dann konfigurieren Sie den zweiten XBee mit den folgenden Befehlen:

```
ATMY5678
ATDL1234
ATDH0
ATID0
ATWR
```

Nun können Sie Befehle im Terminal-Fenster eines XBee eingeben und die Eingaben erscheinen im Terminal-Fenster des anderen XBee (und umgekehrt).

Der Befehl `ATMY` legt den Identifier für einen XBee fest, `ATDL` und `ATDH` das untere und obere Byte für den Ziel-XBee. `ATID` legt die Netzwerk-ID fest (die für alle XBees, die miteinander kommunizieren sollen, gleich sein muss) und `ATWR` speichert die Einstellungen, damit der XBee sie nicht vergisst, wenn er aus- und wieder eingeschaltet wird.

Mit dem Arduino kommunizieren

Nachdem die XBee-Module konfiguriert sind, wählen Sie einen der XBees aus, schließen dessen Terminal-Fenster und trennen ihn vom Computer. Dann laden Sie den nachfolgenden Code auf den Arduino hoch und verbinden den XBee wie in Abbildung 14-3 zu sehen mit dem Arduino. Wenn Sie nun Zeichen über ein Terminal-Programm eingeben, wird es zweimal ausgegeben (Echo), d.h., wenn Sie a eingeben, sehen Sie aa).

Wenn jedes Zeichen doppelt erscheint, liegt das am lokalen Echo, das Sie im Terminal-Programm aktiviert haben. Wenn Sie wollen, können Sie die Verbindung trennen und mit deaktiviertem lokalen Echo wieder herstellen (folgen Sie dazu den obigen Anweisungen für CoolTerm oder PuTTY und schalten Sie das lokale Echo aus).

Siehe auch

14.3, 14.4 und 14.5

14.3 Eine Nachricht an einen bestimmten XBee senden

Problem

Sie wollen festlegen, welcher Knoten die Nachrichten von Ihrem Arduino-Sketch erhält.

Lösung

Senden Sie die AT-Befehle direkt aus dem Arduino-Sketch:

```
/*
 XBeeMessage
 Sendet eine Nachricht über die Adresse an einen XBee
 */

boolean configured;

boolean configureRadio() {

 // Befehlsmodus aktivieren:
 Serial.print("+++");

 String ok_response = "OK\r"; // Die von uns erwartete Antwort

 // Text der Antwort in response-Variable einlesen
 String response = String("");
 while (response.length() < ok_response.length()) {
  if (Serial.available() > 0) {
   response += (char) Serial.read();
  }
 }
```

```
 // Bei der richtigen Antwort konfigurieren wir den XBee und geben 'wahr' zurück.
 if (response.equals(ok_response)) {
  Serial.print("ATDH0013A200\r"); // Höherwertiges Bytes des Ziels-ERSETZEN
  Serial.print("ATDL403B9E1E\r"); // Niederwertiges Byte des Ziels-ERSETZEN
  Serial.print("ATCN\r");   // Zurück zum Datenmodus
  return true;
 } else {
  return false; // Die Antwort war falsch
 }
}

void setup () {
 Serial.begin(9600); // Seriellen Port starten
 configured = configureRadio();
}

void loop () {
 if (configured) {
  Serial.print("Hallo!");
  delay(3000);
 }
 else {
  delay(30000);   // 30 Sekunden warten
  configured = configureRadio(); // Erneut versuchen
 }
}
```

Diskussion

Zwar funktioniert die Konfiguration aus Rezept 14.2 für zwei XBees, ist aber nicht besonders flexibel, wenn mehr als zwei genutzt werden.

Nehmen wir zum Beispiel ein Netzwerk aus drei Serie-2-XBee-Knoten, bei dem ein XBee mit der COORDINATOR AT-Firmware ausgestattet ist und zwei andere mit der ROUTER AT-Firmware. Vom Koordinator gesendete Nachrichten gehen an die beiden Router. Nachrichten von den jeweiligen Routern landen beim Koordinator.

Die Serie-1-Konfiguration in diesem Rezept ist etwas flexibler, da es die Ziele explizit angibt. Da die Geräte über AT-Befehle konfiguriert und die Konfiguration dann gespeichert wird, kodieren Sie die Zieladressen fest in die Firmware ein.

Diese Lösung erlaubt es hingegen, AT-Befehle zur Konfiguration der XBees zu senden. Das Herz der Lösung bildet die Funktion `configureRadio()`. Sie sendet die Escape-Sequenz `+++`, um den XBee in den Befehlsmodus zu schalten (so wie bei der Serie-1-Konfiguration am Ende von Rezept 14.2). Nach dem Senden dieser Escape-Sequenz wartet der Arduino-Sketch auf die OK-Antwort, bevor er die folgenden AT-Befehle sendet:

```
ATDH0013A200
ATDL403B9E1E
ATCN
```

In Ihrem Code müssen Sie *0013A200* und *403B9E1E* durch die oberen und unteren Adressbytes des Ziels ersetzen.

Die ersten beiden Befehle entsprechen denen der Serie-1-Konfiguration am Ende von Rezept 14.2, doch die Zahlen sind länger. Das liegt daran, dass in diesem Rezept Serie-2–Adressen verwendet werden. Wie in Rezept 14.2 erklärt, können Sie die Adresse eines Serie-1-XBee mit dem Befehl `ATMY` festlegen, doch bei Serie-2-XBees hat jedes Modul eine eindeutige Adresse, die in den Chip integriert ist. Sie können sich den höherwertigen (`ATDH`) und niederwertigen (`ATDL`) Teil der Seriennummer mit X-CTU ansehen (siehe Abbildung 14-6). Dieser Wert steht auch auf dem Label unter dem XBee.

Der Befehl `ATCN` beendet den Befehlmodus und ist sozusagen die Umkehrung der +++-Sequenz.

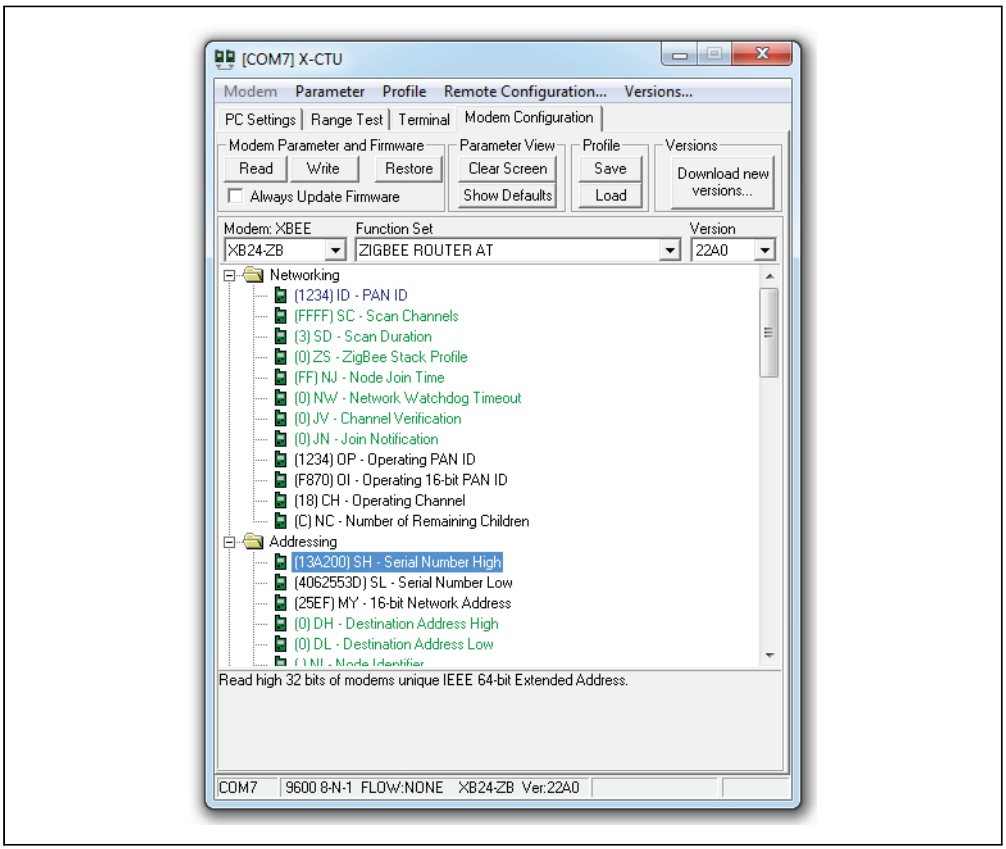


Abbildung 14-6: Höher- und niederwertige Bytes der Seriennummer mit X-CTU nachschauen

Siehe auch

Rezept 14.2

14.4 Sensordaten zwischen XBees senden

Problem

Sie wollen den Status von Digital-, Analog- oder Steuerpins basierend auf vom XBee empfangenen Befehlen senden.

Lösung

Verbinden Sie einen der XBees (den sendenden XBee) mit einem Analogsensor und konfigurieren Sie ihn so, dass er den Wert regelmäßig sendet. Verbinden Sie den Arduino mit einem XBee (dem empfangenden XBee), der für den API-Modus konfiguriert ist und die API-Frames einliest, die vom anderen XBee empfangen werden.

Diskussion

XBees verfügen über einen integrierten Analog/Digital-Wandler (analog-to-digital converter, ADC), der regelmäßig abgefragt werden kann. Der XBee kann so konfiguriert werden, dass er die Werte (zwischen 0 und 1023) an andere XBees im Netzwerk sendet. Die Konfiguration und der Code unterscheiden sich zwischen Serie-2- und Serie-1-XBees ein wenig.

Serie-2-XBees

Mit X-CTU (siehe Rezept 14.3 in Rezept 14.2), konfigurieren Sie den sendenden XBee als ZIGBEE ROUTER AT (*nicht* API) und mit den folgenden Einstellungen (klicken Sie Write, wenn Sie fertig sind):

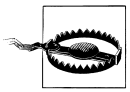
PAN ID: **1234** (oder ein anderer von Ihnen gewählter Wert, der aber für alle XBees gleich sein muss)
Channel Verification (JV): **1** (stellt sicher, dass der Router nach einer Stromunterbrechung bzw. Neustart zunächst auf dem vorhandenen Kanal nach dem (bekannten) Coordinator sucht und falls er keinen findet, den Kanal verlässt und versucht, einen neuen Coordinator in einem neuen PAN zu finden. Bei JV=0 würde er das letztere nicht tun, sondern auf dem Kanal verbleiben)
Destination Address High (DH): der höherwertige Teil der Adresse (SH) des anderen XBee, üblicherweise 13A200
Destination Address Low (DL): der niederwertige Teil der Adresse (SL) des anderen XBee
Unter I/O Settings, AD0/DIO0 Configuration (D0): **2**
Unter I/O Settings→Sampling Rate (IR): **64** (100 Millisekunden in hex)

Sie können den höher- (ATDH) und niederwertigen (ATDL) Teil der Seriennummer mit X-CTU ermitteln (siehe Abbildung 14-6). Die Werte stehen aber auch auf dem Label auf der Unterseite des XBee.

Konfigurieren Sie den empfangenden XBee als ZIGBEE COORDINATOR API (*nicht* AT) und den folgenden Einstellungen:

PAN ID: **1234** (oder ein anderer von Ihnen gewählter Wert, der aber für alle XBees gleich sein muss)
Destination Address High (DH): der höherwertige Teil der Adresse (SH) des anderen XBee, üblicherweise 13A200
Destination Address Low (DL): der niederwertige Teil der Adresse (SL) des anderen XBee

Verbinden Sie den sendenden XBee mit dem Sensor, wie in Abbildung 14-7 zu sehen. Der Wert von R1 muss doppelt so hoch sein wie der des Potis (bei einem 10K-Poti also ein 20K-Widerstand). Das liegt daran, dass der Analog/Digital-Wandler des Seri- 2-XBees im Bereich von 0 bis 1,2 Volt arbeitet. R1 sorgt dafür, dass die 3,3 Volt auf unter 1,2 Volt sinken.



Überprüfen Sie sorgfältig das Pinout Ihres XBee-Breakout-Boards, da die Pins des Breakout-Boards nicht immer mit den Pins des XBee übereinstimmen. Beispielsweise ist bei einigen Breakout-Boards der obere linke Pin Masse und der darunter 3,3V.

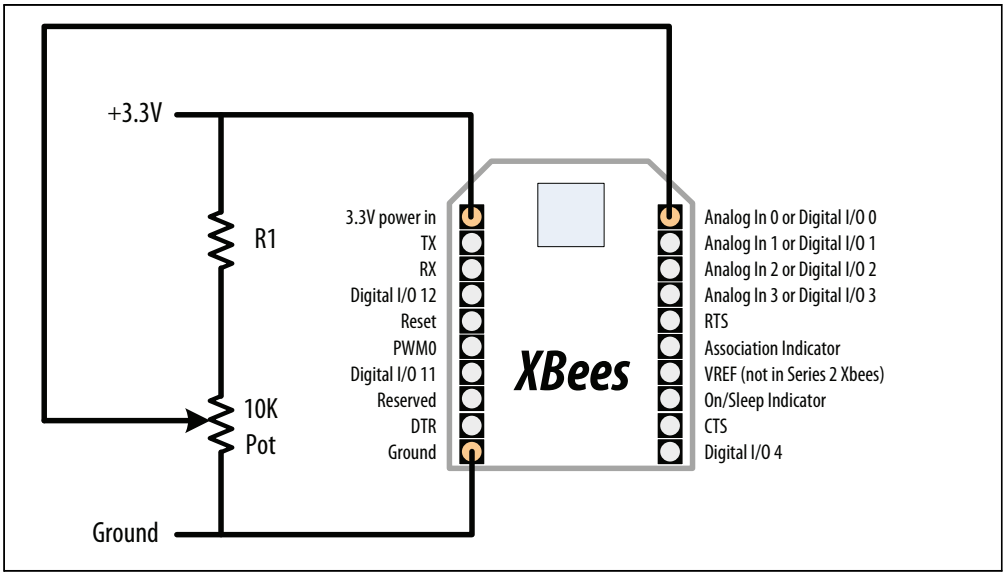


Abbildung 14-7: Anschluss des empfangenden Serie 2-XBee an einen Analogsensor

Nun laden Sie den folgenden Sketch auf den Arduino hoch und schließen den sendenden XBee wie in Rezept 14.2 an den Arduino an. Wenn Sie den Arduino neu programmieren, müssen Sie den XBee zuerst abklemmen:

```
/*
  XBeeAnalogReceive
```

```
 Analogwert von einem XBee API-Frame einlesen und die Helligkeit einer LED entsprechend setzen
*/

#define LEDPIN 9

void setup() {
 Serial.begin(9600);
 pinMode(LEDPIN, OUTPUT);
}

void loop() {

 if (Serial.available() >= 21) { // Warten, bis ein paar Daten vorliegen

   if (Serial.read() == 0x7E) { // Startzeichen eines Frames

    // Uninteressante Bytes im API-Frame überspringen
    for (int i = 0; i < 18; i++) {
     Serial.read();
    }

    // Die nächsten beiden Bytes sind der höher- und der niederwertige Teil des Sensorwerts
    int analogHigh = Serial.read();
    int analogLow = Serial.read();
    int analogValue = analogLow + (analogHigh * 256);

    // Helligkeit auf PWM-Bereich abbilden
    int brightness = map(analogValue, 0, 1023, 0, 255);

    // LED einschalten
    analogWrite(LEDPIN, brightness);
   }
 }
}
```

Serie-1-XBees

Mit Hilfe eines Terminal-Programms (siehe Rezept 14.3 in Rezept 14.2) senden Sie die folgenden Konfigurationsbefehle an den XBee:

```
ATRE
ATMY1234
ATDL5678
ATDH0
ATID0
ATD02
ATIR64
ATWR
```

Dann senden Sie die folgenden Konfigurationsbefehle an die XBees:

```
ATRE
ATMY5678
ATDL1234
ATDH0
ATID0
ATWR
```

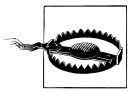
Beide XBees

`ATRE` setzt den XBee auf die Werkseinstellung zurück. Der `ATMY`-Befehl legt die ID des XBee fest. `ATDL` und `ATDH` setzen das nieder- und das höherwertige Byte des Ziel-XBee. `ATID` legt die Netzwerk-ID fest (und muss bei allen miteinander kommunizierenden XBees gleich sein). `ATWR` speichert die Einstellungen im XBee, damit sie auch erhalten bleiben, wenn er aus- und wieder eingeschaltet wird.

Sendender XBee

`ATD02` konfiguriert Pin 20 (Analog- oder Digitaleingang 0) als analogen Eingang; `ATIR64` weist den XBee an, den Sensor alle 100 Millisekunde (64 hex) abzufragen und an den durch `ATDL` und `ATDH` festgelegten XBee zu senden.

Schließen Sie den sendenden XBee wie in Abbildung 14-8 zu sehen an den Sensor an.



Achten Sie auf das Pinout Ihres XBee-Breakout-Boards, da dessen Pins nicht mit den Pins des XBees übereinstimmen müssen. Zum Beispiel liegt bei manchen Breakout-Boards Masse (GND) am oberen linken Pin, und darunter 3,3V. Auch könnte der VREF-Pin (RES beim SparkFun XBee Explorer USB) der fünfte Pin unten rechts sein, während er beim XBee selbst der vierte von unten ist.

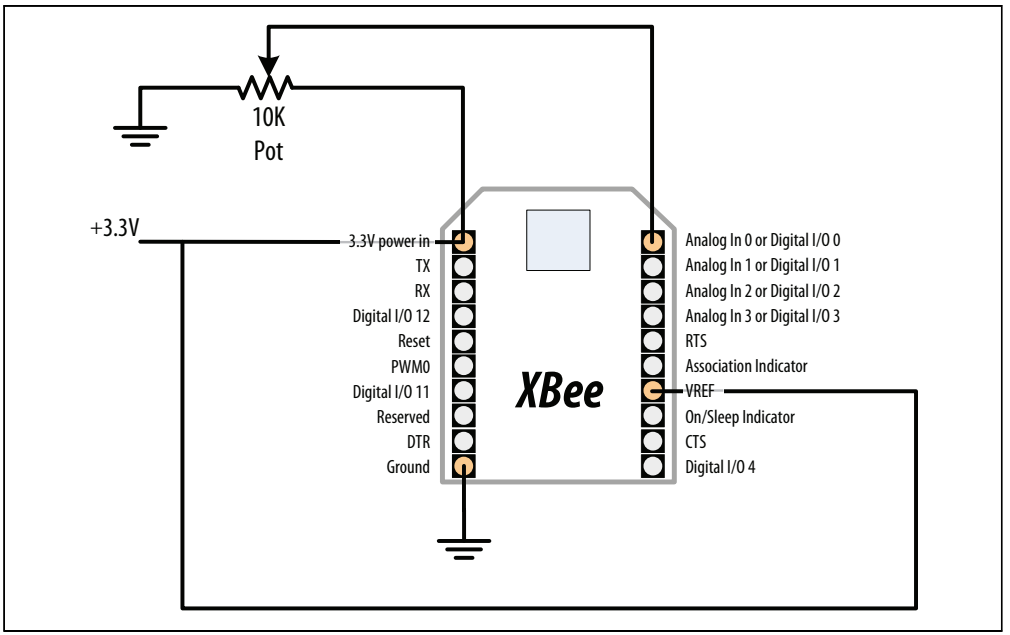


Abbildung 14-8: Anschluss des empfangenden Serie 1-XBee an einen Analogsensor

Im Gegensatz zur Serie 2 verwenden Serie1-XBees eine externe Referenz, die mit 3,3V verbunden ist. Da die Spannung am Schleifer des Potis nie höher sein kann als die Referenzspannung, wird der Widerstand aus Abbildung 14-7 nicht benötigt.

Nun laden Sie den folgenden Sketch auf den Arduino hoch und schließen den sendenden XBee wie in Rezept 14.2 beschrieben an den Arduino an. Muss der Arduino neu programmiert werden, klemmen Sie den XBee zuerst ab:

```
/*
 XBeeAnalogReceiveSeries1
 Analogwert von XBee API-Frame einlesen und Helligkeit einer LED entsprechend einstellen
*/

const int ledPin = 9;

void setup() {
 Serial.begin(9600);
 pinMode(ledPin, OUTPUT);
 configureRadio(); // Überprüfen Sie den Rückgabewert, wenn eine Fehlerbehandlung notwendig ist
}

boolean configureRadio() {

 // Befehlsmodus aktivieren
 Serial.flush();
 Serial.print("+++");
 delay(100);

 String ok_response = "OK\r"; // Die von uns erwartete Antwort

 // Text der Antwort in die response-Variable einlesen
 String response = String("");
 while (response.length() < ok_response.length()) {
  if (Serial.available() > 0) {
   response += (char) Serial.read();
  }
 }

 // Bei der richtigen Antwort XBee konfigurieren und 'wahr' zurückgeben
 if (response.equals(ok_response)) {
  Serial.print("ATAP1\r"); // API-Modus aktivieren
  delay(100);
  Serial.print("ATCN\r");  // Zurück zum Datenmodus
  return true;
 } else {
  return false; // Die Antwort war falsch
 }
}

void loop() {

 if (Serial.available() >= 14) { // Auf ein paar Daten warten

   if (Serial.read() == 0x7E) { // Startzeichen eines Frames

    // Uninteressante Bytes des API-Frames überspringen
    for (int i = 0; i < 10; i++) {
     Serial.read();
    }
```

```
    // Die nächsten beiden Bytes sind der höher- und der niederwertige Teil des Sensorwerts
    int analogHigh = Serial.read();
    int analogLow = Serial.read();
    int analogValue = analogLow + (analogHigh * 256);

    // Helligkeit auf PWM-Bereich abbilden
    int brightness = map(analogValue, 0, 1023, 0, 255);

    // LED einschalten
    analogWrite(ledPin, brightness);
   }
  }
}
```

Bei Serie-1-XBees muss der Arduino-Code den XBee mit einem AT-Befehl (`ATAP1`) für den API-Modus konfigurieren. Bei Serie-2-XBees geschieht das, indem man eine andere Firmware-Version in den Flash-Speicher schreibt. Der Grund für die Rückkehr in den Datemodus (`ATCN`) ist, dass der Befehlsmodus vorher mit `+++` aktiviert wurde und die Rückkehr in den Datenmodus notwendig ist,um Daten empfangen zu können.

Siehe auch

Rezept 14.2

14.5 Einen mit dem XBee verbundenen Aktuator aktivieren

Problem

Sie wollen einen XBee einen Pin aktivieren lassen, über den ein daran angeschlossener Aktuator (z.B. ein Relais oder eine LED) eingeschaltet werden kann.

Lösung

Konfigurieren Sie den mit dem Aktuator verbundenen XBee so, dass er Anweisungen von einem anderen XBee akzeptiert. Verbinden Sie den anderen XBee mit einem Arduino, der die Befehle sendet, die zur Aktivierung des digitalen E/A-Pins notwendig sind.

Diskussion

Die digitalen/analogen E/A-Pins des XBee können als digitale Ausgänge konfiguriert werden. Darüber hinaus können XBees so konfiguriert werden, dass Sie Anweisungen von anderen XBees akzeptieren, mit denen diese Pins ein- und ausgeschaltet werden können. Bei Serie-2-XBees nutzen Sie das »Remote AT Command«-Feature. Bei Serie-1-XBees können Sie die direkte Ein-/Ausgabe nutzen, was einen »virtuellen Draht« (virtual wire) zwischen den XBees erzeugt.

Serie-2-XBees

Mittels X-CTU (siehe Rezept 14.3) konfigurieren Sie den *empfangenden* XBee als ZIGBEE ROUTER AT (*nicht* API) und nehmen die folgenden Einstellungen vor:

PAN ID: **1234** (oder ein anderer von Ihnen gewählter Wert, der aber für alle XBees gleich sein muss)
Channel Verification (JV): **1** (stellt sicher, dass der Router nach einer Stromunterbrechung bzw. Neustart zunächst auf dem vorhandenen Kanal nach dem (bekannten) Coordinator sucht und falls er keinen findet, den Kanal verlässt und versucht, einen neuen Coordinator in einem neuen PAN zu finden. Bei JV=0 würde er das letztere nicht tun, sondern auf dem Kanal verbleiben)
Destination Address High (DH): der höherwertige Teil der Adresse (SH) des anderen XBee, üblicherweise 13A200
Destination Address Low (DL): der niederwertige Teil der Adresse (SL) des anderen XBee
Unter I/O Settings, AD1/DIO1 Configuration (D1): **4** (digitaler Ausgang, low)

Sie können den höherwertigen (ATDH) und niederwertige (ATDL) Teil der Seriennummer mit X-CTU ermitteln, wie in Abbildung 14-6 gezeigt. Die Zahlen stehen auch auf dem Label auf der Unterseite des XBee.

Konfigurieren Sie den *sendenden* XBee mit der ZIGBEE COORDINATOR API (*nicht* AT) und nehmen Sie die folgenden Einstellungen vor:

PAN ID: **1234** (stellt sicher, dass der Router nach einer Stromunterbrechung bzw. Neustart zunächst auf dem vorhandenen Kanal nach dem (bekannten) Coordinator sucht und falls er keinen findet, den Kanal verlässt und versucht, einen neuen Coordinator in einem neuen PAN zu finden. Bei JV=0 würde er das letztere nicht tun, sondern auf dem Kanal verbleiben)
Destination Address High (DH): der höherwertige Teil der Adresse (SH) des anderen XBee, üblicherweise 13A200
Destination Address Low (DL): der niederwertige Teil der Adresse (SL) des anderen XBee

Schließen Sie eine LED an den empfangenden XBee an, wie in Abbildung 14-9 zu sehen.

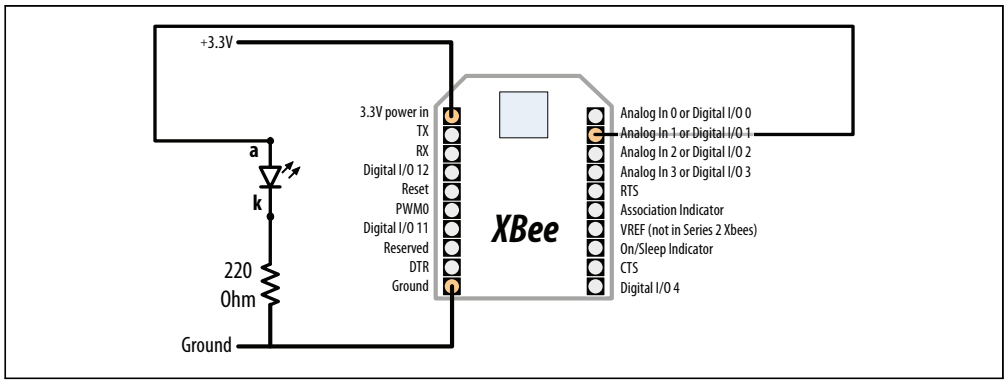


Abbildung 14-9: Anschluss einer LED an XBees digitalen E/A-Pin 1 (Serie 1 und Serie 2)

Nun laden Sie den folgenden Sketch auf den Arduino hoch und schließen den sendenden XBee wie in Rezept 14.2 beschrieben an den Arduino an. Wenn Sie den Arduino neu programmieren müssen, dürfen Sie nicht vergessen, den XBee zuerst abzuklemmen. Der Sketch sendet einen Remote AT-Befehl (ATD14 oder ATD15), der Pin 1 (ATD1) abwechselnd ein und ausschaltet:

```
/*
 XBeeActuate
 Sendet einen Remote AT-Befehl, um den Digitalpin eines anderen XBee zu aktivieren
 */

const byte frameStartByte = 0x7E;
const byte frameTypeRemoteAT  = 0x17;
const byte remoteATOptionApplyChanges = 0x02;

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

void loop()
{

 toggleRemotePin(1);
 delay(3000);
 toggleRemotePin(0);
 delay(2000);
}


byte sendByte(byte value) {
 Serial.write(value);
 return value;
}

void toggleRemotePin(int value) {  // 0 = aus, nicht-0 = an

 byte pin_state;
 if (value) {
  pin_state = 0x5;
 } else {
  pin_state = 0x4;
 }

 sendByte(frameStartByte); // Start des API-Frames

 // Höher- und niederwertiger Teil der Frame-Länge (ohne Prüfsumme)
 sendByte(0x0);
 sendByte(0x10);

 long sum = 0; // Prüfsumme akkumulieren

 sum += sendByte(frameTypeRemoteAT); // Dieser Frame enthält einen
                     // Remote AT-Befehl

 sum += sendByte(0x0);  // Frame-ID ist 0; wir erwarten keine Antwort
```

```
  // Die folgenden 8 Bytes geben die ID des Empfängers an
  // Broadcasting an alle Knoten mit 0xFFFF
  sum += sendByte(0x0);
  sum += sendByte(0x0);
  sum += sendByte(0x0);
  sum += sendByte(0x0);
  sum += sendByte(0x0);
  sum += sendByte(0x0);
  sum += sendByte(0xFF);
  sum += sendByte(0xFF);

  // Die beiden folgenden Bytes enthalten die 16-Bit-Adresse des Empfängers
  // Broadcasting an alle Knoten mit 0xFFFE
  sum += sendByte(0xFF);
  sum += sendByte(0xFF);

  sum += sendByte(remoteATOptionApplyChanges); // Remote AT-Optionen senden

  // Text des AT-Befehls senden
  sum += sendByte('D');
  sum += sendByte('1');

  // Der Wert (0x4 für aus, 0x5 für an)
  sum += sendByte(pin_state);

  // Prüfsumme senden
  sendByte( 0xFF - ( sum & 0xFF));

  delay(10); // Pause, damit der Mikrocontroller zur Ruhe kommt
 }
```

Serie 1-XBees

Mit Hilfe eines Terminal-Programms (siehe Rezept 14.3) senden Sie die folgenden Konfigurationsbefehle an den *sendenden* XBee (der mit dem Arduino verbunden ist):

```
ATRE
ATMY1234
ATDL5678
ATDH0
ATID0
ATD13
ATICFF
ATWR
```

Dann senden Sie die folgenden Konfigurationsbefehle an den *empfangenden* XBee:

```
ATRE
ATMY5678
ATDL1234
ATDH0
ATID0
ATD14
ATIU0
ATIA1234
ATWR
```

Beide XBees

ATRE setzt den XBee auf die Werkseinstellung zurück. Der ATMY-Befehl legt die XBee-ID fest. ATDL und ATDH legen das niederwertige und höherwertige Byte des Ziel-XBee fest. ATID legt die Netzwerk-ID fest (die für alle miteinander kommunizierenden XBees gleich sein muss). ATWR speichert die Einstellungen im XBee, damit sie auch erhalten bleiben, wenn die Spannung aus- und wieder eingeschaltet wird.

Sendender XBee

ATICFF weist den XBee an, alle Digitaleingänge abzufragen und deren Werte an den XBee zu senden, der mit ATDL und ATDH festgelegt wurde. ATD13 konfiguriert Pin 19 (Analog- oder Digitaleingang 1) als digitalen Eingang. Der Zustand dieses Pins wird vom sendenden zum empfangenden XBee weitergegeben.

Empfangender XBee

ATIU1 weist den XBee an, die empfangenen Frames nicht an den seriellen Port zu schicken. ATIA1234 weist ihn an, Befehle vom anderen XBee (mit der MY-Adresse 1234) zu akzeptieren. ATD14 setzt Pin 19 (Analog- oder Digitaleingang 1) auf 0 (aus).

Schließen Sie den sendenden XBee wie in Abbildung 14-10 zu sehen an den Arduino an.

Schließen Sie dann den empfangenden XBee wie in Rezept 14.2 beschrieben an den Arduino an. Beachten Sie, dass wir jetzt keine AT-Befehle über den seriellen Port senden, sondern eine elektrische Verbindung nutzen, um den XBee-Pin einzuschalten. Die beiden 10K-Widerstände bilden einen Spannungsteiler, der die 5V-Spannung des Arduino auf etwa 2,5V reduziert (hoch genug, damit der XBee sie erkennt, aber niedrig genug, um eine Beschädigung der XBee-3,3V-Pins zu vermeiden).

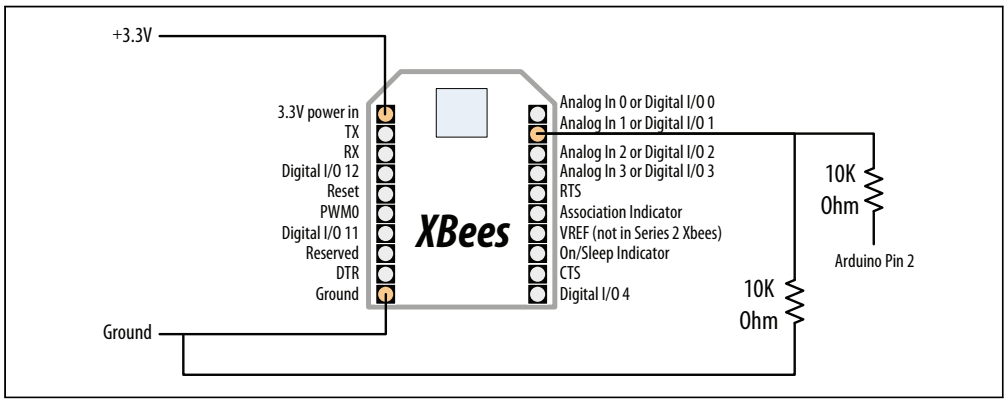


Abbildung 14-10: Anschluss des Arduino an E/A-Pin 1 des sendenden Serie 1-XBee

Nun laden Sie den folgenden Sketch auf den sendenden Arduino hoch. Der Sketch schaltet den digitalen E/A-Pin des XBee abwechselnd ein (5) und aus (4). Da der sendende XBee so eingestellt ist, dass er die Pin-Zustände an den empfangenden XBee weitergibt, ändern sich die Zustände auch auf dem empfangenden XBee:

```
/*
 XBeeActuateSeries1
```

```
 Digitalpin eines anderen XBees aktivieren.
*/

const int xbeePin = 2;

void setup() {
 pinMode(xbeePin, OUTPUT);
}

void loop()
{

 digitalWrite(xbeePin, HIGH);
 delay(3000);
 digitalWrite(xbeePin, LOW);
 delay(3000);
}
```

Siehe auch

Rezept 14.2

14.6 Nachrichten über Low-Cost-Transceiver senden

Problem

Sie wünschen sich eine Low-Cost-Drahtlos-Lösung, die mehr kann als die einfachen Module in Rezept 14.1.

Lösung

Verwenden Sie die zunehmend beliebter werdenden Hope RFM12B-Module zum Senden und Empfangen von Daten. Das Rezept verwendet zwei Arduino-Boards und Drahtlos-Module. Ein Paar liest und sendet Werte, das andere gibt die empfangenen Werte aus. Die Verschaltung ist bei beiden gleich.

Schließen Sie die Module wie in Abbildung 14-11 zu sehen an. Die Antenne ist nur ein Stück Draht, zugeschnitten auf die richtige Länge für die Frequenz Ihrer Module (78 mm für 915 MHz, 82 mm für 868 MHz und 165 mm für 433 MHz).

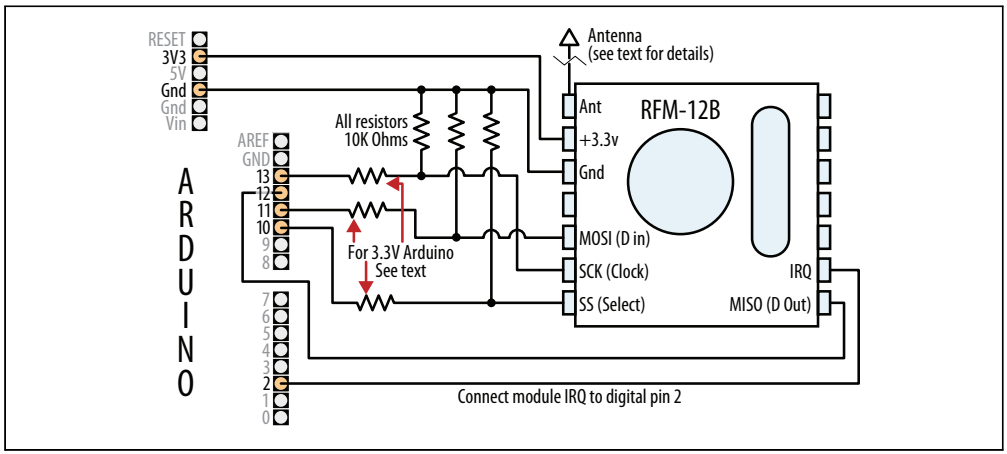


Abbildung 14-11: Anschluss eines RFM12B-Transceivers

Wenn Sie einen 3,3V-Arduino nutzen (wie den Fio oder den 3,3V-Arduino-Pro), lassen Sie die Widerstände weg und verbinden Sie die Arduino-Pins 10, 11 und 13 direkt mit den entsprechenden RFM12B-Pins.

Der Sender-Sketch sendet einmal pro Sekunde die Werte der sechs Analogpins:

```
/*
 * SimpleSend
 * RFM12B Wireless-Demo - Sender - kein ACK
 * Sendet die Werte der Analogeingänge 0 bis 5
 *
 */

#include <RF12.h> // Von jeelabs.org
#include <Ports.h>

// RF12B constants:
const byte network = 100;  // Netzwerk-Gruppe (im Bereich von 1-255)
const byte myNodeID = 1;   // Eindeutige Knoten-ID des Empfängers (zwischen 1 und 30)

// Frequenz des RF12B kann RF12_433MHZ, RF12_868MHZ oder RF12_915MHZ sein.
const byte freq = RF12_868MHZ; // Frequenz an Modul anpassen

const byte RF12_NORMAL_SENDWAIT = 0;

void setup()
{
 rf12_initialize(myNodeID, freq, network);  // RFM12 initialisieren
}

const int payloadCount = 6; // Zahl der Integerwerte in den Nutzdaten den gelesenen
                            // Analogeingängen anpassen
int payload[payloadCount];

void loop()
{
```

```
 for( int i= 0; i < payloadCount; i++)
 {
  payload[i] = analogRead(i);
 }
 while (!rf12_canSend()) // Treiber zum Senden bereit?
  rf12_recvDone();     // Nein, warten

 rf12_sendStart(rf12_hdr, payload, payloadCount*sizeof(int));
 rf12_sendWait(RF12_NORMAL_SENDWAIT); // Warten, bis Senden abgeschlossen ist

 delay(1000); // Einmal pro Sekunde senden
}
```

Der Empfänger-Sketch gibt die sechs Analogwerte über den seriellen Monitor aus:

```
/*
 * SimpleReceive
 * RFM12B Wireless-Demo - Empfänger - kein ACK
 *
 */

#include <RF12.h>   // Von jeelabs.org
#include <Ports.h>

// RFM12B constants:
const byte network  = 100;  // Netzwerk-Gruppe (im Bereich von 1-255)
const byte myNodeID = 2;    // Eindeutige Knoten-ID des Empfängers (1 bis 30)

// Frequenz des RFM12B kann RF12_433MHZ, RF12_868MHZ oder RF12_915MHZ sein
const byte freq = RF12_868MHZ; // Frequenz an Modul anpassen

void setup()
{
 rf12_initialize(myNodeID,freq,network);  // RFM12 mit obigen Einstellungen initialisieren
 Serial.begin(9600);
 Serial.println("RFM12B-Empfaenger bereit");
 Serial.println(network,DEC);  // Netzwerk- und
 Serial.println(myNodeID,DEC); // Knoten-ID ausgeben
}

const int payloadCount = 6; // Zahl der Integerwerte in den Nutzdaten  den gelesenen
                            // Analogeingängen anpassen

void loop()
{
 if (rf12_recvDone() && rf12_crc == 0 && (rf12_hdr & RF12_HDR_CTL) == 0)
 {
  int *payload = (int*)rf12_data;  // Zugriff auf rf12-Datenpuffer über Array von ints
  for( int i= 0; i < payloadCount; i++)
  {
   Serial.print(payload[i]);
   Serial.print(" ");
  }
  Serial.println();
 }
}
```

Diskussion

Die RFM12B-Module sind für 3,3V konzipiert und die Widerstände in Abbildung 14-11 werden benötigt, um die Spannung auf den richtigen Pegel zu bringen. Die JeeLabs-Website*http://jeelabs.com/products/rfm12b-board* enthält Details zu Breakout-Boards und Modulen für den RFM12B.

Die RF12-Bibliothek unterstützt unterschiedliche Gruppen von Modulen in der gleichen Umgebung. Jede Gruppe wird über eine Netzwerk-ID identifiziert. Die Sender- und Empfänger-Sketches müssen die gleiche Netzwerk-ID verwenden, um miteinander kommunizieren zu können. Jeder Knoten muss eine eindeutige ID innerhalb des Netzwerks besitzen. In diesem Beispiel verwenden wird das Netzwerk 100 mit der Sender-ID 1 und der Empfänger-ID 2.

Der loop-Code füllt ein Array (siehe Rezept 2.4) namens `payload` mit sechs Integerwerten, die von den Analogeingängen 0 bis 5 eingelesen wurden.

Das Senden erfolgt über den Aufruf von `rf12_sendStart`. Das Argument `rf12-hdr` bestimmt den Zielknoten, der mit 0 voreingestellt ist. (Das Senden an Knoten 0 gibt die Daten an alle Knoten im Netzwerk weiter); `&payload` ist die Adresse des Nutzdaten-Puffers. `payloadCount * sizeof(int)` ist die Anzahl der Bytes im Puffer. `rf12_sendWait` wartet, bis das Senden abgeschlossen ist (Stromsparoptionen finden Sie in der RF12-Dokumentation).

Dieser Code bestätigt den Empfang der Nachrichten nicht. Bei Anwendungen wie diesen, bei denen fortlaufend Informationen gesendet werden, ist der gelegentliche Verlust von Informationen aber kein Problem, da sie beim nächsten Senden aktualisiert werden. Im Beispiel-Code der Bibliothek finden Sie Sketches, die andere Techniken zum Senden und Empfangen der Daten verwenden.

Alle Arten von Daten, die in einen 66-Byte-Puffer passen, können gesendet werden. Der folgende Sketch sendet zum Beispiel eine binäre Datenstruktur, die aus einem Integer- und einem Fließkommawert besteht:

```
/*
 * RFM12B Wireless-Demo - struct-Sender - kein ACK
 * Sendet einen Fließkommawert in einer C-Struktur
 */

#include <RF12.h> // Von jeelabs.org
#include <Ports.h>

// RF12B constants:
const byte network  = 100;  // Netzwerk-Gruppe (im Bereich von 1-255)
const byte myNodeID = 1;    // Eindeutige Knoten-ID des Empfängers (1 bis 30)

// Frequenz des RF12B kann RF12_433MHZ, RF12_868MHZ oder RF12_915MHZ sein
const byte freq = RF12_868MHZ; // Frequenz an Modul anpassen

const byte RF12_NORMAL_SENDWAIT = 0;

void setup()
```

```
{
 rf12_initialize(myNodeID, freq, network);  // RFM12 initialisieren
}

typedef struct {  // Datenstruktur der Nachricht, muss Tx entsprechen
 int pin;  // Zur Messung verwendeter Pin
 float value;  // Messwert als Fließkommazahl
}
Payload;

Payload sample;  // Instanz vom Typ Payload deklarieren

void loop()
{
 int inputPin = 0; // Der Eingangspin
 float value = analogRead(inputPin) * 0.01; // Ein Fließkommawert
 sample.pin = inputPin; // send demontx.ct1=emontx.ct1+1;
 sample.value = value;

 while (!rf12_canSend())  // Treiber zum Senden bereit?
  rf12_recvDone();     // Nein, warten

 rf12_sendStart(rf12_hdr, &sample, sizeof sample);
 rf12_sendWait(RF12_NORMAL_SENDWAIT);  // Warten, bis Senden abgeschlossen ist

 Serial.print(sample.pin);
 Serial.print(" = ");
 Serial.println(sample.value);
 delay(1000);
}
```

Hier der Sketch, der die struct-Daten empfängt und ausgibt:

```
/*
 * RFM12B Wireless-Demo - struct-Empfänger - kein ACK
 *
 */


#include <RF12.h>  // Von jeelabs.org
#include <Ports.h>

// RF12B constants:
const byte network  = 100;  // Netzwerk-Gruppe (zwischen 1-255)
const byte myNodeID = 2;    // Eindeutige Knoten-ID des Empfängers (1 bis 30)

// Frequenz des RF12B kann RF12_433MHZ, RF12_868MHZ oder RF12_915MHZ sein
const byte freq = RF12_868MHZ; // Frequenz an Modul anpassen

void setup()
{
 rf12_initialize(myNodeID,freq,network);  // RFM12 mit obigen Einstellungen initialisieren
 Serial.begin(9600);
 Serial.print("RFM12B-Empfänger bereit");
}

typedef struct {  // Datenstruktur der Nachricht, muss Tx entsprechen
```

```
 int  pin;  // Zur Messung verwendete Pin-Nummer
 float value;  // Messwert als Fließkommazahl
}
Payload;

Payload sample;      // Instanz vom Typ Payload deklarieren

void loop() {

 if (rf12_recvDone() && rf12_crc == 0 && (rf12_hdr & RF12_HDR_CTL) == 0)
 {
  sample = *(Payload*)rf12_data;       // Nutzdaten abrufen
  Serial.print("Analogeingang ");
  Serial.print(sample.pin);
  Serial.print(" = ");
  Serial.println(sample.value);
 }
}
```

Dieser Code ähnelt den beiden vorigen Sketches, doch diesmal wurde der `payload`-Puffer durch einen Zeiger namens `sample` ersetzt, der auf die `Payload`-Struktur verweist.

Siehe auch

Die in diesem Rezept verwendeten Bibliotheken wurden von Jean-Claude Wippler entwickelt. Eine Fülle von Informationen finden Sie auf der Website *http://www.jeelabs.com*.

Alle Funktionen der RF12-Bibliothek sind hier dokumentiert: *http://jeelabs.net/projects/cafe/wiki/RF12*.

Einen Beispiel-Sketch zum Senden von Strings mit dem RFM12 finden Sie hier: *http://jeelabs.org/2010/09/29/sending-strings-in-packets*.

Ein Beispiel für den Ruhemodus (als Stromsparmodus) zwischen den Sendeoperationen finden Sie hier: *https://github.com/openenergymonitor/emonTxFirmware*.

Ein Breakout-Board für den RFM12B finden Sie hier: *http://jeelabs.com/products/rfm12b-board*.

Das JeeNode ist ein Board, das den RFM12B und einen Arduino-kompatiblen Chip kombiniert: *http://http://jeelabs.com/products/jeenode*.

Eine 433 MHz-Version des RFM12B, die auf der ganzen Welt funktionieren sollte, finden Sie bei SparkFun: *http://www.sparkfun.com/products/9582*.

14.7 Mit Bluetooth-Geräten kommunizieren

Problem

Sie wollen Informationen per Bluetooth an/von andere(n) Geräte(n) (wie Laptops oder Mobiltelefone) senden oder empfangen.

Lösung

Verbinden Sie den Arduino mit einem Bluetooth-Modul wie dem BlueSMiRF, dem Bluetooth Mate oder dem Bluetooth Bee (siehe Abbildung 14-12).

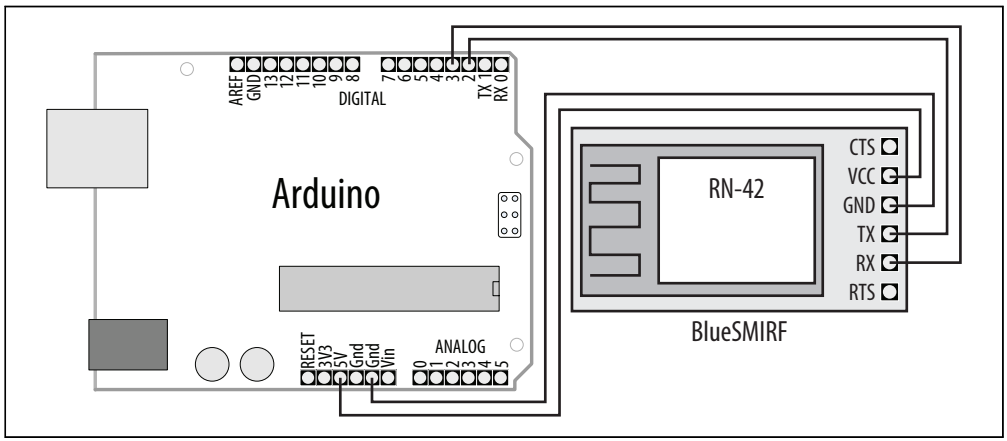


Abbildung 14-12: Anschluss eines BlueSMiRF Bluetooth-Moduls über SoftwareSerial-Pins

Dieser Sketch ist dem aus Rezept 4.13 ähnlich. Er überwacht, welche Zeichen über den seriellen Hardware-Port und einen Software-Port (der mit Bluetooth verbunden ist) eingehen und sendet alle empfangenen Zeichen an den jeweils anderen Port:

```
/*
 * Per SoftwareSerial mit BlueSMiRF-Modul kommunizieren
 * Pairing-ID ist 1234
 */

#include <SoftwareSerial.h>

const int rxpin = 2;       // Empfänger-Pin
const int txpin = 3;       // Sender-Pin
SoftwareSerial bluetooth(rxpin, txpin); // Neuer serieller Port an festgelegten Pins

void setup()
{
 Serial.begin(9600);
 bluetooth.begin(9600); // Seriellen Software-Port initialisieren
 Serial.println("Seriell bereit");
 bluetooth.println("Bluetooth bereit");
}
```

```
void loop()
{
 if (bluetooth.available())
 {
  char c = (char)bluetooth.read();
  Serial.write(c);
 }
 if (Serial.available())
 {
  char c = (char)Serial.read();
  bluetooth.write(c);
 }
}
```

Diskussion

Ihr Computer (oder Telefon) muss Bluetooth-fähig sein, um mit diesem Sketch kommunizieren zu können. Beide Seiten einer Bluetooth-Konversation müssen »gekoppelt« werden (das sog. Pairing) – die ID des mit dem Arduino verbundenen Moduls muss der anderen Seite bekannt sein. Die Standard-ID für das BlueSMiRF ist 1234. In der Dokumentation Ihres Computers/Telefons können Sie nachlesen, wie man die Pairing-ID setzt und eine Verbindung herstellt.

Wenn Ihr Board an ein FTDI-Kabel aufgesteckt werden kann, können Sie es direkt mit dem Bluetooth-Mate-Modul verbinden (siehe Abbildung 14-13).

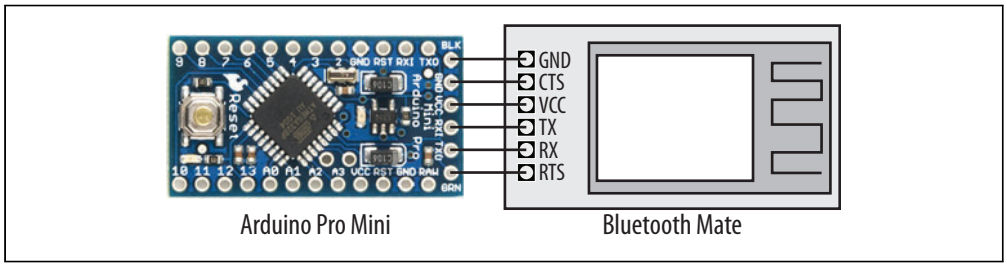


Abbildung 14-13: Bluetooth Mate verwendet die gleichen Anschlüsse wie FTDI

Das Bluetooth Mate kann auch an ein Standard-Board angeschlossen werden, wie Abbildung 14-14 zeigt.

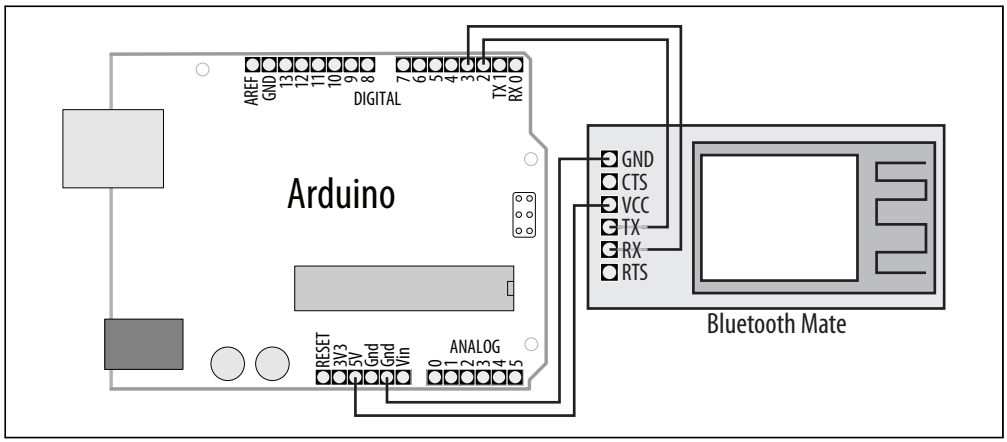


Abbildung 14-14: Anschluss von Bluetooth Mate für SoftwareSerial

Alle gängigen Bluetooth-Module für den Arduino implementieren das Bluetooth Serial Port Profile (SPP). Sobald die Geräte gekoppelt sind, betrachtet der Computer oder das Telefon das Modul als seriellen Port. Diese Module können nicht als andere Bluetooth-Geräte, etwa als Maus oder Tastatur, auftreten.

Die Reichweite von Bluetooth liegt zwischen 5 und 100 Metern, je nachdem, ob Sie mit Klasse-3-, -2- oder -1-Geräten arbeiten.

Siehe auch

Ein SparkFun-Tutorial behandelt die Installation und Verwendung von Bluetooth: *http://www.sparkfun.com/tutorials/67*

Bluetooth Bee ist ein Bluetooth-Modul, das in einen XBee-Sockel passt, d.h., Sie können für XBee entwickelte Shields und Adapter verwenden: *http://www.seeedstudio.com/depot/bluetooth-bee-p-598.html*.

KAPITEL 15

Ethernet und Netzwerke

15.0 Einführung

Sie wollen Ihre Sensordaten teilen? Die Aktionen Ihres Arduino durch andere Leute steuern lassen? Ihr Arduino kann über Ethernet und Netzwerke mit einem breiteren Publikum kommunizieren. Dieses Kapitel beschreibt die vielen Möglichkeiten, mit denen Sie den Arduino mit dem Internet nutzen können. Die Beispiele demonstrieren, wie Sie Web-Clients und -Server aufbauen und nutzen und wie man die gängigsten Internet-Kommunikationsprotokolle mit dem Arduino verwendet.

Das Internet erlaubt es einem Client (z.B. Ihrem Web-Browser), Informationen von einem Server (einem Web-Server oder einem anderen Internet-Serviceanbieter) abzurufen. Dieses Kapitel enthält Rezepte, die zeigen, wie man einen Internet-Client aufbaut, der Informationen von Diensten wie Google oder Yahoo! abruft. Andere Rezepte zeigen, wie der Arduino als Internet-Server fungieren kann, der Informationen über Internetprotokolle an Clients ausliefert, und wie man einen Web-Server aufbaut, der Webseiten an Web-Browser zurückgibt.

Die Arduino Ethernet-Bibliothek unterstützt eine Reihe von Methoden (Protokollen), die es Ihrem Sketch ermöglichen, als Internet-Client oder -Server zu fungieren. Die Ethernet-Bibliothek verwendet eine Reihe von Standard-Internet-Protokollen und versteckt einen Großteil der auf unterster Ebene angesiedelten Details. Clients und Server zum Laufen zu bringen und nützliche Dinge machen lassen, verlangt ein grundlegendes Verständnis der Netzwerkadressierung und -Protokolle. Sie könnten sich eine der vielen Einführungen im Netz ansehen oder eines der folgenden einführenden Bücher:

- *Netzwerke von Kopf bis Fuß* (ISBN 978-3-89721-944-1) von Al Anderson und Ryan Benedetti (O'Reilly)
- *Network Know-How: An Essential Guide for the Accidental Admin* von John Ross (No Starch Press)
- *Windows NT TCP/IP Network Administration* von Craig Hunt und Robert Bruce Thompson (O'Reilly)
- *Making Things Talk* (ISBN 978-3-86899-162-8) von Tom Igoe (O'Reilly)

(Suchen Sie nach O'Reilly-Titeln auf www.oreilly.de.)

Hier einige Schlüsselkonzepte dieses Kapitels, mit denen Sie sich ausführlicher auseinandersetzen sollten, als das hier möglich ist:

Ethernet
: Die auf unterster Ebene angesiedelte Signalisierungsschicht, die für die grundlegende physikalische Nachrichtenübertragung sorgt. Die Quell- und Zieladressen dieser Nachrichten werden über die MAC-Adresse (Media Access Control) festgelegt. Ihr Arduino-Sketch definiert eine MAC-Adresse, die innerhalb Ihres Netzwerks eindeutig sein muss.

TCP und IP
: Transmission Control Protocol (TCP) und Internet Protocol (IP) sind die Kernprotokolle des Internet, die direkt über Ethernet ansetzen. Sie stellen die Nachrichtenübertragung zur Verfügung, die über das globale Internet läuft. TCP/IP-Nachrichten werden zwischen eindeutigen IP-Adressen für Sender und Empfänger ausgeliefert. Ein Server im Internet verwendet einen numerischen Wert (eine Adresse), die kein anderer Server besitzt, so dass er eindeutig identifiziert werden kann. Diese Adresse besteht aus vier Bytes, die üblicherweise durch Punkte voneinander getrennt sind (z.B. ist 64.233.187.64 eine IP-Adresse, die von Google verwendet wird). Das Internet verwendet das Domain Name System (DNS), um gängige Dienstnamen (*http://www.google.com*) in numerische IP-Adressen umzuwandeln. Diese Fähigkeit wurde im Arduino 1.0 eingeführt. Rezept 15.3 zeigt, wie Sie das in Ihren Sketches nutzen können.

Lokale IP-Adressen
: Ist in Ihrem Heimnetzwerk mehr als ein Computer über einen Breitband-Router oder ein Gateway angeschlossen, verwendet jeder Rechner wahrscheinlich eine lokale IP-Adresse, die von Ihrem Router zur Verfügung gestellt wird. Diese lokale Adresse wird mit Hilfe des DHCP-Dienstes (Dynamic Host Configuration Protocol) Ihres Routers erzeugt. Die Arduino Ethernet-Bibliothek enthält nun (seit Release 1.0) einen DHCP-Dienst. Die meisten Rezepte in diesem Kapitel verwenden eine vom Benutzer festgelegte IP-Adresse, die Sie an Ihr Netzwerk anpassen müssen. Rezept 15.2 zeigt, wie die IP-Adresse automatisch über DHCP bezogen werden kann.

Die Web-Requests eines Web-Browsers und die daraus resultierenden Antworten versenden HTTP-Nachrichten (Hypertext Transfer Protocol). Damit ein Web-Client oder -Server korrekt auf HTTP-Requests und -Responses reagieren kann, muss er das Protokoll verstehen. Viele Rezepte in diesem Kapitel nutzen dieses Protokoll. Die oben aufgeführten Referenzen enthalten weitere Details, die Ihnen helfen zu verstehen, wie diese Rezepte im Detail funktionieren.

Webseiten werden üblicherweise in HTML (Hypertext Markup Language) formatiert. Zwar ist der Einsatz von HTML beim Aufbau eines Arduino-Webservers nicht zwingend (wie Rezept 15.9 zeigt), aber die von Ihnen ausgelieferten Webseiten können diese Fähigkeit nutzen.

Daten aus einer Webseite zu extrahieren, die man sich eigentlich mit einem Web-Browser ansehen soll, ist wegen der ganzen zusätzlichen Texte, Bilder und Format-Tags einer

typischen Seite ein wenig wie die Suche nach der Stecknadel im Heuhaufen. Diese Aufgabe kann durch die Stream-Parsing-Funktionalität von Arduino 1.0 vereinfacht werden, die bestimmte Zeichenfolgen aufspüren und Strings oder numerische Daten aus einem Datenstrom herausfiltern kann. Wenn Sie mit einer älteren Arduino-Release arbeiten, können Sie eine Bibliothek namens TextFinder aus dem Arduino Playground herunterladen. TextFinder extrahiert Informationen aus einem Datenstrom. Stream-Parsing und TextFinder bieten eine vergleichbare Funktionalität (Stream-Parsing basiert auf dem TextFinder-Code, den die erste Ausgabe dieses Buches nutzt). Allerdings wurden einige Methoden umbenannt. In der TextFinder-Dokumentation im Playground finden Sie Hilfe, wenn Sie Sketches von TextFinder nach Arduino 1.0 migrieren wollen.

Web-Austauschformate wurden entwickelt, um eine zuverlässige Extrahierung von Web-Daten per Software zu ermöglichen. XML and JSON sind zwei der am weitesten verbreiteten Formate und Rezept 15.5 zeigt, wie man sie mit dem Arduino nutzen kann.

Arduino 1.0 Enhancements

Die Arduino Ethernet-Bibliothek hat in der 1.0-Release einige Verbesserungen erfahren, die ihre Nutzung vereinfachen. Darüber hinaus wurden Dienste wie DHCP und DNS hinzugefügt, für die früher der Download zusätzlicher Bibliotheken notwendig war. Bei einigen Klassen und Methoden hat sich der Name geändert, d.h., für ältere Releases geschriebene Sketches müssen angepasst werden, um unter Arduino 1.0 zu funktionieren. Hier eine Zusammenfassung der Änderungen, die für ältere Sketches notwendig sind:

- `SPI.h` muss (seit Arduino 0018) vor dem Ethernet-Include am Anfang des Sketches stehen.
- `Client client(server, 80);` wird zu `EthernetClient client;`.
- `if(client.connect())`wird zu `if(client.connect(serverName, 80)>0)`.
- `Server server(80)` wird zu `EthernetServer server(80)`.
- DHCP benötigt keine externe Bibliothek (siehe Rezept 15.2).
- DNS benötigt keine externe Bibliothek (siehe Rezept 15.3).
- Suche nach Wörtern und Zahlen wird durch das neue Stream-Parsing vereinfacht (siehe Rezept 15.4).
- `F(text)`-Konstrukt hinzugefügt, um das Speichern von Text im Flash-Speicher zu vereinfachen (Rezept 15.11).

Der Code dieses Kapitels ist für die Arduino Release 1.0 gedacht. Wenn Sie eine ältere Version verwenden, laden Sie den Code der ersten Ausgabe über *http://oreilly.com/catalog/9780596802486* herunter.

Der Code für dieses Buch wurde mit den Release Candidates von Arduino 1.0 getestet. Mögliche Updates an Sketches sind in der Datei *changelog.txt* *http://shop.oreilly.com/product/0636920022244.do* aufgeführt.

Alternative Hardware für kostengünstige Netzwerke

Wenn Sie ein kostengünstiges und Eigenbau-freundliches Ethernet-Board brauchen, das ohne SMD-Technik auskommt, können Sie das Open-Source-Design nutzen, das für das Nanode-Projekt entwickelt wurde. Es nutzt den gleichen ATmega328-Controller wie der Arduino, ersetzt aber den Wiznet-Chip durch den günstigeren ENC28J60. Dieser Chip bietet die gleiche Funktionalität, die in diesem Kapitel beschrieben wird, verwendet aber andere Bibliotheken, d.h., Sie müssen mit Sketches arbeiten, die speziell für den ENC28J60 entwickelt wurden.

Weitere Informationen finden Sie auf der Nanode-Homepage: *http://www.nanode.eu/*.

15.1 Ein Ethernet-Shield einrichten

Problem

Sie wollen ein Ethernet-Shield mit einer fest kodierten IP-Adresse einrichten.

Lösung

Dieser Sketch basiert auf dem Ethernet-Client Beispiel-Sketch, das mit Arduino mitgeliefert wird. Stellen Sie sicher, dass Sie eine für Ihr Netzwerk gültige Arduino-IP-Adresse (der Wert der ip-Variablen) verwenden:

```
/*
 * Simple Web Client
 * Arduino 1.0-Version
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 177 };   // Auf gültige Adresse achten
byte server[] = { 209,85,229,104  }; // Google
                    // Mehr zur IP-Adressierung erfahren Sie im Text

EthernetClient client;

void setup()
{
 Serial.begin(9600);    // Seriellen Port starten
 Ethernet.begin(mac,ip);
 delay(1000);        // Der Ethernet-Hardware eine Sekunde zur Initialisierung geben

 Serial.println("Verbinde...");

 if (client.connect(server, 80)) {
  Serial.println("Verbunden");
  client.println("GET /search?q=arduino HTTP/1.0"); // Der HTTP-Request
  client.println();
 }
```

```
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
}

void loop()
{
 if (client.available()) {
  char c = client.read();
  Serial.print(c); // Alle empfangenen Daten über seriellen Monitor ausgeben
 }

 if (!client.connected()) {
  Serial.println();
  Serial.println("Trenne Verbindung...");
  client.stop();
  for(;;)
   ;
 }
}
```

Diskussion

Der Sketch führt eine Google-Suche nach dem Wort »arduino« durch. Sein Zweck besteht darin, Sie mit funktionierendem Code zu versorgen, mit dem Sie prüfen können, ob Ihre Netzwerk-Konfiguration für das Arduino Ethernet-Shield funktioniert.

Bis zu vier Adressen müssen richtig konfiguriert werden, damit der Sketch eine Verbindung herstellen und das Ergebnis der Suche über den seriellen Monitor ausgeben kann:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Die MAC-Adresse identifiziert Ihr Ethernet-Shield eindeutig. Jedes Netzwerk-Gerät muss eine andere MAC-Adresse verwenden und wenn es in Ihrem Netzwerk mehr als ein Arduino-Shield gibt, muss jedes eine andere Adresse nutzen. Bei neueren Ethernet-Shields ist die MAC-Adresse auf einem Aufkleber auf der Unterseite aufgedruckt. Bei nur einem Ethernet-Shield müssen Sie sie nicht ändern.

```
byte ip[] = { 192, 168, 1, 177 };    // Auf gültige Adresse achten
```

Die IP-Adresse wird genutzt, um etwas zu identifizieren, was über das Internet kommuniziert. Sie muss innerhalb Ihres Netzwerks ebenfalls eindeutig sein. Die Adresse besteht aus vier Bytes und der Bereich gültiger Werte hängt von der Konfiguration Ihres Netzwerks ab. IP-Adressen werden üblicherweise mit Punkten dargestellt, die die einzelnen Bytes trennen – zum Beispiel 192.168.1.177. In allen Arduino-Sketches werden Kommata anstelle von Punkten verwendet, da die Bytes in einem Array abgelegt sind (siehe Rezept 2.4).

Wenn ihr Netzwerk über einen Router oder ein Gateway mit dem Internet verbunden ist, müssen Sie möglicherweise die IP-Adresse des Gateways beim Aufruf von `ethernet.begin` übergeben. Sie finden die Adresse des Gateways in der Dokumentation Ihres Routers/

Gateways. Fügen Sie zwei Zeilen hinter den IP- und Server-Adressen im Sketch ein. Eine mit der Adresse Ihres DNS-Servers, und die andere mit der Gateway-Adresse:

```
// falls von Router oder Gateway verlangt
byte dns_server[] = { 192, 168, 1, 2 }; // Adresse des DNS-Servers
byte gateway[] = { 192, 168, 1, 254 }; // Adresse des Gateways
```

Passen Sie die erste Zeile in `setup` so an, dass die Gateway-Adresse in der Ethernet-Initialisierung enthalten ist:

```
Ethernet.begin(mac, ip, dns_server, gateway);
```

Die Server-Adresse besteht auf der 4-Byte-IP-Adresse des Servers, mit dem Sie die Verbindung herstellen wollen – in diesem Fall also Google. Server-IP-Adressen ändern sich gelegentlich, d.h., Sie müssen eventuell das ping-Utility Ihres Betriebssystems nutzen, um die aktuelle IP-Adresse des gewünschten Servers zu ermitteln:

```
byte server[] = { 64, 233, 187, 99 }; // Google
```

Die Zeile zu Beginn des Sketches, die `<SPI.h>` einbindet, wird seit Arduino-Release 0019 verlangt.

Siehe auch

Die Web-Referenz zum Arduino Ethernet-Shield finden Sie unter *http://arduino.cc/en/Guide/ArduinoEthernetShield*.

15.2 Die IP-Adresse automatisch beziehen

Problem

Die vom Ethernet-Shield verwendete IP-Adresse muss innerhalb Ihres Netzwerks eindeutig sein und Sie wollen sie automatisch beziehen. Das Ethernet-Shield soll die IP-Adresse von einem DHCP-Server abrufen.

Lösung

Dieses Rezept ähnelt dem Sketch aus Rezept 15.1, übergibt aber keine IP-Adresse an die `Ethernet.begin`-Methode:

```
/*
 * Einfacher Client zur Ausgabe der vom DHCP-Server zugewiesenen IP-Adresse
 * Arduino-1.0-Version
 */
```

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte server[] = { 209,85,229,104 }; // Google

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet mit mac & DHCP starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben

 Serial.print("IP-Adresse: ");
 IPAddress myIPAddress = Ethernet.localIP();
 Serial.print(myIPAddress);
 if(client.connect(server, 80)>0) {
  Serial.println(" verbunden");
  client.println("GET /search?q=arduino HTTP/1.0");
  client.println();
 } else {
  Serial.println("Verbindung fehlgeschlagen");
 }
}

void loop()
{
 if (client.available()) {
  char c = client.read();
  // Kommentarzeichen der nachfolgenden Zeile entfernen, um alle empfangenen Zeichen auszugeben
  // Serial.print(c);
 }

 if (!client.connected()) {
  Serial.println();
  Serial.println("Trenne Verbindung");
  client.stop();
  for(;;)
   ;
 }
}
```

Diskussion

Die mit Arduino 1.0 mitgelieferte Bibliothek unterstützt nun DHCP (frühere Releases benötigten die Bibliothek eines Drittanbieters von *http://blog.jordanterrell.com/post/Arduino-DHCP-Library-Version-04.aspx*.

Der Hauptunterschied zum Sketch in Rezept 15.1 besteht darin, dass es keine Variable für die IP-Adresse (oder das Gateway) gibt. Diese Werte werden vom DHCP-Server abge-

rufen, wenn der Sketch startet. Zusätzlich wird überprüft, ob der Aufruf von `ethernet.begin` erfolgreich war. Nur so können Sie sicherstellen, dass eine gültige IP-Adresse vom DHCP-Server bereitgestellt wurde (ohne eine gültige IP-Adresse ist kein Zugang zum Internet möglich).

Der Code gibt die IP-Adresse im seriellen Monitor über die Methode `IPAddress.printTo` aus, die bei Arduino 1.0 eingeführt wurde:

```
Serial.print("IP-Adresse: ");
IPAddress myIPAddress = Ethernet.localIP();
Serial.print(myIPAddress);
```

Das an `Serial.print` übergebene Argument mag etwas seltsam wirken, doch die neue `IPAddress`-Klasse ist in der Lage, ihren Wert an `Serial` zu übergeben, die von der `Print`-Klasse abgeleitet sind.

Wenn Sie mit der Ableitung von Klassen nicht vertraut sind, reicht es, zu sagen, dass das `IPAddress`-Objekt clever genug ist, bei Bedarf seine Adresse auszugeben.

15.3 Hostnamen in IP-Adressen umwandeln (DNS)

Problem

Sie wollen einen Servernamen wie yahoo.com anstelle einer festen IP-Adresse verwenden. Web-Dienste nutzen häufig mehrere IP-Adressen für ihre Server und die von Ihnen angegebene Adresse muss nicht aktiv sein, wenn Sie die Verbindung herstellen wollen.

Lösung

Sie können DNS nutzen, um eine gültige IP-Adresse für den von Ihnen angegebenen Namen zu ermitteln:

```
/*
 * Web Client DNS Sketch
 * Arduino 1.0-Version
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char serverName[] = "www.google.com";

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 if (Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
```

```
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben

 int ret = client.connect(serverName, 80);
 if (ret == 1) {
  Serial.println("Verbunden"); //  Erfolgreiche Verbindung melden
  // Make an HTTP request:
  client.println("GET /search?q=arduino HTTP/1.0");
  client.println();
 }
 else {
  Serial.println("Verbindung fehlgeschlagen: ");
  Serial.print(ret,DEC);
 }
}

void loop()
{
 // Eingehende Bytes vom Server einlesen und ausgeben
 if (client.available()) {
  char c = client.read();
  Serial.print(c);
 }

 // Client nach Trennung anhalten
 if (!client.connected()) {
  Serial.println();
  Serial.println("Trenne Verbindung");
  client.stop();

  while(true) ; // Endlosschleife
 }
}
```

Diskussion

Der Code entspricht dem aus Rezept 15.2, d.h., er führt eine Google-Suche nach »arduino« aus. Doch bei dieser Version ist es nicht nötig, eine Google-IP-Adresse anzugeben – sie wird über eine Anfrage an den Internet DNS-Dienst ermittelt.

Der Request wird angestoßen, indem wir »www.google.com« anstelle einer IP-Adresse an die `client.connect`-Methode übergeben:

```
char serverName[] = "www.google.com";

int ret = client.connect(serverName, 80);
if(ret == 1) {
  Serial.println("Verbunden"); // Erfolgreiche Verbindung melden
```

Die Funktion gibt 1 zurück, wenn der Hostname vom DNS-Server erfolgreich in eine IP-Adresse aufgelöst werden konnte und wenn die Verbindung vom Client erfolgreich war. Hier die Werte, die von `client.connect` zurückgegeben werden:

```
 1 = Erfolg
 0 = Verbindung fehlgeschlagen
-1 = kein DNS-Server angegeben
-2 = keine DNS-Einträge gefunden
-3 = Timeout
```

Beim Fehlercode –1 müssen Sie den zu nutzenden DNS-Server von Hand festlegen. Die Adresse des DNS-Servers liefert üblicherweise der DHCP-Server, doch wenn Sie das Shield von Hand konfigurieren, müssen Sie ihn angeben (anderenfalls gibt connect –1 zurück).

15.4 Daten von einem Webserver abrufen

Problem

Sie wollen mit dem Arduino Daten von einem Webserver abrufen. Zum Beispiel könnten Sie von einem Webserver gelieferte Werte extrahieren und verarbeiten wollen.

Lösung

Der folgende Sketch nutzt die Yahoo!-Suche, um 50 Kilometer in Meilen umzuwandeln. Er sendet die Anfrage (Query) »50+km+in+mi« und gibt das Ergebnis über den seriellen Monitor aus. Der Sketch funktioniert in dieser Form erst ab Arduino 1.0.1 IDE.

```
/*
 Simple Client Parsing Sketch
 Arduino 1.0-Version
 */
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char serverName[] = "search.yahoo.com";

EthernetClient client;

int result; // Das Ergebnis der Berechnung

void setup()
{
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben

 Serial.println("Verbinde...");
}

void loop()
```

```
{
 if (client.connect(serverName, 80)>0) {
  Serial.print("Verbunden... ");
  client.println("GET /search?p=50+km+in+mi HTTP/1.0");
  client.println();
 } else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
  if(client.find("<b>50 Kilometers")){
   if(client.find("=")){
     result = client.parseInt();
     Serial.print("50 km sind " );
     Serial.print(result);
     Serial.println(" Meilen");
   }
  }
  else
    Serial.println("Ergebnis nicht gefunden");
  client.stop();
  delay(10000); // In 10 Sekunden erneut abfragen
 }
 else {
  Serial.println();
  Serial.println("Nicht verbunden");
  client.stop();
  delay(1000);
 }
}
```

Diskussion

Der Sketch erwartet, dass das Ergebnis mit Fettdruck eingeleitet wird (mit Hilfe des HTML-Tags <b>-Tags). Dann folgen der in der Query angegebene Wert und das Wort *Kilometers*.

Die Suche erfolgt über die Stream-Parsing-Funktionen, die in der Einführung zu diesem Kapitel beschrieben wurden. Die `find`-Methode durchsucht die empfangenen Daten und gibt `true` zurück, wenn der gesuchte String gefunden wurde. Der Code sucht nach Text, der mit der Antwort verknüpft ist. Im Beispiel wird in der folgenden Zeile versucht, den Text »<b>50 Kilometers« zu finden:

```
if (client.find("<b>50 Kilometers")){
```

`client.find` wird erneut genutzt, um das Gleichheitszeichen zu finden, das vor dem numerischen Ergebnis steht.

Das Ergebnis wird mit der Methode `parseInt` herausgefiltert und über den seriellen Monitor ausgegeben.

parseInt liefert einen Integerwert zurück. Wenn Sie einen Fließkomma-Wert einlesen müssen, verwenden Sie stattdessen parseFloat:

```
float floatResult = client.parseInt();
Serial.println(floatResult);
```

Wenn Sie eine robuste Suche brauchen, müssen Sie nach einem eindeutigen Tag Ausschau halten, der nur vor den gewünschten Daten auftaucht. Das ist bei Seiten mit eindeutigen Tags für jedes Feld einfacher zu erreichen. Das folgende Beispiel gibt den Google-Aktienkurs von Google Finance zurück und schreibt den Wert an den Analogausgang 3 (siehe Kapitel 7) und an den seriellen Monitor:

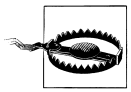
```
/*
 * Web Client Google Finance Sketch
 * Google-Aktienkurs abrufen und an Analogpin 3 schreiben.
 */

#include <SPI.h>    // Für Arduino-Versionen ab 0018 Pflicht
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char serverName[] = "www.google.com";

EthernetClient client;
float value;

void setup()
{
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration über DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben
}

void loop()
{
 Serial.print("Verbinde...");
 if (client.connect(serverName, 80)>0) {
  client.println("GET //finance?q=google HTTP/1.0");
  client.println("User-Agent: Arduino 1.0");
  client.println();
 }
 else
 {
  Serial.println("Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
  if(client.find("<span class=\"pr\">"))
  {
   client.find(">"); // Nächstes '>' suchen
   value = client.parseFloat();
   Serial.print("Google-Kurs steht bei ");
   Serial.println(value); // Wert ausgeben
```

```
  }
  else
   Serial.print("Konnte Feld nicht finden");
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
 delay(5000); // 5 Sekunden warten
}
```

Diese Beispiele verwenden den GET-Befehl, um eine bestimmte Seite abzurufen. Manche Web-Requests müssen Daten im »Rumpf« (Body) der Nachricht an den Server senden, weil mehr Daten übertragen werden, als der GET-Befehl verarbeiten kann. Diese Requests werden über den POST-Befehl verarbeitet. Hier ein Beispiel für den POST-Befehl, der den Babel-Fish-Übersetzungsdienst nutzt, um Text vom Italienischen ins Englische zu übersetzen:

```
/*
 * Web Client Babel Fish Sketch
 * Nutzt Post, um Daten von einem Webserver abzurufen
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char serverName[] = "babelfish.yahoo.com";

EthernetClient client;

// zu übersetzender Text
char * transText = "trtext=Ciao+mondo+da+Arduino.&lp=it_en";

const int MY_BUFFER_SIZE = 30;  // Groß genug für Ergebnis
char buffer [MY_BUFFER_SIZE+1]; // Abschließende Null berücksichtigen

void setup()
{
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben
}

void loop()
{
 Serial.print("Verbinde...");
 postPage( "/translate_txt", transText);
 delay(5000);
}
```

```
void postPage(char *webPage, char *parameter){
 if (client.connect(serverName,80)>0) {
  client.print("POST ");
  client.print(webPage);
  client.println(" HTTP/1.0");
  client.println("Content-Type: application/x-www-form-urlencoded");
  client.println("Host: babelfish.yahoo.com");
  client.print("Content-Length: ");
  client.println(strlen(parameter));
  client.println();
  client.println(parameter);
 }
 else {
  Serial.println(" Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
   client.find("<div id=\"result\">");
   client.find( ">");
   memset(buffer,0, sizeof(buffer)); // clear the buffer
   client.readBytesUntil('<' ,buffer, MY_BUFFER_SIZE);
   Serial.println(buffer);
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
}
```



POST muss die Länge des Inhalts senden, damit der Server weiß, wie viele Daten er zu erwarten hat. Das Fehlen der Länge oder ein falscher Wert sind häufig die Ursachen für Probleme mit POST. In Rezept 15.12 finden Sie ein weiteres Beispiel für einen POST-Request.

Sites wie Google Weather und Google Finance ändern die zur Identifizierung von Feldern genutzten Tags üblicherweise nicht. Doch wenn sich bei der Aktualisierung einer Site die Tags ändern, nach denen Sie suchen, funktioniert der Sketch nicht mehr, bis Sie den Such-Code korrigieren. Eine zuverlässigere Möglichkeit, Daten von einem Webserver zu extrahieren, bietet die Verwendung eines formalen Protokolls wie XML oder JSON. Das nächste Rezept zeigt, wie man Informationen von einer Site extrahiert, die mit XML arbeitet.

15.5 XML-Daten von einem Webserver abrufen

Problem

Sie möchten Daten von einer Site abrufen, die Informationen im XML-Format veröffentlicht. Beispielsweise könnten Sie Werte bestimmter Felder aus einem der Google API-Dienste nutzen wollen.

Lösung

Der folgende Sketch ruft das Wetter in London über Google Weather ab. Er verwendet die Google XML-API:

```
/*
 * Simple Client Google Weather
 * Ruft XML-Daten von  http://www.google.com/ig/api?weather=london,uk
 * Liest die Temperatur aus dem Feld <temp_f data="66" />
 * Schreibt die Temperatur an Analogausgang
 */

#include <SPI.h>      // Für Arduino-Versionen ab 0018 Pflicht
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
char serverName[] = "www.google.com";

const int temperatureOutPin = 3; // Analogausgang für Temperatur
const int humidityOutPin   = 5; // Analogausgang für Luftfeuchtigkeit

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration über DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 delay(1000); // Ethernet-Shield eine Sekunde zur Initialisierung geben

 Serial.println("Verbinde...");
}

void loop()
{
 if (client.connect(serverName,80)>0) {
 // Google-Wetter für London abfragen
  client.println("GET /ig/api?weather=london HTTP/1.0");
  client.println();
 }
 else {
  Serial.println(" Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
   // Temperatur in Fahrenheit ("<temp_c data=" für Celsius)
   if(client.find("<temp_f data=") )
   {
     int temperature = client.parseInt();
     analogWrite(temperatureOutPin, temperature); // Wert an Analogport schreiben
     Serial.print("Temperatur: "); // und über seriellen Port ausgeben
     Serial.println(temperature);
   }
  else
```

```
    Serial.print("Konnte Temperatur-Feld nicht finden");
##!   // Temperatur in Fahrenheit ("<temp_c data=" für Celsius) !##
   if(client.find("<humidity data=") )
   {
    int humidity = client.parseInt();
    analogWrite(humidityOutPin, humidity); // Wert an Analogport schreiben
    Serial.print("Luftfeuchtigkeit: "); // und über seriellen Port ausgeben
    Serial.println(humidity);
   }
   else
    Serial.print("Konnte Luftfeuchtigkeits-Feld nicht finden");
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
 delay(60000); // Eine Minute bis zum nächsten Update warten
}
```

Vor jedem Feld steht ein Tag. Derjenige, der bei Google Weather die Temperatur in Fahrenheit enthält, ist "<temp_f data=".

Wenn Sie die Temperatur in Grad Celsius brauchen, suchen Sie bei dieser Site nach dem Tag "<temp_c data=".

Sie müssen sich die Dokumentation der Sie interessierenden Seite ansehen, um die Tags für die gewünschten Daten zu ermitteln.

Sie wählen die Seite über die Informationen aus, die in Ihrem GET-Befehl gesendet werden. Das ist auch von der jeweiligen Site abhängig, d.h., im obigen Beispiel wird die Stadt in der GET-Anweisung durch den Text hinter dem Gleichheitszeichen festgelegt. Wenn Sie also die Stadt von London in Rom ändern wollen, ändern Sie

```
client.println("GET /ig/api?weather=london HTTP/1.0"); // Wetter für London
```

in:

```
client.println("GET /ig/api?weather=Rome HTTP/1.0"); // Wetter für Rom
```

Sie können eine Variable nutzen, wenn die Stadt vom Programm aus kontrolliert werden soll:

```
char *cityString[4] = { "London", "New%20York", "Rome", "Tokyo"};
int  city;

void loop()
{
 city = random(4); // Stadt zufällig auswählen
 if (client.connect(serverName,80)>0) {
  Serial.print("Wetter fuer ");
  Serial.println(cityString[city]);

  client.print("GET /ig/api?weather=");
  client.print(cityString[city]); // Eine von 4 zufälligen Städten ausgeben
  client.println(" HTTP/1.0");
```

```
 client.println();
}
else {
 Serial.println(" Verbindung fehlgeschlagen");
}
if (client.connected()) {
  // Temperatur in Fahrenheit ("<temp_c data=\"" für Celsius)
  if(client.find("<temp_f data=") )
  {
    int temperature = client.parseInt();
    analogWrite(temperatureOutPin, temperature); // Wert an Analogausgang schreiben
    Serial.print(cityString[city]);
    Serial.print(" Temperatur: "); // und über seriellen Port ausgeben
    Serial.println(temperature);
  }
 else
   Serial.println("Konnte Temperatur-Feld nicht finden");
  // Temperatur in Fahrenheit ("<temp_c data=\"" für Celsius)
  if(client.find("<humidity data=") )
  {
    int humidity = client.parseInt();
    analogWrite(humidityOutPin, humidity);  // Wert an Analogausgang schreiben
    Serial.print("Luftfeuchtigkeit: "); // und über seriellen Port ausgeben
    Serial.println(humidity);
  }
  else
   Serial.println("Konnte Luftfeuchtigkeits-Feld nicht finden");
}
else {
 Serial.println("Verbindung getrennt");
}
client.stop();
client.flush();
delay(60000); // Eine Minute bis zum nächsten Update warten
}

// Der restliche Code entspricht dem des obigen Sketchs
```

In URLs gesendete Informationen dürfen keine Leerzeichen enthalten, weshalb New York als »New%20York« geschrieben werden muss. Die Kodierung für das Leerzeichen ist %20. Ihr Browser übernimmt die Kodierung vor dem Senden des Requests, doch beim Arduino müssen Sie das Leerzeichen selbst durch %20 ersetzen.

15.6 Den Arduino als Webserver einrichten

Problem

Ihr Arduino soll Webseiten ausliefern. Zum Beispiel könnten Sie sich mit Ihrem Web-Browser die Werte der Sensoren ansehen wollen, die mit den Arduino-Analogpins verbunden sind.

Lösung

Dieser Sketch wird mit dem Arduino als Standard-Beispiel für einen Webserver mitgeliefert. Er zeigt die Werte der analogen Eingangspins an. Dieses Rezept erläutert, wie der Sketch funktioniert und wie man ihn erweitern kann:

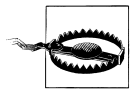
```
/*
 * Web Server
 * Einfacher Webserver, der die Werte der analogen Eingangspins ausgibt.
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 177};  // IP-Adresse des Webservers

EthernetServer server(80);

void setup()
{
 Ethernet.begin(mac, ip);
 server.begin();
}

void loop()
{
 EthernetClient client = server.available();
 if (client) {
  // Ein HTTP-Request endet mit einer Leerzeile
  boolean current_line_is_blank = true;
  while (client.connected()) {
   if (client.available()) {
    char c = client.read();
    // Haben wir das Zeilenende erreicht (ein Newline
    // Zeichen empfangen) und ist die Zeile leer, dann ist der HTTP-Request beendet
    // und wir können eine Antwort senden
    if (c == '\n' && current_line_is_blank) {
     // Standard HTTP-Response-Header senden
     client.println("HTTP/1.1 200 OK");
     client.println("Content-Type: text/html");
     client.println();

     // Werte aller analogen Eingangspins ausgeben
     for (int i = 0; i < 6; i++) {
      client.print("Analogeingang ");
      client.print(i);
      client.print(" ist ");
      client.print(analogRead(i));
      client.println("<br />");
     }
     break;
    }
    if (c == '\n') {
     // Wir beginnen eine neue Zeile
     current_line_is_blank = true;
```

```
      } else if (c != '\r') {
        // Es gibt Zeichen in der aktuellen Zeile
        current_line_is_blank = false;
      }
     }
    }
    // Dem Web-Browser Zeit geben, die Daten zu empfangen
    delay(1);
    client.stop();
   }
  }
```

Diskussion

Wie in Rezept 15.1 diskutiert, benötigen alle Sketches, die die Ethernet-Bibliothek nutzen, eine eindeutige MAC- und IP-Adresse. Die in diesem Sketch zugewiesene IP-Adresse bestimmt die Adresse des Webservers. In diesem Beispiel liefert die Eingabe von 192.168.1.177 im Adressfeld des Browsers eine Seite zurück, die die Werte der analogen Eingangspins 0 bis 5 enthält (in Kapitel 5 erfahren Sie mehr über Analogports).

Wie in der Einführung dieses Kapitel erläutert, ist 192.168.1.177 eine lokale Adresse, die nur innerhalb Ihres lokalen Netzwerks sichtbar ist. Soll der Webserver im Internet verfügbar sein, müssen Sie Ihren Router so konfigurieren, dass er eingehende Nachrichten an den Arduino weitergibt. Diese Technik wird *Port-Weiterleitung (port forwarding)* genannt und Sie müssen in der Dokumentation Ihres Routers nachsehen, wie man sie konfiguriert. Mehr zur Port-Weiterleitung im Allgemeinen erfahren Sie in *SSH, The Secure Shell: The Definitive Guide* von Daniel J. Barrett, Richard E. Silverman und Robert G. Byrnes. Suchen Sie bei www.oreilly.de danach.)



Wenn Sie das Arduino Ethernet-Board so konfigurieren, dass es im Internet sichtbar ist, ist es für jeden zugänglich, der diese spezielle IP-Adresse hat. Die Arduino Ethernet-Bibliothek bietet keine sicheren Verbindungen an, d.h., Sie müssen darauf achten, welche Informationen Sie bereitstellen.

Die beiden Zeilen in `setup` initialisieren die Ethernet-Bibliothek und konfigurieren den Webserver mit der von Ihnen festgelegten IP-Adresse. Der `loop` wartet und verarbeitet alle Requests, die am Webserver eingehen:

```
EthernetClient client = server.available();
```

Das `client`-Objekt ist der Webserver – es verarbeitet die Nachrichten für die IP-Adresse, die Sie dem Server zugewiesen haben.

`if (client)` überprüft, ob der Client erfolgreich gestartet wurde.

`while (client.connected())` überprüft, ob der Webserver mit einem Client verbunden ist, der Daten anfordert.

`client.available()` und `client.read()` prüfen, ob Daten verfügbar sind und lesen ein Byte ein, wenn dass der Fall ist. Das entspricht `Serial.available()`, das in Kapitel 4 diskutiert

wurde, nur dass die Daten über das Internet kommen und nicht über den seriellen Port. Der Code liest die Daten ein, bis er eine Zeile ohne Daten findet, was das Ende eines Requests anzeigt. Ein HTTP-Header wird mit `client.println` ausgegeben, gefolgt von den Werten der Analog-Ports.

15.7 Eingehende Web-Requests verarbeiten

Problem

Sie wollen digitale und analoge Ausgänge mit einem Arduino steuern, der als Webserver fungiert. Zum Beispiel wollen Sie die Werte bestimmter Pins steuern, indem Sie entsprechende Parameter von Ihrem Web-Browser senden.

Lösung

Dieser Sketch liest von einem Browser gesendete Requests ein und ändert die Werte digitaler und analoger Ausgangs-Ports.

Der URL (der vom Browser-Request empfangene Text) besteht aus einem oder mehreren Feldern, die mit dem Wort *pin* beginnen, gefolgt von einem *D* für digital oder *A* für analog und der Pin-Nummer. Der Wert des Pins folgt auf ein Gleichheitszeichen.

Senden Sie zum Beispiel *http://192.168.1.177/?pinD2=1* über die Adresse Ihres Web-Browsers, wird der Digitalpin 2 eingeschaltet. Mit *http://192.168.1.177/?pinD2=0* wird er wieder ausgeschaltet. (In Kapitel 7 finden Sie Informationen zum Anschluss von LEDs an Arduino-Pins.)

Abbildung 15-1 zeigt die Ausgabe des Web-Browsers, wenn Sie die Verbindung mit dem hier entwickelten Webserver-Rezept herstellen.

Abbildung 15-1: Browser-Ausgabe der in diesem Rezept entwickelten Lösung

```
/*
 * WebServerParsing
 * Reagiert auf Requests in der URL zur Änderung digitaler und analoger Ausgänge
 * Gibt die Anzahl geänderter Ports und die Werte der analogen Eingänge aus
 * Beispiel:
 *  http://192.168.1.177/?pinD2=1 schaltet Digitalpin 2 an
 *  http://192.168.1.177/?pinD2=0 schaltet Pin 2 aus
 * Der Sketch demonstriert das Text-Parsing der 1.0 Stream-Klasse
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192,168,1,177 };

EthernetServer server(80);

void setup()
{
 Serial.begin(9600);
 Ethernet.begin(mac, ip);
 server.begin();
 Serial.println("Bereit");
}

void loop()
{
 EthernetClient client = server.available();
 if (client) {
  while (client.connected()) {
   if (client.available()) {
    // Zähler für geänderte Pins
    int digitalRequests = 0;
    int analogRequests = 0;
    if( client.find("GET /") ) {  // Nach 'GET' suchen
     // Mit "pin" beginnende Tokens suchen und bei der ersten Leerzeile aufhören
     // Bis zum Zeilenende nach 'pin' suchen
     while(client.findUntil("pin", "\n\r")){
      char type = client.read(); // D oder A
      // Der nächste ASCII-Integerwert im Stream ist der Pin
      int pin = client.parseInt();
      int val = client.parseInt(); // Die folgende Zahl ist der Wert
      if( type == 'D') {
       Serial.print("Digitalpin ");
       pinMode(pin, OUTPUT);
       digitalWrite(pin, val);
       digitalRequests++;
      }
      else if( type == 'A'){
       Serial.print("Analog-Pin ");
       analogWrite(pin, val);
       analogRequests++;
      }
      else {
       Serial.print("Unbekannter Typ ");
       Serial.print(type);
```

```
      }
      Serial.print(pin);
      Serial.print("=");
      Serial.println(val);
     }
    }
    Serial.println();

    // findUntil hat Leerzeile entdeckt(lf gefolgt von cr),
    // d.h., der HTTP-Request ist abgeschlossen und wir können eine Antwort senden
    // Standard HTTP-Response-Header senden
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println();

    // Anzahl der vom Request verarbeiteten Pins ausgeben
    client.print(digitalRequests);
    client.print(" Digitalpin(s) gesetzt");
    client.println("<br />");
    client.print(analogRequests);
    client.print(" Analog-Pin(s) gesetzt");
    client.println("<br />");
    client.println("<br />");


    // Werte aller analogen Eingangspins ausgeben
    for (int i = 0; i < 6; i++) {
     client.print("Analogeingang ");
     client.print(i);
     client.print(" ist ");
     client.print(analogRead(i));
     client.println("<br />");
    }
    break;
   }
  }
  // Web-Browser Zeit geben, die Daten zu empfangen
  delay(1);
  client.stop();
 }
}
```

Diskussion

Folgendes wurde gesendet: *http://192.168.1.177/?pinD2=1*. Die Information wird dann wie folgt heruntergebrochen: Alles vor dem Fragezeichen wird als Adresse des Webservers betrachtet (in diesem Beispiel 192.168.1.177; diese Adresse ist die IP-Adresse, die Sie zu Beginn des Sketches für das Arduino-Board festgelegt haben). Die restlichen Daten bestehen aus einer Liste von Feldern, die mit dem Wort *pin* beginnen, gefolgt von einem D für einen Digitalpin und einem A für einen Analogpin. Der numerische Wert, der auf das *D* oder *A* folgt, ist die Pin-Nummer. Darauf folgt ein Gleichheitszeichen und schließlich der Wert, auf den der Pin gesetzt werden soll. `pinD2=1` setzt also den Digitalpin 2 auf `HIGH`. Es gibt für jeden Pin ein Feld und die einzelnen Felder sind durch &-Zeichen

(Ampersand) voneinander getrennt. Sie geben so viele Felder an, wie Arduino-Pins geändert werden sollen.

15.8 Das Anfordern bestimmter Seiten verarbeiten

Problem

Sie wollen auf Ihrem Webserver mehr als eine Seite anbieten, z.B. um den Status verschiedener Sensoren auf unterschiedlichen Seiten darzustellen.

Lösung

Der folgende Sketch untersucht Requests für Seiten namens »analog« und »digital« und gibt die entsprechenden Pin-Werte aus:

```
/*
 * WebServerMultiPage
 * Verarbeitet Requests zur Darstellung digitaler und analoger Ausgangs-Ports
 * http://192.168.1.177/analog/  Gibt analoge Pindaten aus
 * http://192.168.1.177/digital/ Gibt digitale Pindaten aus
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] =  { 192,168,1,177 };

const int MAX_PAGE_NAME_LEN = 8; // Max. Zeichen im Seitenamen
char buffer[MAX_PAGE_NAME_LEN+1]; // Seitenname + abschließende Null

EthernetServer server(80);
EthernetClient client;

void setup()
{
 Serial.begin(9600);
 Ethernet.begin(mac, ip);
 server.begin();
 Serial.println("Bereit");
}

void loop()
{
 client = server.available();
```

```
 if (client) {
  while (client.connected()) {
   if (client.available()) {
    if( client.find("GET ") ) {
     // Seitennamen suchen
     memset(buffer,0, sizeof(buffer)); // Puffer löschen
     if(client.find( "/"))
      if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
      {
       if(strcmp(buffer, "analog") == 0)
        showAnalog();
       else if(strcmp(buffer, "digital") == 0)
        showDigital();
       else
        unknownPage(buffer);
      }
    }
    Serial.println();
    break;
   }
  }
  // Web-Browser Zeit geben, die Seite zu empfangen
  delay(1);
  client.stop();
 }
}

void showAnalog()
{
 Serial.println("analog");
 sendHeader();
 client.println("<h1>Analoge Pins</h1>");
 // Werte aller analogen Eingangspins ausgeben

  for (int i = 0; i < 6; i++) {
  client.print("Analog-Pin ");
  client.print(i);
  client.print(" = ");
  client.print(analogRead(i));
  client.println("<br />");
 }
}

void showDigital()
{
 Serial.println("digital");
 sendHeader();
 client.println("<h1>Digitale Pins</h1>");
 // Werte der Digitalpins ausgeben
 for (int i = 2; i < 8; i++) {
  pinMode(i, INPUT);
  client.print("Digitalpin ");
  client.print(i);
  client.print(" ist ");
  if(digitalRead(i) == LOW)
   client.print("LOW");
  else
```

```
   client.print("HIGH");
  client.println("<br />");
 }
 client.println("</body></html>");
}

void unknownPage(char *page)
{
 sendHeader();
 client.println("<h1>Unbekannte Seite</h1>");
 client.print(page);
 client.println("<br />");
 client.println("Bekannte Seiten sind:<br />");
 client.println("/analog/<br />");
 client.println("/digital/<br />");
 client.println("</body></html>");
}

void sendHeader()
{
 // Standard HTTP-Response-Header senden
 client.println("HTTP/1.1 200 OK");
 client.println("Content-Type: text/html");
 client.println();
 client.println("<html><head><title>Webserver Multipage-Beispiel</title>");
 client.println("<body>");
}
```

Diskussion

Sie können das mit Ihrem Web-Browser ausprobieren, indem Sie *http://192.168.1.177/analog/* oder *http://192.168.1.177/digital/* eingeben (wenn Sie eine andere IP-Adresse für Ihren Webserver verwenden, müssen Sie den URL entsprechend anpassen).

Abbildung 15-2 zeigt die erwartete Ausgabe.

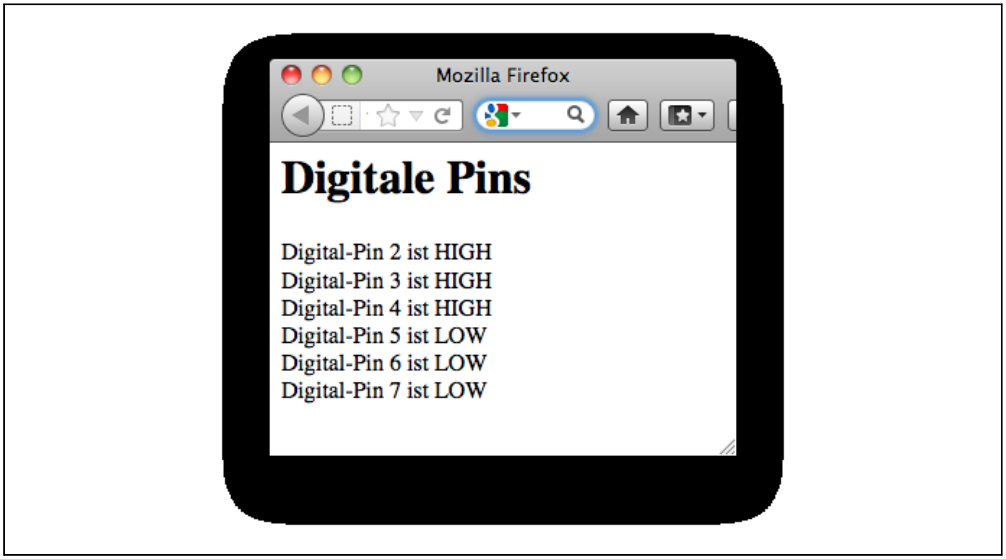


Abbildung 15-2: Ausgabe der Digitalpin-Werte im Browser

Der Sketch sucht nach einem »/«-Zeichen, um das Ende des Seitennamen zu bestimmen. Der Server meldet eine unbekannte Seite, wenn das »/«-Zeichen den Seitennamen nicht abschließt.

Sie können das ganz einfach um etwas Code aus Rezept 15.7 erweitern, der die Steuerung der Arduino-Pins über eine weitere Seite namens `update` erlaubt. Hier der neue `loop`-Code:

```
void loop()
{
 client = server.available();
 if (client) {
  while (client.connected()) {
   if (client.available()) {
    if( client.find("GET ") ) {
     // Seitennamen suchen
     memset(buffer,0, sizeof(buffer)); // Puffer löschen
      if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
      {
       if(strcmp(buffer, "analog") == 0)
        showAnalog();
       else if(strcmp(buffer, "digital") == 0)
        showDigital();
       // Zusätzlicher Code für neue update-Seite
       else if(strcmp(buffer, "update") == 0)
        doUpdate();
       else
        unknownPage(buffer);
      }
    }
    Serial.println();
```

```
      break;
     }
    }
    // Web-Browser Zeit geben, die Daten zu empfangen
    delay(1);
    client.stop();
   }
  }
```

Hier die doUpdate-Funktion:

```
void doUpdate()
{
 Serial.println("update");
 sendHeader();
 // Mit "pin" beginnende Tokens finden und bei der ersten Leerzeile aufhören
 while(client.findUntil("pin", "\n\r")){
  char type = client.read(); // D or A
  int pin = client.parseInt();
  int val = client.parseInt();
  if( type == 'D') {
   Serial.print("Digitalpin ");
   pinMode(pin, OUTPUT);
   digitalWrite(pin, val);

  }
  else if( type == 'A'){
   Serial.print("Analog-Pin ");
   analogWrite(pin, val);

  }
  else {
   Serial.print("Unbekannter Typ ");
   Serial.print(type);
  }
  Serial.print(pin);
  Serial.print("=");
  Serial.println(val);
 }
}
```

Wenn Sie *http://192.168.1.177/update/?pinA5=128* über Ihren Web-Browser senden, wird der Wert 128 an den Analog-Pin 5 geschrieben.

15.9 Antworten des Webservers mit HTML aufbereiten

Problem

Sie wollen HTML-Elemente wie Tabellen und Bilder nutzen, um das Aussehen der vom Arduino zurückgelieferten Webseiten zu verbessern. Zum Beispiel wollen Sie die Ausgabe aus Rezept 15.8 in einer HTML-Tabelle darstellen.

Lösung

Abbildung 15-3 zeigt, wie dieses Rezept die Ausgabe der Pin-Werte aufbereitet (formatiert). (Vergleichen Sie das mit den unformatierten Werten in Abbildung 15-2.)

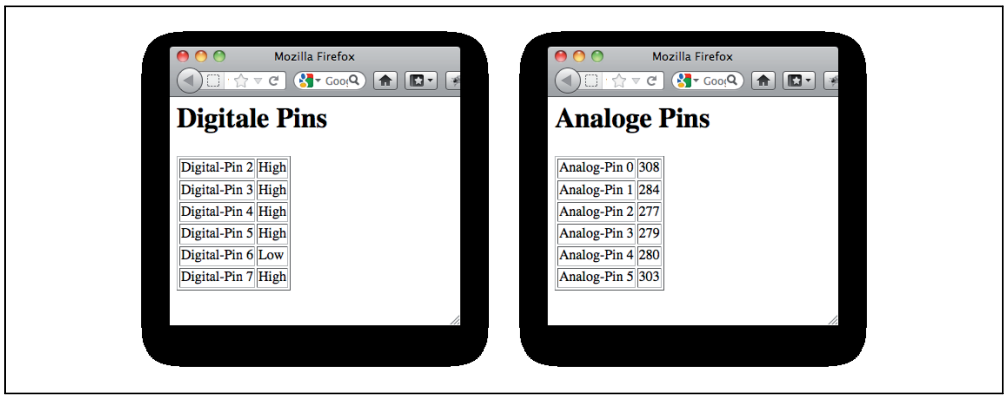


Abbildung 15-3: Seiten mit HTML-Formatierung

Der folgende Sketch bietet die gleiche Funktionalität wie Rezept 15.8 und formatiert die Ausgabe mit Hilfe von HTML:

```
/*
 * WebServerMultiPageHTML
 * Arduino 1.0 version
 * Analoge und digitale Pin-Werte mit HTML ausgeben
 */

#include <SPI.h>      // Seit Arduino-Version 0018 Pflicht
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] =  { 192,168,1,177 };

// Puffer muss groß genug sein, die angeforderten Seitennamen und die abschließende Null auf
// zunehmen
const int MAX_PAGE_NAME_LEN = 8+1;  // Max. Zeichen im Seitennamen + Null
char buffer[MAX_PAGE_NAME_LEN];

EthernetServer server(80);
EthernetClient client;

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

 Ethernet.begin(mac, ip);
 server.begin();
 pinMode(13,OUTPUT);
 for(int i=0; i < 3; i++)
 {
   digitalWrite(13,HIGH);
```

```
   delay(500);
   digitalWrite(13,LOW);
   delay(500);
 }
}

void loop()
{
 client = server.available();
 if (client) {
  while (client.connected()) {
   if (client.available()) {
    if( client.find("GET ") ) {
     // look for the page name
     memset(buffer,0, sizeof(buffer)); // Puffer löschen
     if(client.find( "/"))
      if(client.readBytesUntil('/', buffer, MAX_PAGE_NAME_LEN ))
      {
       if(strcasecmp(buffer, "analog") == 0)
        showAnalog();
       else if(strcasecmp(buffer, "digital") == 0)
        showDigital();
       else
        unknownPage(buffer);
      }
    }
    break;
   }
  }
  // Web-Browser Zeit geben, die Daten zu empfangen
  delay(1);
  client.stop();
 }
}

void showAnalog()
{
 sendHeader("Multipage: Analog");
 client.println("<h2>Analoge Pins</h2>");
 client.println("<table border='1' >");
 for (int i = 0; i < 6; i++) {
  // Wert aller analogen Eingangspins ausgeben
  client.print("<tr><td>Analogpin ");
  client.print(i);
  client.print(" </td><td>");
  client.print(analogRead(i));
  client.println("</td></tr>");
 }
 client.println("</table>");
 client.println("</body></html>");
}

void showDigital()
{
 sendHeader("Multi-page: Digital");
 client.println("<h2>Digitale Pins</h2>");
 client.println("<table border='1'>");
```

```
 for (int i = 2; i < 8; i++) {
  // Wert der Digitalpins ausgeben
  pinMode(i, INPUT);
  digitalWrite(i, HIGH); // Pullups einschalten
  client.print("<tr><td>Digitalpin ");
  client.print(i);
  client.print(" </td><td>");
  if(digitalRead(i) == LOW)
   client.print("Low");
  else
   client.print("High");
  client.println("</td></tr>");
 }
 client.println("</table>");
 client.println("</body></html>");
}

void unknownPage(char *page)
{
 sendHeader("Unbekannte Seite");
 client.println("<h1>Unbekannte Seite</h1>");
 client.print(page);
 client.println("<br />");
 client.println("Bekannte Seiten sind:<br />");
 client.println("/analog/<br />");
 client.println("/digital/<br />");
 client.println("</body></html>");
}

void sendHeader(char *title)
{
 // Standard HTTP-Response-Header senden
 client.println("HTTP/1.1 200 OK");
 client.println("Content-Type: text/html");
 client.println();
 client.print("<html><head><title>");
 client.println(title);
 client.println("</title><body>");
}
```

Diskussion

Die gleichen Informationen liefert auch Rezept 15.8, aber hier werden die Daten in einer HTML-Tabelle aufbereitet. Der folgende Code weist den Web-Browser an, eine Tabelle mit einer Rahmenbreite von 1 zu erzeugen:

```
client.println("<table border='1' >");
```

Die `for`-Schleife definiert die Datenzellen der Tabelle mit dem `<td>`- und die Zeilen mit dem `<tr>`-Tag. Der folgende Code platziert den String `"Analogpin "` in einer Zelle, die in einer neuen Zeile beginnt:

```
client.print("<tr><td>Analogpin ");
```

Dann wird der Wert der Variablen `i` ausgegeben:

```
client.print(i);
```

Die nächste Zeile enthält die Tags, die die Zelle schließen und eine neue Zelle beginnen:

```
client.print("</td><td>");
```

Nun wird der von `analogRead` zurückgelieferte Wert in die Zelle geschrieben:

```
client.print(analogRead(i));
```

Die Tags, die eine Zelle und eine Zeile abschließen, sehen wie folgt aus:

```
  client.println("</td></tr>");
```

Die `for`-Schleife wird durchlaufen, bis alle sechs Analogwerte ausgegeben wurden. Jedes in Rezept 14.3 erwähnte Buch und eine der vielen HTML-Referenz-Sites liefert weitere Details zu den HTML-Tags.

Siehe auch

Learning Web Design von Jennifer Niederst Robbins (O'Reilly)

Web Design in a Nutshell von Jennifer Niederst Robbins (O'Reilly)

HTML & XHTML: The Definitive Guide von Chuck Musciano und Bill Kennedy (O'Reilly)

(Suchen Sie nach O'Reilly-Titeln auf www.oreilly.de.)

15.10 Formulare (POST) verarbeiten

Problem

Sie wollen Formular-Webseiten entwickeln, die es dem Benutzer erlauben, eine Aktion auszuwählen, die vom Arduino ausgeführt wird. Abbildung 15-4 zeigt die Webseite, die in diesem Rezept erzeugt wird.

Abbildung 15-4: Web-Formular mit Buttons

Lösung

Der folgende Sketch erzeugt ein Formular mit Buttons. Der Benutzer kann die Buttons in seinem Browser anklicken und der Arduino-Webserver reagiert darauf. In diesem Beispiel schaltet der Sketch in Abhängigkeit vom gedrückten Button einen Pin an oder aus:

```
/*
 * WebServerPost Sketch
 * Schaltet Pin 8 über HTML-Formular an oder aus
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] =  { 192,168,1,177 };

const int MAX_PAGENAME_LEN = 8; // Max. Zeichen im Seitennamen
char buffer[MAX_PAGENAME_LEN+1]; // Zusätzliches Zeichen für abschließende Null

EthernetServer server(80);

void setup()
{
 Serial.begin(9600);
 Ethernet.begin(mac, ip);
 server.begin();
 delay(2000);
}

void loop()
{
 EthernetClient client = server.available();
 if (client) {
  int type = 0;
  while (client.connected()) {
      if (client.available()) {
    // GET, POST, or HEAD
```

```
    memset(buffer,0, sizeof(buffer)); // Puffer löschen
    if(client.find("/"))
     if(client.readBytesUntil('/', buffer,sizeof(buffer))){
      Serial.println(buffer);
      if(strcmp(buffer,"POST ") == 0){
       client.find("\n\r"); // Body überspringen
       // Mit "pin" beginnenden String finden, bei der ersten Leerzeile anhalten
       // POST-Parameter werden in der Form pinDx=Y erwartet
       // x ist dabei die Pin-Nummer und Y ist 0 für LOW und 1 für HIGH
       while(client.findUntil("pinD", "\n\r")){
        int pin = client.parseInt();    // Die Pin-Nummer
        int val = client.parseInt();    // 0 oder 1
        pinMode(pin, OUTPUT);
        digitalWrite(pin, val);
       }
      }
      sendHeader(client,"Post-Beispiel");
      //HTML-Button, um Pin 8 auszuschalten
      client.println("<h2>Buttons anklicken, um Pin 8 ein- oder auszuschalten</h2>");
      client.print(
      "<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
      client.println(" value='0'><input type='submit' value='Off'/></form>");
      //HTML-Button, um Pin 8 einzuschalten
      client.print(
      "<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
      client.print(" value='1'><input type='submit' value='On'/></form>");
      client.println("</body></html>");
      client.stop();
     }
    break;
   }
  }
  // Web-Browser Zeit geben, die Daten zu empfangen
  delay(1);
  client.stop();
 }
}
void sendHeader(EthernetClient client, char *title)
{
 // Standard HTTP-Response-Header senden
 client.println("HTTP/1.1 200 OK");
 client.println("Content-Type: text/html");
 client.println();
 client.print("<html><head><title>");
 client.print(title);
 client.println("</title><body>");
}
```

Diskussion

Eine Webseite mit einem Formular besteht aus HTML-Tags, die die Steuerelemente (Buttons, Checkboxen, Label etc.) festlegen, aus denen die Benutzerschnittstelle aufgebaut ist. Dieses Rezept nutzt Buttons zur Interaktion mit dem Benutzer.

Die folgenden Zeilen erzeugen einen Button namens pinD8 mit dem Text »AUS«, der beim Anklicken den Wert 0 (Null) sendet:

```
client.print("<form action='/' method='POST'><p><input type='hidden' name='pinD8'");
client.println(" value='0'><input type='submit' value='AUS'/></form>");
```

Empfängt der Server einen Request von einem Browser, sucht er nach dem String "POST", um den Anfang des gesendeten Formulars zu erkennen:

```
if (strcmp(buffer,"POST ") == 0) // Beginn des Formulars erkennen

  client.find("\n\r"); // Weiter zum Body
  // Mit "pin" beginnenden Parameter finden und Suche bei der erste Leerzeile beenden
  // Die POST-Parameter werden in der Form pinDx=Y erwartet
  // wobei x die Pin-Nummer ist. Y ist 0 für LOW und 1 für HIGH
```

Wird der AUS-Button gedrückt, enthält die empfangene Seite den String pinD8=0. Sie enthält pinD8=1, wenn der AN-Button gedrückt wird.

Der Sketch durchsucht die empfangenen Daten, bis er den Button-Namen (pinD) findet:

```
while(client.findUntil("pinD", "\n\r"))
```

Die Methode findUntil im obigen Code sucht nach »pinD« und beendet seine Suche am Zeilenende (\n\r ist die Kombination aus Newline und Carriage Return, die der Browser am Ende des Formulars sendet).

Die auf pinD folgende Zahl ist die Pin-Nummer:

```
int pin = client.parseInt();     // die Pin-Nummer
```

Der auf die Pin-Nummer folgende Wert ist 0, wenn der Button AUS bedrückt wurde, bzw. 1, wenn der EIN-Button gedrückt wurde:

```
int val = client.parseInt();     // 0 oder 1
```

Der empfangene Wert wird an den Pin geschrieben, nachdem dieser als Ausgang geschaltet wurde:

```
pinMode(pin, OUTPUT);
 digitalWrite(pin, val);
```

Weitere Buttons können eingefügt werden, indem man die entsprechenden Tags für weitere Steuerelemente aufnimmt. Die folgenden Zeilen fügen einen weiteren Button ein, der den Digitalpin 9 einschaltet:

```
//HTML-Button schaltet Pin 9 ein
client.print("<form action='/' method='POST'><p><input type='hidden' name='pinD9'");
client.print(" value='1'><input type='submit' value='EIN'/></form>");
```

15.11 Webseiten mit großen Datenmengen zurückgeben

Problem

Ihre Webseiten benötigen mehr Speicher, als Ihnen zur Verfügung steht, weshalb Sie den Programmspeicher (auch *Flash-Speicher* genannt) zur Speicherung von Daten nutzen wollen (siehe Rezept 16.4).

Lösung

Der folgende Sketch kombiniert den POST-Code aus Rezept 15.10 mit dem HTML-Code aus Rezept 15.9 und fügt zusätzlichen Code ein, der auf Text zugreift, der im Programmspeicher enthalten ist. Wie in Rezept 15.9 kann der Server den Status der analogen und digitalen Pins ausgeben und die Digitalpins ein- und ausschalten (siehe Abbildung 15-5).

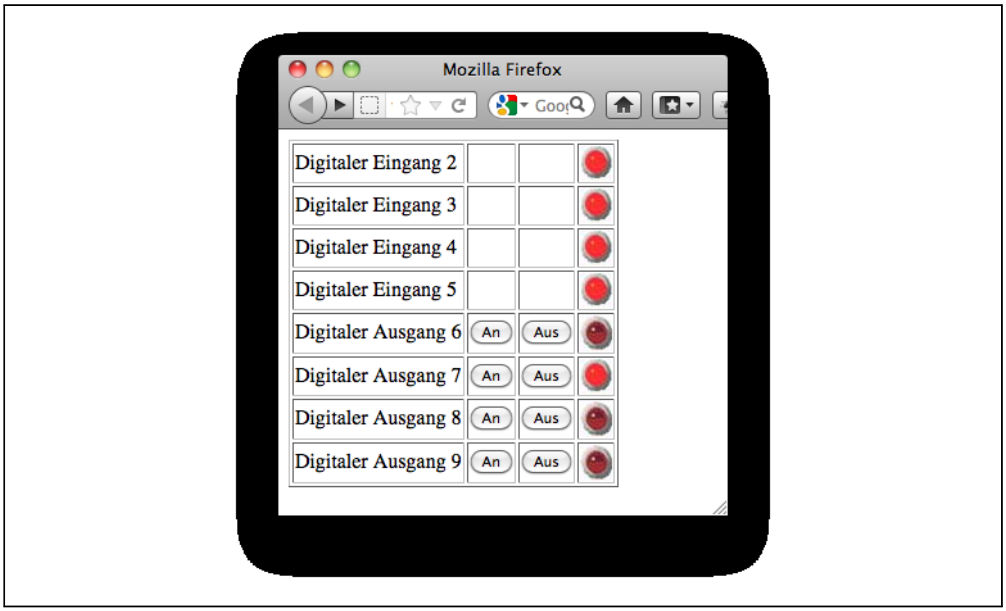


Abbildung 15-5: Webseite mit LED-Images

```
/*
 * WebServerMultiPageHTMLProgmem Sketch
 *
 * Verarbeitet Requests zur Änderung digitaler und analoger Ausgangs-Port
 * Gibt die Zahl der geänderten Ports und die Werte der analogen Eingangspins aus.
 *
 * http://192.168.1.177/analog/  Gibt analoge Pindaten aus
 * http://192.168.1.177/digital/ Gibt digitale Pindaten aus
 * http://192.168.1.177/change/  Ändert digitale Pindaten
 *
 */
```

```
#include <SPI.h>      // Seit Arduino-Version 0018 Pflicht
#include <Ethernet.h>

#include <avr/pgmspace.h> // Für Programmspeicher
#define P(name)  static const prog_uchar name[] PROGMEM // Statischen String deklarieren

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] =  { 192,168,1,177 };

const int MAX_PAGENAME_LEN = 8;  // Max. Zeichen im Seitenamen
char buffer[MAX_PAGENAME_LEN+1]; // Zusätzliches Zeichen für abschließende Null

EthernetServer server(80);
EthernetClient client;

void setup()
{
 Serial.begin(9600);
 Ethernet.begin(mac, ip);
 server.begin();
 delay(1000);
 Serial.println(F("Bereit"));
}

void loop()
{

 client = server.available();
 if (client) {
  int type = 0;
  while (client.connected()) {
   if (client.available()) {
    // GET, POST, or HEAD
    memset(buffer,0, sizeof(buffer)); // Puffer löschen
    if(client.readBytesUntil('/', buffer,MAX_PAGENAME_LEN)){
     if(strcmp(buffer, "GET ") == 0 )
      type = 1;
     else if(strcmp(buffer,"POST ") == 0)
      type = 2;
     // Seitennamen suchen
     memset(buffer,0, sizeof(buffer)); // Puffer löschen
     if(client.readBytesUntil( '/', buffer,MAX_PAGENAME_LEN ))
     {
      if(strcasecmp(buffer, "analog") == 0)
       showAnalog();
      else if(strcasecmp(buffer, "digital") == 0)
       showDigital();
      else if(strcmp(buffer, "change")== 0)
       showChange(type == 2);
      else
       unknownPage(buffer);
     }
    }
    break;
   }
  }
  // Web-Browser Zeit geben, die Daten zu empfangen
```

```
  delay(1);
  client.stop();
 }
}

void showAnalog()
{
 Serial.println(F("analog"));
 sendHeader("Multipage-Beispiel - Analog");
 client.println("<h1>Analoge Pins</h1>");
 // Werte der analogen Eingangspins ausgeben

  client.println(F("<table border='1' >"));
 for (int i = 0; i < 6; i++) {
  client.print(F("<tr><td>Analog-Pin "));
  client.print(i);
  client.print(F(" </td><td>"));
  client.print(analogRead(i));
  client.println(F("</td></tr>"));
 }
 client.println(F("</table>"));
 client.println(F("</body></html>"));
}

// MIME-kodierte Daten für Images ein- und ausgeschalteter LED:
// siehe: http://www.motobit.com/util/base64-decoder-encoder.asp
P(led_on) =  "<img src=\""
"/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAHgAA/+4ADkFkb2JlAGTAAAAAAf/b"
"AIQAEAsLCwwLEAwMEBcPDQ8XGxQQEBQbHxcXFxcXHx4XGhoaGhceHiMlJyUjHi8vMzMvLOBAQEBA"
"QEBAQEBAQEBAQAERDw8RExEVEhIVFBEUERQaFBYWFBomGhocGhomMCMeHh4eIzArLicnJy4rNTUw"
"MDU1QEA/QEBAQEBAQEBAQEBA/8AAEQgAGwAZAwEiAAIRAQMRAf/EAIIAAAICAwAAAAAAAAAAAAA"
"AAUGAAcCAwQBAAMBAAAAAAAAAAAAAAAAAAAACBAUQAAECBAQBCgcAAAAAAAAAAECAwARMRIhQQQF"
"UWFxkaHRMoITUwYiQnKSIxQ1EQAAAwYEBwAAAAAAAAAAAAAAAAAARECEgMTBBQhQWEiMVGBMkJiJP/a"
"AAwDAQACEQMRAD8AcNz3BGibKieOnhCOv3A+teKJt8JmZEdHuZalOitgUoHnEpQEWtSyLqgACWFI"
"nixWiaQhsUFFBiQSbiMvvrmeCBp27eLnG7lFTDxs+Kra8oOyium3ltJUAcDIy4EUMN/7Dnq9cPMO"
"W9OE9kxeyF2d3HFOQ175olKudUm7TqlfKqDQEDOFR1sNqtC7k5ERYjndNPFSArtvnI/nV+ed9coI"
"ktd2BgozrSZO3J5jVEXRcwD2bbXNdqOzT+BohTyjgPp5SYdPJZ9NP2jsiIz7vhjLohtjnqJ/ouPK"
"co//2Q=="
"\"/>";

P(led_off) = "<img src=\""
"/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAHgAA/+4ADkFkb2JlAGTAAAAAAf/b"
"AIQAEAsLCwwLEAwMEBcPDQ8XGxQQEBQbHxcXFxcXHx4XGhoaGhceHiMlJyUjHi8vMzMvLOBAQEBA"
"QEBAQEBAQEBAQAERDw8RExEVEhIVFBEUERQaFBYWFBomGhocGhomMCMeHh4eIzArLicnJy4rNTUw"
"MDU1QEA/QEBAQEBAQEBAQEBA/8AAEQgAHAAZAwEiAAIRAQMRAf/EAHgAAQEAAwAAAAAAAAAAAAA"
"AAYFAgQHAQEBAQAAAAAAAAAAAAAAAAAACAQQQAAECBQAHBQkAAAAAAAAAAECAwAREhMEITFhoSIF"
"FUFROUIGgZHBMlIjM1MWEQABAwQDAQEAAAAAAAAAAAABABECIWESA1ETIyIE/9oADAMBAAIRAxEA"
"PwBvl5SWEkkylpJMGsj1XjXSE1kCUf8IJZAQ6loNtBooTJaqp3DP5oBlV"
"nWrTpEouQS/Cf4POOuKbqWHGXTSlztSoTMu8aiNsXet5/17hFyo6PR64rA"
"ZnuqfqDDDySFpNpYH3E6aFjzGBr2DkMuFBSFDsWkilUdLftW13pWpcdWqnbBzI/l6hVXKZlROUSe"
"L1KX5zvAPXESjdHsTFWpxLKOJ54hIA1DZCj+Vx/3r96fCNrkvRaTO+V3zV/llplr9sVeHZui/ONk"
"H3dzt6cL/9k="
"\"/>";
;
```

```
void showDigital()
{
 Serial.println(F("digital"));
 sendHeader("Multipage-Beispiel - Digital");
 client.println(F("<h2>Digitale Pins</h2>"));
 // Werte der Digitalpins ausgeben
 client.println(F("<table border='1'>"));
 for (int i = 2; i < 8; i++) {
  pinMode(i, INPUT);
  digitalWrite(i, HIGH); // Pullups einschalten
  client.print(F("<tr><td>Digitalpin "));
  client.print(i);
  client.print(F(" </td><td>"));
  if(digitalRead(i) == LOW)
   printP(led_off);
  else
   printP(led_on);
  client.println(F("</td></tr>"));
 }
 client.println(F("</table>"));

 client.println(F("</body></html>"));
}


void showChange(boolean isPost)
{
 Serial.println(F("change"));
 if(isPost)
 {
  Serial.println("isPost");
  client.find("\n\r"); // Body überspringen
  // Nach mit "pin" beginnenden Parametern suchen und bei der ersten Leerzeile abbrechen
  Serial.println(F("Suche nach Parametern"));
  while(client.findUntil("pinD", "\n\r")){
   int pin = client.parseInt();    // Pin-Nummer
   int val = client.parseInt();    // 0 oder 1
   Serial.print(pin);
   Serial.print("=");
   Serial.println(val);
   pinMode(pin, OUTPUT);
   digitalWrite(pin, val);
  }
 }
 sendHeader("Multipage-Beispiel - Ändern");
 // Tabelle mit Buttons von 2 bis 9
 // 2 bis 5 sind Eingänge, die anderen Buttons sind Ausgänge
 client.println(F("<table border='1'>"));

 // Eingangspins ausgeben
 for (int i = 2; i < 6; i++) {  // Pins 2-5 sind Eingänge
  pinMode(i, INPUT);
  digitalWrite(i, HIGH); // Pullups einschalten
  client.print(F("<tr><td>Digitaler Eingang "));
  client.print(i);
  client.print(F(" </td><td>"));
```

```
  client.print(F("  </td><td>"));
  client.print(F(" </td><td>"));
  client.print(F("  </td><td>"));


  if(digitalRead(i) == LOW)
   //client.print("AUS");
   printP(led_off);
  else
   //client.print("EIN");
   printP(led_on);
  client.println("</td></tr>");
 }

 // Ausgangspins 6-9 ausgeben
 // Hinweis: Pins 10-13 werden vom Ethernet-Shield genutzt
 for (int i = 6; i < 10; i++) {
  client.print(F("<tr><td>Digitaler Ausgang "));
  client.print(i);
  client.print(F(" </td><td>"));
  htmlButton( "EIN", "pinD", i, "1");
  client.print(F(" </td><td>"));
  client.print(F(" </td><td>"));
  htmlButton("AUS", "pinD", i, "0");
  client.print(F(" </td><td>"));

  if(digitalRead(i) == LOW)
   //client.print("AUS");
   printP(led_off);
  else
   //client.print("EIN");
   printP(led_on);
  client.println(F("</td></tr>"));
 }
 client.println(F("</table>"));
}

// HTML-Button erzeugen
void htmlButton( char * label, char *name, int nameId, char *value)
{
 client.print(F("<form action='/change/' method='POST'><p><input type='hidden' name='"));
 client.print(name);
 client.print(nameId);
 client.print(F("' value='"));
 client.print(value);
 client.print(F("'><input type='submit' value='"));
 client.print(label);
 client.print(F("'/></form>"));
}

void unknownPage(char *page)
{
 Serial.print(F("Unbekannt : "));
 Serial.println(F("page"));

 sendHeader("Unbekannte Seite");
 client.println(F("<h1>Unbekannte Seite</h1>"));
```

```
 client.println(page);
 client.println(F("</body></html>"));
}

void sendHeader(char *title)
{
 // Standard HTTP-Response-Header senden
 client.println(F("HTTP/1.1 200 OK"));
 client.println(F("Content-Type: text/html"));
 client.println();
 client.print(F("<html><head><title>"));
 client.println(title);
 client.println(F("</title><body>"));
}

void printP(const prog_uchar *str)
{
 // Daten aus dem Programmspeicher in den lokalen Speicher kopieren
 // 32-Byte-Segmente schreiben, um sehr kurze TCP/IP-Pakete zu vermeiden
 // Aus der webduino-Bibliothek, Copyright 2009 Ben Combee, Ran Talbott
 uint8_t buffer[32];
 size_t bufferEnd = 0;

 while (buffer[bufferEnd++] = pgm_read_byte(str++))
 {
  if (bufferEnd == 32)
  {
   client.write(buffer, 32);
   bufferEnd = 0;
  }
 }

 // Rest rausschreiben, bis auf die abschließende NUL
 if (bufferEnd > 1)
  client.write(buffer, bufferEnd - 1);
}
```

Diskussion

Die Logik beim Aufbau der Webseite ähnelt der aus den vorangegangenen Rezepten. Das Formular basiert auf Rezept 15.10, enthält aber mehr Elemente in der Tabelle und verwendet eingebettete Grafik-Objekte für den Status der Pins. Wenn Sie schon einmal eine Webseite entwickelt haben, sind Sie wahrscheinlich mit der Verwendung von JPEG-Images innerhalb einer Seite vertraut. Die Arduino Ethernet-Bibliothek kann Images im *.jpg*-Format nicht verarbeiten.

Images müssen in einem Internet-Standard wie MIME (Multipurpose Internet Mail Extensions) kodiert sein. Auf diese Weise lassen sich grafische (und andere) Medien in Textform darstellen. Der Sketch zeigt, wie die LED-Images MIME-kodiert aussehen. Viele Webbasierte Dienste ermöglichen die MIME-Kodierung Ihrer Images. Die hier genutzten Images wurden mit dem Dienst von *http://www.motobit.com/util/base64-decoder-encoder.asp* umgewandelt.

Selbst die in diesem Beispiel verwendeten kleinen LED-Images sind zu groß, um in das Arduino-RAM zu passen. Daher nutzen wir Programmspeicher (Flash). In Rezept 16.3 finden Sie eine Erläuterung des `P(name)`-Ausdrucks.

Die Images der ein- und ausgeschalteten LEDs werden als eine Folge von Zeichen gespeichert. Das Array für die eingeschaltete LED beginnt wie folgt:

```
P(led_on) = "<img src=\""
```

`P(led_on) =` definiert `led_on` als Namen dieses Arrays. Die Zeichen sind die HTML-Tags für ein Image gefolgt von den MIME-kodieren Daten, aus denen das Image besteht.

Dieses Beispiel basiert auf Code, der für den Webduino-Webserver geschrieben wurde. Webduino wird zum Aufbau von Webseiten wärmstens empfohlen, wenn Ihre Anwendung komplizierter ist, als die hier vorgestellten Beispiele.

Siehe auch

In Rezept 16.4 erfahren Sie mehr über das `F("text")`-Konstrukt zur Speicherung von Text im Flash-Speicher.

Webduino-Webseite: *http://code.google.com/p/webduino/*

15.12 Twitter-Nachrichten senden

Problem

Ihr Arduino soll Nachrichten an Twitter senden, z.B. wenn ein Sensor eine Aktivität erkennt, die Sie per Twitter überwachen wollen.

Lösung

Der folgende Sketch sendet eine Twitter-Nachricht, wenn ein Schalter geschlossen wird. Er nutzt einen Proxy auf *http://www.thingspeak.com* für die Autorisierung, d.h., Sie müssen sich auf dieser Site registrieren, um einen (kostenlosen) API-Schlüssel zu erhalten. Klicken Sie den Sign-Up-Button auf der Homepage an und füllen Sie das Formular aus (die gewünschte Benutzer-ID, E-Mail, Zeitzone und das Passwort). Wenn Sie den Create-Account-Button anklicken, erhalten Sie einen ThingSpeak API-Schlüssel. Um den ThingSpeak-Dienst nutzen zu können, müssen Sie Ihren Twitter-Account autorisieren, damit ThingTweet Nachrichten an Ihren Account posten kann. Nachdem Sie das eingerichtet haben, ersetzen Sie `"YourThingTweetAPIKey"` durch Ihren Schlüssel und führen den folgende Sketch aus:

```
/*
 * Sende Tweet, wenn Taster an Pin 2 gedrückt wird
 * Nutzt api.thingspeak.com als Twitter-Proxy
 * Siehe: http://community.thingspeak.com/documentation/apps/thingtweet/
 */
```

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte server[] = { 184, 106, 153, 149 }; // IP-Adresse für ThingSpeak-API

char *thingtweetAPIKey = "YourThingTweetAPIKey"; // Durch Ihren ThingTweet API-Schlüssel
                                                 // ersetzen

EthernetClient client;

boolean MsgSent = false;
const int Sensor = 2;

void setup()
{
 Serial.begin(9600);
 if (Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration über DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
 pinMode(Sensor, INPUT);
 digitalWrite(Sensor, HIGH); // Pullup-Widerstände einschalten
 delay(1000);
 Serial.println("Bereit");
}

void loop()
{
 if(digitalRead(Sensor) == LOW)
 { // Briefkasten geöffnet

  if(MsgSent == false){ // Nachricht gesendet?
   MsgSent = sendMessage("Post ist da");
   if(MsgSent)
     Serial.println("Tweet erfolgreich");
   else
     Serial.println("Tweet fehlgeschlagen");
  }
 }
 else{
   MsgSent = false; // Briefkasten geschlossen, Status zurücksetzen
 }
 delay(100);
}

boolean sendMessage( char *message)
{
boolean result = false;

 const int tagLen = 16; // Anzahl Tag-Zeichen zum Framing der Nachricht
 int msgLen = strlen(message) + tagLen + strlen(thingtweetAPIKey);
 Serial.println("Verbinde ...");
```

```
 if (client.connect(server, 80) ) {
  Serial.println("Fuehre POST-Request aus...");
  client.print("POST /apps/thingtweet/1/statuses/update HTTP/1.1\r\n");
  client.print("Host: api.thingspeak.com\r\n");
  client.print("Connection: close\r\n");
  client.print("Content-Type: application/x-www-form-urlencoded\r\n");
  client.print("Content-Length: ");
  client.print(msgLen);
  client.print("\r\n\r\n");
  client.print("api_key=");      // msg-Tag
  client.print(thingtweetAPIKey);  // api-Schlüssel
  client.print("&status=");      // msg-Tag
  client.print(message);       // Die Nachricht
  client.println("\r\n");
 }
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 // Response-String
 if (client.connected()) {
  Serial.println("Verbunden");
  if(client.find("HTTP/1.1") && client.find("200 OK") ){
    result = true;
  }
  else
    Serial.println("Trenne Verbindung - kein 200 OK");
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();

 return result;
}
```

Diskussion

Der Sketch wartet, bis ein Pin auf `LOW` geht, und sendet dann über die ThingTweet-API eine Nachricht an Twitter.

Das Web-Interface übernimmt die Funktion `sendMessage();`, die die übergebene Nachricht sendet. Bei diesem Sketch versucht sie, die Nachricht »Post ist da« an Twitter zu senden und gibt `true` zurück, wenn die Verbindung hergestellt werden kann.

Weitere Details finden Sie in der Dokumentation auf der ThingTweet-Website: *http://community.thingspeak.com/documentation/apps/thingtweet/*

Die folgende Version nutzt die gleiche `sendMessage`-Funktion, kann aber mehrere Sensoren überwachen:

```
/*
 * Sende über mehrere Sensoren ausgelösten Tweet
 * Nutzt api.thingspeak.com als Twitter-Proxy
```

```
 * Siehe: http://community.thingspeak.com/documentation/apps/thingtweet/
 */

#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte server[] = { 184, 106, 153, 149 }; // IP-Adresse für ThingSpeak-API

char *thingtweetAPIKey = "YourThingTweetAPIKey"; // Ihr ThingTweet API-Schlüssel

EthernetClient client;
boolean MsgSent = false;

char frontOpen[] = "Die Vordertuer wurde geoeffnet";
char backOpen[] = "Die Hintertuer wurde geoeffnet";

const int frontSensor = 2; // Sensor-Pins
const int backSensor = 3;

boolean frontMsgSent = false;
boolean backMsgSent = false;

void setup()
{
// Ethernet.begin(mac,ip);
 Serial.begin(9600);
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true) // Weitermachen zwecklos, in Endlosschleife warten
    ;
 }
 pinMode(frontSensor, INPUT);
 pinMode(backSensor, INPUT);
 digitalWrite(frontSensor, HIGH); // pull-ups
 digitalWrite(backSensor, HIGH);
 delay(1000);
 Serial.println("Bereit");
}

void loop()
{
 if(digitalRead(frontSensor) == LOW)
 { // Vordertür offen
   if (frontMsgSent == false) { // Nachricht gesendet?
     frontMsgSent = sendMessage(frontOpen);
   }
 }
 else{
   frontMsgSent = false; // Tür geschlossen, Status zurücksetzen
 }
 if(digitalRead(backSensor) == LOW)
 {
  if(frontMsgSent == false) {
    backMsgSent = sendMessage(backOpen);
  }
 }
```

```
 else {
   backMsgSent = false;
 }
 delay(100);
}

// sendMesage-Funktion aus obigem Sketch einfügen
```

Der zur Kommunikation mit Twitter verwendete Code ist identisch, doch die gesendeten Nachrichten werden aus den Sensorwerten konstruiert, die über zwei Arduino-Digitalpins eingelesen werden.

Siehe auch

Ein ThingSpeak Arduino-Tutorial finden Sie auf *http://community.thingspeak.com/tutorials/arduino/using-an-arduino-ethernet-shield-to-update-a-thingspeak-channel/*

15.13 Einfache Nachrichten (UDP) senden und empfangen

Problem

Sie wollen einfache Nachrichten über das Internet senden und empfangen.

Lösung

Der folgende Sketch nutzt die Arduino UDP-Bibliothek (User Datagram Protocol) zum Senden und Empfangen von Strings. In diesem einfachen Beispiel gibt der Arduino den empfangenen String über den seriellen Monitor aus und schickt dem Sender den String »ACK« zurück:

```
/*
 * UDPSendReceiveStrings
 * Empfängt String in UDP-Nachrichten, gibt sie über den seriellen Port aus
 * und schickt den String "ACK" an den Sender zurück
 * Benötigt Arduino 1.0
 *
 */

#include <SPI.h>      // Für Arduino-Versionen ab 0018 Pflicht
#include <Ethernet.h>
#include <EthernetUdp.h> // Arduino-1.0-UDP-Bibliothek

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC-Adresse
byte ip[] = {192, 168, 1, 177 };   // IP-Adresse des Arduino

unsigned int localPort = 8888;    // Lokaler Port

// Puffer zum Empfangen und Senden von Daten
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //Puffer für eingehende Pakete
char replyBuffer[] = "ACK";     // Zurückgesendeter String
```

```
// UDP-Instanz zum Senden und Empfangen von Paketen per UDP
EthernetUDP Udp;

void setup() {
  // Ethernet und UDP starten
 Ethernet.begin(mac,ip);
 Udp.begin(localPort);
 Serial.begin(9600);
}

void loop() {
 // Wenn Daten vorhanden sind, Paket einlesen
 int packetSize = Udp.parsePacket();
 if(packetSize)
 {
  Serial.print("Paket empfangen... Groesse: ");
  Serial.println(packetSize);

  // Paket in packetBuffer einlessen und IP-Adresse und Port-Nummer des Senders ermitteln
  Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);
  Serial.println("Inhalt:");
  Serial.println(packetBuffer);

  // String an Sender zurückschicken
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
  Udp.write(replyBuffer);
  Udp.endPacket();
 }
 delay(10);
}
```

Sie können das testen, indem Sie den folgenden Processing-Sketch auf Ihrem Computer ausführen (in Kapitel 4 finden Sie Hinweise zur Installation und Betrieb von Processing):

```
// Processing-UDP-Beispiel zum Senden und Empfangen von String-Daten vom Arduino
// Drücken Sie eine beliebige Taste, um die Nachricht "Hallo, Arduino" zu senden

import hypermedia.net.*;

UDP udp; // UDP-Objekt definieren

void setup() {
 udp = new UDP( this, 6000 ); // Datagramm-Verbindung an Port 6000 herstellen
 //udp.log( true );       // <-- Verbindungs-Aktivität ausgeben
 udp.listen( true );       // und auf eingehende Nachrichten warten
}

void draw()
{
}

void keyPressed() {
 String ip     = "192.168.1.177";   // Entfernte IP-Adresse
 int port      = 8888;    // Ziel-Port

 udp.send("Hallo, Arduino", ip, port );   // Die zu sendende Nachricht
}
```

```
void receive( byte[] data ) {       // <-- Standard-Handler
//void receive( byte[] data, String ip, int port ) { // Erweiterter Handler

 for(int i=0; i < data.length; i++)
   print(char(data[i]));
 println();
}
```

Diskussion

Stecken Sie das Ethernet-Shield auf den Arduino und verbinden Sie das Ethernet-Kabel mit dem Computer. Laden Sie den Sketch auf den Arduino hoch und führen Sie den Processing-Sketch auf dem Computer aus. Drücken Sie eine beliebige Taste, um die Nachricht »Hallo, Arduino« zu senden. Arduino sendet »ACK« zurück, was im Processing-Textfenster erscheint. Die Länge des Strings ist durch eine Konstante in *EthernetUdp.h* beschränkt. Voreingestellt sind 24 Bytes, was Sie aber erhöhen können, indem Sie die folgende Zeile in *Udp.h* entsprechend anpassen:

```
#define UDP_TX_PACKET_MAX_SIZE 24
```

UDP ist eine einfache und schnelle Möglichkeit, Nachrichten über Ethernet zu senden und zu empfangen. Doch es gibt Einschränkungen: Die Zustellung der Nachrichten wird nicht garantiert und bei einem stark ausgelasteten Netzwerk können Nachrichten verloren- oder in der falschen Reihenfolge eingehen. Aber UDP ist gut geeignet, wenn es um solche Dinge wie die Status-Ausgabe von Arduino-Sensoren geht – jede Nachricht enthält den aktuellen Sensorwert und verlorene Nachrichten werden durch nachfolgende Nachrichten ersetzt.

Der folgende Sketch demonstriert das Senden und Empfangen von Sensor-Nachrichten. Er empfängt Nachrichten mit Werten, die an die analogen Ausgangsports geschrieben werden sollen, und antwortet dem Sender mit den Werten der analogen Eingangspins:

```
/*
 * UDPSendReceive Sketch:
 */

#include <SPI.h>      // Ab Arduino-Version 0018 Pflicht
#include <Ethernet.h>
#include <EthernetUDP.h>  // Arduino 1.0 UDP-Bibliothek

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC-Adresse
byte ip[] = {192, 168, 1, 177 };   // IP-Adresse des Arduino

unsigned int localPort = 8888;    // Lokaler Port

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //Puffer für eingehende Pakete
int packetSize; // Größe empfangener Pakete

const int analogOutPins[] = { 3,5,6,9};  // Ethernet-Shield nutzt Pins 10 und 11

// UDP-Instanz zum Senden und Empfangen von UDP-Paketen
EthernetUDP Udp;
```

```
void setup() {
 Ethernet.begin(mac,ip);
 Udp.begin(localPort);

 Serial.begin(9600);
 Serial.println("Bereit");
}

void loop() {
 // Wenn Daten vorhanden, Paket einlesen
 packetSize = Udp.parsePacket();
 if(packetSize > 0 )
 {
  Serial.print("Paket empfangen... Groesse: ");
  Serial.print(packetSize);
  Serial.println("Inhalt:");
  // Paket in packetBuffer schreiben und IP-Adresse und Portnummer des Senders ermitteln
  packetSize = min(packetSize,UDP_TX_PACKET_MAX_SIZE);
  Udp.read(packetBuffer,UDP_TX_PACKET_MAX_SIZE);

  for( int i=0; i < packetSize; i++)
  {
    byte value = packetBuffer[i];
    if( i < 4)
    {
      // Nur an die ersten vier analogen Ausgangspins schreiben
      analogWrite(analogOutPins[i], value);
    }
    Serial.println(value, DEC);
  }
  Serial.println();
  // Sender die Werte unserer analogen Ports mitteilen
  sendAnalogValues(Udp.remoteIP(), Udp.remotePort());
 }
 //kurz warten
 delay(10);
}

void sendAnalogValues( IPAddress targetIp, unsigned int targetPort )
{
 int index = 0;
 for(int i=0; i < 6; i++)
 {
  int value = analogRead(i);

  packetBuffer[index++] = lowByte(value);  // Niederwertiges Byte);
  packetBuffer[index++] = highByte(value); // Höherwertiges Byte);  }
 }
 //Paket an Sender zurückschicken
 Udp.beginPacket(targetIp, targetPort);
 Udp.write(packetBuffer);
 Udp.endPacket();
}
```

Der Sketch sendet und empfängt die Werte der Analogports 0 bis 5 als Binärdaten. Wenn Sie nicht mit Binärdaten enthaltenden Nachrichten vertraut sind, sehen Sie sich die Einführung zu Kapitel 4 sowie Rezepte 4.6 und 4.7 an.

Der Unterschied besteht darin, dass die Daten mit `Udp.write` anstelle von `Serial.write` gesendet werden.

Hier ein Processing-Sketch, den Sie mit dem obigen Sketch nutzen können. Es nutzt sechs Rollbalken, die Sie mit der Maus bewegen können, um die sechs `analogWrite`-Werte festzulegen. Er gibt die empfangenen Sensordaten im Processing-Textfenster aus:

```
// Processing UDPTest
// Demo Sketch senden + empfangen von Arduino-Daten durch Gebrauch von UDP

import hypermedia.net.*;

UDP udp;  // Das UDP-Objekt definieren

HScrollbar[] scroll = new HScrollbar[6]; //Siehe: topics/gui/scrollbar

void setup() {
 size(256, 200);
 noStroke();
 for(int i=0; i < 6; i++) // Rollbalken ausgeben
  scroll[i] = new HScrollbar(0, 10 + (height / 6) * i, width, 10, 3*5+1);

 udp = new UDP( this, 6000 );  // Datagramm-Verbindung anPort 6000 aufbauen
 //udp.log( true );       // Verbindungsaktivität ausgeben
 udp.listen( true );       // und auf eingehende Verbindung warten
}

void draw()
{
 background(255);
 fill(255);
 for(int i=0; i < 6; i++) {
  scroll[i].update();
  scroll[i].display();
 }
}

void keyPressed() {
 String ip     = "192.168.1.177";   // Entfernte IP-Adresse
 int port      = 8888;         // Ziel-Port
 byte[] message = new byte[6] ;

 for (int i=0; i < 6; i++){
  message[i] = byte(scroll[i].getPos());
  println(int(message[i]));
 }
 println();
 udp.send( message, ip, port );

}
```

```
void receive( byte[] data ) {                 // <-- Standard-Handler
//void receive( byte[] data, String ip, int port ) {   // <-- Erweiterter Handler

 println("eingehende Daten:");
 for(int i=0; i < 6; i++){
  scroll[i].setPos(data[i]);
  println((int)data[i]);
 }
}

class HScrollbar
{
 int swidth, sheight;   // Breite und Höhe des Balkens
 int xpos, ypos;      // x- und y-Position des Balkens
 float spos, newspos;   // x-Position des Sliders
 int sposMin, sposMax;  // max- und min-Wert des Sliders
 int loose;         // wie locker/fest
 boolean over;       // Maus über Slider?
 boolean locked;
 float ratio;

 HScrollbar (int xp, int yp, int sw, int sh, int l) {
  swidth = sw;
  sheight = sh;
  int widthtoheight = sw - sh;
  ratio = (float)sw / (float)widthtoheight;
  xpos = xp;
  ypos = yp-sheight/2;
  spos = xpos + swidth/2 - sheight/2;
  newspos = spos;
  sposMin = xpos;
  sposMax = xpos + swidth - sheight;
  loose = l;
 }

 void update() {
  if (over()) {
   over = true;
  } else {
   over = false;
  }
  if (mousePressed && over) {
   locked = true;
  }
  if (!mousePressed) {
   locked = false;
  }
  if (locked) {
   newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
  }
  if(abs(newspos - spos) > 1) {
   spos = spos + (newspos-spos)/loose;
  }
 }
```

```
int constrain(int val, int minv, int maxv) {
 return min(max(val, minv), maxv);
}

boolean over() {
 if (mouseX > xpos && mouseX < xpos+swidth &&
 mouseY > ypos && mouseY < ypos+sheight) {
  return true;
 } else {
  return false;
 }
}

void display() {
 fill(255);
 rect(xpos, ypos, swidth, sheight);
 if (over || locked) {
  fill(153, 102, 0);
 } else {
  fill(102, 102, 102);
 }
 rect(spos, ypos, sheight, sheight);
}

float getPos() {
 return spos * ratio;
}

void setPos(int value) {
 spos = value / ratio;
}
}
```

15.14 Die Zeit von einem Internet-Zeitserver abrufen

Problem

Sie wollen die aktuelle Zeit von einem Internet-Zeitserver abrufen, z.B. um die auf dem Arduino laufende Software-Uhr zu synchronisieren.

Lösung

Der folgende Sketch ruft die Zeit von einem NTP-Server (Network Time Protocol) ab und gibt das Ergebnis in Sekunden seit dem 1.1.1900 (NTP-Zeit) und in Sekunden seit dem 1.1.1970 aus:

```
/*
 * UdpNtp Sketch
 * Zeit von einem NTP-Zeitserver abrufen
 * Demonstriert UDP-sendPacket und -ReceivePacket
 */

#include <SPI.h>
```

```
#include <Ethernet.h>

#include <EthernetUDP.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // MAC-Adresse

unsigned int localPort = 8888;    // Lokaler Port

IPAddress timeServer(192, 43, 244, 18); // time.nist.gov NTP-Server
const int NTP_PACKET_SIZE= 48; // NTP-Zeitstempel steht in den ersten 48
                  // Bytes der NAchricht
byte packetBuffer[ NTP_PACKET_SIZE]; // Puffer für ein- und ausgehende Pakete

// UDP-Instanz zum Senden und Empfangen von UDP-Paketen
EthernetUDP Udp;

void setup()
{
 Serial.begin(9600);
 // start Ethernet and UDP
 if (Ethernet.begin(mac) == 0) {
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  // Weitermachen zwecklos, in Endlosschleife warten
  for(;;)
   ;
 }
 Udp.begin(localPort);
}

void loop()
{
 sendNTPpacket(timeServer); // NTP-Paket an Zeitserver senden
 // Antwort verfügbar?
 delay(1000);
 if ( Udp.parsePacket() ) {
  Udp.read(packetBuffer,NTP_PACKET_SIZE);  // Paket in Puffer einlesen

   // Timestamp beginnt an Byte 40, vier Bytes in long-Wert umwandeln
   unsigned long hi = word(packetBuffer[40], packetBuffer[41]);
   unsigned long low = word(packetBuffer[42], packetBuffer[43]);
   unsigned long secsSince1900 = hi << 16 | low;  // NTP-Zeit
                              // (Sekunden seit dem 1.1.1900)

  Serial.print("Sekunden seit dem 1.1.1900 = " );
  Serial.println(secsSince1900);

  Serial.print("Unix-Zeit = ");
  // Unix-Zei beginnt am 1.1.1970
  const unsigned long seventyYears = 2208988800UL;
  unsigned long epoch = secsSince1900 - seventyYears;  // 70 Jahre abziehen
  Serial.println(epoch);                    // Unix-Zeit ausgeben


  // Stunde, Minute und Sekunde ausgeben:
  // UTC ist die Zeit am Greenwich-Meridian (GMT)
  Serial.print("Die UTC-Zeit ist ");
  // Stunde ausgeben (86400 sind Sek./Tag)
```

```
  Serial.print((epoch  % 86400L) / 3600);
  Serial.print(':');
  if ( ((epoch % 3600) / 60) < 10 ) {
   // Führende Null für die ersten zehn Minuten
   Serial.print('0');
  }
  // Minute ausgeben (3600 Sek sind 1 Std.)
  Serial.print((epoch  % 3600) / 60);
  Serial.print(':');
  if ( (epoch % 60) < 10 ) {
   // Führende Null für die erste zehn Sekunden
   Serial.print('0');
  }
  Serial.println(epoch %60); // Sekunden ausgeben
 }
 // Zehn Sekunden bis zur nächsten Abfrage warten
 delay(10000);
}

// NTP-Request an den Zeitserver mit der angegebenen Adresse senden
unsigned long sendNTPpacket(IPAddress& address)
{
 memset(packetBuffer, 0, NTP_PACKET_SIZE);  // Alle Bytes im Puffer auf 0 setzen

 // Für NTP-Request benötigte Werte setzen
 packetBuffer[0] = B11100011;  // LI, Version, Mode
 packetBuffer[1] = 0;   // Stratum
 packetBuffer[2] = 6;   // Max. Intervall zwischen Nachrichten in Sekunden
 packetBuffer[3] = 0xEC;  // Genauigkeit der Uhr
 // Bytes 4 - 11 sind Root Delay und wurden von memset auf 0 gesetzt
 packetBuffer[12]  = 49;  // Referenz-ID, vier Bytes
 packetBuffer[13]  = 0x4E;
 packetBuffer[14]  = 49;
 packetBuffer[15]  = 52;

 // Alle NTP-Felder enthalten vorgegebene Werte, nun
 // kann das anfordernde Paket gesendet werden
 Udp.beginPacket(address, 123); //NTP-Requests gehen an Port 123
 Udp.write(packetBuffer,NTP_PACKET_SIZE);
 Udp.endPacket();
}
```

Diskussion

NTP ist ein Protokoll zur Zeitsynchronisation über Internet-Nachrichten. NTP-Server geben die Zeit als Wert der seit dem 1.1.1900 verstrichenen Sekunden zurück. NTP gibt die Zeit als koordinierte Weltzeit (UTC, Coordinated Universal Time, entspricht der Greenwich Mean Time) zurück und berücksichtigt weder Zeitzonen noch die Sommerzeit.

NTP-Server verwenden UDP-Nachrichten. Eine Einführung in UDP finden Sie in Rezept 15.13. Die NTP-Nachricht wird in der Funktion `sendNTPpacket` erzeugt und Sie müssen den Code dieser Funktion sehr wahrscheinlich nicht ändern. Die Funktion erwartet die Adresse eines NTP-Servers. Sie können die Adresse aus dem obigen Beispiel

verwenden, oder Sie suchen sich einen der vielen Server aus, die eine Suche nach »NTP Adresse« ausspuckt. Wenn Sie mehr über die einzelnen NTP-Felder erfahren wollen, sehen Sie sich die Dokumentation auf *http://www.ntp.org/* an.

Die Antwort von NTP ist eine Nachricht in einem festen Format. Die Zeitinformation besteht aus vier Bytes, die am 40. Byte beginnen. Diese vier Bytes bilden den 32-Bit-Wert (ein Integerwert vom Typ unsigned long), mit der seit dem 1.1.1900 verstrichenen Zeit in Sekunden. Dieser Wert (und die in Unix-Zeit umgewandelte Zeit) werden ausgegeben. Soll die Zeit vom NTP-Server in ein freundlicheres Format mit Stunden, Minuten, Sekunden sowie Tag, Monat, Jahr umgewandelt werden, können Sie die Arduino Time-Bibliothek nutzen (siehe Kapitel 12). Hier eine Variante des obigen Codes, die die Zeit im Format `14:32:56 Monday 18 Jan 2010` ausgibt:

```
/*
 * Time_NTP Sketch
 * Zeisynchronisation per NTP-Zeitserver
 * Der Sketch nutzt die Time- und die
 * Arduino Ethernet-Bibliothek
 */

#include <Time.h>
#include <SPI.h>      // Seit Arduino-Version 0018 Pflicht
#include <Ethernet.h>
#include <EthernetUDP.h>

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 1, 44 }; // Gültige IP-Adresse (oder DHCP) verwenden

unsigned int localPort = 8888;    // Lokaler Port

IPAddress timeServer(192, 43, 244, 18); // time.nist.gov NTP-Server

const int NTP_PACKET_SIZE= 48; // NTP.Zeitstempel in den ersten 48 Bytes der Nachricht
byte packetBuffer[ NTP_PACKET_SIZE]; // Puffer für ein- und ausgehende Pakete

time_t prevDisplay = 0;   // Wann wurde die Zeit zuletzt ausgegeben

// UDP-Instanz zum Senden und Empfangen von UDP-Paketen
EthernetUDP Udp;

void setup()
{
 Serial.begin(9600);
 Ethernet.begin(mac,ip);
 Udp.begin(localPort);
 Serial.println("Warte auf Synchronisation");
 setSyncProvider(getNtpTime);
 while(timeStatus()== timeNotSet)
   ; // Warten, bis die Zeit vom Sync-Provider gesetzt wurde
}
```

```
void loop()
{
 if( now() != prevDisplay)  // Ausgabe nur aktualisieren, wenn sich die Zeit geändert hat
 {
  prevDisplay = now();
  digitalClockDisplay();
 }
}

void digitalClockDisplay(){
 // Digitalanzeige der Uhrzeit
 Serial.print(hour());
 printDigits(minute());
 printDigits(second());
 Serial.print(" ");
 Serial.print(dayStr(weekday()));
 Serial.print(" ");
 Serial.print(day());
 Serial.print(" ");
 Serial.print(monthShortStr(month()));
 Serial.print(" ");
 Serial.print(year());
 Serial.println();
}

void printDigits(int digits){
 // Hilfsfunktion für Digitaluhr: Gibt
 // vorstehenden Doppelpunkt und führende 0 aus
 Serial.print(":");
 if(digits < 10)
  Serial.print('0');
 Serial.print(digits);
}

/*-------- NTP-Code ----------*/

unsigned long getNtpTime()
{
 sendNTPpacket(timeServer); // Sendet NTP-Paket an Zeitserver
 delay(1000);
 if ( Udp.parsePacket() ) {
   Udp.read(packetBuffer,NTP_PACKET_SIZE);  // Paket in Puffer einlesen

   //Zeitstempel beginnt an Byte 40, vier Bytes in long long integer umwandeln
   unsigned long hi = word(packetBuffer[40], packetBuffer[41]);
   unsigned long low = word(packetBuffer[42], packetBuffer[43]);
   // NTP-Zeit (Sekunden seit dem 1.1.1900)
   unsigned long secsSince1900 = hi << 16 | low;
   // Unix-Zeit beginnt am 1.1.1970
   const unsigned long seventyYears = 2208988800UL;
   unsigned long epoch = secsSince1900 - seventyYears;  // 70 Jahre abziehen
   return epoch;
 }
 return 0; // Bei Fehler 0 zurückgeben
}
```

```
// NTP-Request an Zeitserver an übergebene Adresse senden
unsigned long sendNTPpacket(IPAddress address)
{
 memset(packetBuffer, 0, NTP_PACKET_SIZE); // Alle Bytes im Puffer auf 0 setzen

 // Werte für NTP-Request initialisieren
 packetBuffer[0] = B11100011;  // LI, Version, Mode
 packetBuffer[1] = 0;    // Stratum
 packetBuffer[2] = 6;    // Max. Intervall zwischen Nachrichten in Sekunden
 packetBuffer[3] = 0xEC; // Genauigkeit
 // Bytes 4 - 11 sind für Root Delay und wurden von memset auf 0 gesetzt
 packetBuffer[12] = 49; // 4-Byte-Referenz-ID
 packetBuffer[13] = 0x4E;
 packetBuffer[14] = 49;
 packetBuffer[15] = 52;

 // Zeitstempel anforderndes Paket senden
 Udp.beginPacket(address, 123); //NTP-Requests gehen an Port 123
 Udp.write(packetBuffer,NTP_PACKET_SIZE);
 Udp.endPacket();
}
```

Siehe auch

Kapitel 12 für weitere Informationen zur Arduino Time-Bibliothek.

Details zu NTP finden Sie auf *http://www.ntp.org/*.

NTP-Code von Jesse Jaggars (der den Sketch in diesem Rezept inspiriert hat), finden Sie unter *http://github.com/cynshard/arduino-ntp*.

Wenn Sie eine Arduino-Release vor 1.0 verwenden, können Sie die UDP-Bibliothek von *https://bitbucket.org/bjoern/arduino_osc/src/tip/libraries/Ethernet/* herunterladen.

15.15 Pachube-Feeds überwachen

Problem

Arduino soll auf Informationen eines Webdienstes reagieren, der Sicherheit und Daten-Backups bietet. Pachube ist ein Web-basierter Dienst, der Daten-Feeds in Echtzeit bietet. Sie wollen basierend auf den Datenwerten eines Pachube-Feeds ein Gerät aktivieren oder einen Alarm auslösen.

Lösung

Der folgende Sketch liest die ersten vier Datenfelder aus Feed Nr. 504 ein und gibt sie über den seriellen Monitor aus:

```
/*
 * Monitor Pachube feed
```

```
 * Lese Feed mit V2-API im CSV-Format
 */

#include <SPI.h>
#include <Ethernet.h>

const int  feedID      =  504; // ID des entfernten
                       // Pachube-Feeds, mit dem
                       // Sie die Verbindung herstellen
const int  streamCount    =    4; // Zahl der einzulesenden Daten-Streams
const long PACHUBE_REFRESH = 600000; // Alle 10 Minuten aktualisieren
const long PACHUBE_RETRY  = 10000;  // Bei Verbindungsfehler/-reset
                       // 10 Sekunden warten
                       // Darf nicht unter 5 Sek. liegen

#define PACHUBE_API_KEY  "your key here . . ." // Durch Ihren API-Schlüssel ersetzen

// MAC-Adresse; muss innerhalb des Netzwerks eindeutig sein
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };
char serverName[] = "api.pachube.com";

int streamData[streamCount];   // Typ bei Bedarf an Ihre Daten anpassen

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 if (Ethernet.begin(mac) == 0) {
  Serial.println(F("Ethernet-Konfiguration ueber DHCP fehlgeschlagen"));
  // Weitermachen zwecklos, in Endlosschleife warten
  for(;;)
   ;
 }
}

void loop()
{
  if( getFeed(feedID, streamCount) == true)
  {
    for(int id = 0; id < streamCount; id++){
     Serial.println( streamData[id]);
    }
    Serial.println("--");
    delay(PACHUBE_REFRESH);
  }
  else
  {
    Serial.println(F("Konnte Feed nicht empfangen"));
    delay(PACHUBE_RETRY);
  }
}

// Gibt wahr zurück, wenn Verbindung hergestellt und alle angeforderten Streams eingelesen
// werden konnten
boolean getFeed(int feedId, int streamCount )
{
```

```
boolean result = false;
 if (client.connect(serverName, 80)>0) {
  client.print(F("GET /v2/feeds/"));
  client.print(feedId);
  client.print(F(".csv HTTP/1.1\r\nHost: api.pachube.com\r\nX-PachubeApiKey: "));
  client.print(PACHUBE_API_KEY);
  client.print("\r\nUser-Agent: Arduino 1.0");
  client.println("\r\n");
 }
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
  Serial.println("Verbunden");
  if( client.find("HTTP/1.1") && client.find("200 OK") )
    result = processCSVFeed(streamCount);
  else
    Serial.println("Trenne Verbindung - kein 200 OK");
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
 return result;
}

int processCSVFeed(int streamCount)
{
 int processed = 0;
 client.find("\r\n\r\n"); // Leerzeile zeigt Anfang der Daten an
 for(int id = 0; id < streamCount; id++)
 {
  int id = client.parseInt(); // Sie können das zur Wahl einer bestimmten ID nutzen
  client.find(","); // Letzten Zeitstempel überspringen
  streamData[id] = client.parseInt();
  processed++;
 }
 return(processed == streamCount ); // Wahr zurückgeben, wenn alle Daten empfangen wurden
}
```

Diskussion

Um Pachube nutzen zu können, müssen Sie zuerst einen Account einrichten. Die Pachube Quickstart-Seite erklärt wie: *http://community.pachube.com/?q=node/4*. Sobald Sie sich angemeldet haben, erhalten Sie per E-Mail einen Benutzernamen und einen API-Schlüssel. Tragen Sie den Schlüssel in die folgende Zeile des Sketches ein:

```
#define PACHUBE_API_KEY "your key here . . ." // Durch Ihren API-Schlüssel ersetzen
```

Jeder Pachube-Feed (Datenquelle) wird über eine ID identifiziert. Unser Beispiel-Sketch nutzt Feed 504 (Umgebungsdaten aus dem Pachube-Büro). Im folgenden Sketch erfolgt der Zugriff auf die Feeds über die Methode `getFeed`, der die Feed-ID und die Zahl der Datenelemente übergeben wird. Bei Erfolg gibt `getFeed` `true` zurück und Sie können die

Daten mit der processFeed-Methode verarbeiten. Sie liefert den Wert der Sie interessierenden Daten zurück (jedes Datenelement wird bei Pachube als *Stream* bezeichnet).

Pachube unterstützt eine Reihe von Datenformaten und der obige Sketch nutzt das einfachste: CSV (unter *http://api.pachube.com/v2/#data-formats* erfahren Sie mehr über die Pachube-Datenformate).

Sie können zusätzliche Informationen aus einem Feed extrahieren, wenn Sie das XML-Format nutzen. Hier ein Beispiel der Pachube XML-Daten des in diesem Rezept verwendeten Streams:

```
<environment updated="2010-06-08T09:30:11Z" id="504"
  creator="http://www.pachube.com/users/hdr">
  <title>Pachube Office environment</title>
  <feed>http://api.pachube.com/v2/feeds/504.xml</feed>
  <status>live</status>
  <website>http://www.haque.co.uk/</website>
  <tag>Tag1</tag>
  <tag>Tag2</tag>
  <location domain="physical" exposure="indoor" disposition="fixed">
   <name>office</name>
   <lat>51.5235375648154</lat>
   <lon>-0.0807666778564453</lon>
   <ele>23.0</ele>
  </location>
  <data id="0">
   <tag>humidity</tag>
   <min_value>0.0</min_value>
   <max_value>847.0</max_value>
   <current_value at="2010-06-08T09:30:11.000000Z">311</current_value>
  </data>
  </environment>
```

Der Titel Pachube Office environment zeigt den Anfang der Daten an. Jeder Stream wird durch den Tag data id= (gefolgt von der numerischen Stream-ID) eingeleitet. Die Funktion processXMLFeed im folgenden Sketch nutzt diese Information, um die gewünschte Feed-ID zu finden und die minimalen, maximalen und aktuellen Werte aus dem gewünschten Feed zu extrahieren:

```
/*
 * Monitor Pachube feed
 * V2-API mit XML-Format
 * Steuert Servo über den Wert eines bestimmten Streams
 */

#include <SPI.h>
#include <Ethernet.h>

#include <Servo.h> // Dieser Sketch steuert einen Servo

const int feedID      =  504; // Gewünschter Pachube-Feed
const int streamToGet   =  0;  // Daten-ID des gewünschten Streams

const long PACHUBE_REFRESH = 600000; // Alle 10 Minuten aktualisieren
const long PACHUBE_RETRY  = 10000;  // Bei Verbindungsfehler/-reset
```

```
#define PACHUBE_API_KEY "your key here . . ." // Ihren API-Schlüssel eintragen

// MAC-Adresse, muss in Ihrem Netzwerk eindeutig sein
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };
char serverName[] = "api.pachube.com";

EthernetClient client;

// Von Pachube zurückgelieferte Stream-Werte
int currentValue; // Aktueller Wert des Streams
int minValue;   // Minimaler Wert des Streams
int maxValue;   // Maximaler Wert des Streams

Servo myservo;  // Servo-Objekt
void setup()
{
 Serial.begin(9600);
 myservo.attach(9);  // Servo an Pin 9 mit Servo-Objekt verbinden

 if (Ethernet.begin(mac) == 0) {
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  // Weitermachen zwecklos, in Endlosschleife warten
  for(;;)
   ;
 }
}

void loop()
{
 if( getFeed(feedID, streamToGet) == true)
 {
  Serial.print(F("Wert="));
  Serial.println(currentValue);
  // Im Bereich von 0 bis 90 Grad proportional positionieren
  int servoPos = map(currentValue, minValue, maxValue, 0,90);
  myservo.write(servoPos);
  Serial.print(F("Pos="))

;
  Serial.println(servoPos);
  delay(PACHUBE_REFRESH);
 }
 else
 {
  Serial.println(F("Konnte Feed nicht lesen"));
  delay(PACHUBE_RETRY);
 }
}

// Gibt wahr zurück, wenn Verbindung mit angefordertem Stream hergestellt und alle angeforderte
Daten abgerufen werden konnten
boolean getFeed(int feedId, int streamId )
{
 boolean result = false;
 if (client.connect(serverName, 80)>0) {
  Serial.print("Verbinde mit Feed ");
```

```
  Serial.print(feedId);
  Serial.print(" ... ");
  client.print("GET /v2/feeds/");
  client.print(feedId);
  client.print(".xml HTTP/1.1\r\nHost: api.pachube.com\r\nX-PachubeApiKey: ");
  client.print(PACHUBE_API_KEY);
  client.print("\r\nUser-Agent: Arduino 1.0");
  client.println("\r\n");
 }
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 if (client.connected()) {
  Serial.println("Verbunden");
  if( client.find("HTTP/1.1") && client.find("200 OK") )
   result = processXMLFeed(streamId);
  else
   Serial.println("Trenne Verbindung - kein 200 OK");
 }
 else {
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
 return result;
}

boolean processXMLFeed(int streamId)
{
 client.find("<environment updated=");
 for(int id = 0; id <= streamId; id++)
 {
  if( client.find( "<data id=" ) ){  // Finde nächstes Datenfeld
   if(client.parseInt()== streamId){ // Ist das unser Stream?
    if(client.find("<min_value>")){
     minValue = client.parseInt();
     if(client.find("<max_value>")){
      maxValue = client.parseInt();
      if(client.find("<current_value ")){
       client.find(">"); // Bis zur spitzen Klammer suchen
       currentValue = client.parseInt();
       return true; // Alle benötigten Daten gefunden
      }
     }
    }
   }
  }
  else {
   Serial.print(F("Kann Daten für ID "));
   Serial.println(id);
   Serial.print(F(" nicht finden"));   }
 }
 return false;  // Parsing der Daten fehlgeschlagen
}
```

Das Stream-Parsing von Arduino 1.0 wird genutzt, um nach den gewünschten Feldern zu suchen. Eine Liste aller Felder finden Sie in der Dokumentation der Pachube-API.

Siehe auch

Die Pachube API-Dokumentation: *http://api.pachube.com/v2/*.

Eine Arduino-Bibliothek, die den Zugriff auf Pachube vereinfacht, finden Sie hier: *http://code.google.com/p/pachubelibrary/*.

15.16 Informationen an Pachube senden

Problem

Arduino soll Feeds auf Pachube aktualisieren. Zum Beispiel sollen die Werte der an den Arduino angeschlossenen Sensoren in einem Pachube-Feed veröffentlicht werden.

Lösung

Der folgende Sketch liest die Temperatur-Sensoren ein, die mit den analogen Eingangspins (siehe Rezept 6.8) verbunden sind, und sendet die Daten an Pachube:

```
/*
 * Update Pachube feed
 * Sendet Temperatur von (bis zu) sechs LM35-Sensoren
 * V2 API
 */

#include <SPI.h>
#include <Ethernet.h>

const int feedID      =  2955; // ID des Feeds
const int streamCount    =    6;  // Anzahl der zu sendenden Daten-Streams (Sensoren)
const long REFRESH_INTERVAL = 60000;  // Jede Minute aktualisieren
// Bei Verbindungsfehler/-reset 10 Sekunden warten
// Darf nicht unter 5 Sekunden liegen
const long RETRY_INTERVAL  = 10000;

#define PACHUBE_API_KEY "Your key here . . . " // Ihren API-Schlüssel eintragen

// Muss für Ihr Netzwerk eindeutig sein
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };
char serverName[] = "www.pachube.com";

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 Serial.println("Bereit");
 if (Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
}
```

```
void loop()
{
 String dataString = "";
 for (int id = 0; id < streamCount; id++)
 {
   int temperature = getTemperature(id);
   dataString += String(id);
   dataString  += ",";
   dataString += String(temperature);
   dataString  += "\n";
 }
 if ( putFeed(feedID, dataString, dataString.length()) == true)
 {
   Serial.println("Feed aktualisiert");
   delay(REFRESH_INTERVAL);
 }
 else
 {
    Serial.println("Konnte Feed nicht aktualisieren");
    delay(RETRY_INTERVAL);
 }
}

// true zurückgeben, wenn Verbindung hergestellt und Daten gesendet werden konnten
boolean putFeed(int feedId, String feedData, int length )
{
boolean result = false;
 if (client.connect(serverName, 80)>0) {
  Serial.print("Verbinde Feed "); Serial.println(feedId);
  client.print("PUT /v2/feeds/");
  client.print(feedId);
  client.print(".csv HTTP/1.1\r\nHost: api.pachube.com\r\nX-PachubeApiKey: ");
  client.print(PACHUBE_API_KEY);
  client.print("\r\nUser-Agent: Arduino 1.0");
  client.print("\r\nContent-Type: text/csv\r\nContent-Length: ");
  client.println(length+2, DEC); // für cr/lf
  client.println("Connection: close");
  client.println("\r\n");
  // jetzt Daten ausgeben
  Serial.println(feedData); // Optional an seriellen Monitor ausgeben
  client.print(feedData);
  client.println("\r\n");
 }
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 // Response-String
 if (client.connected()) {
  Serial.println("Verbunden");
  if(client.find("HTTP/1.1") && client.find("200 OK") ){
    result = true;
  }
  else
    Serial.println("Verbindung getrennt - kein 200 OK");
 }
 else {
```

```
  Serial.println("Verbindung getrennt");
 }
 client.stop();
 client.flush();
 return result;
}

// Temperatur (gerundet auf den nächsten Grad-Wert) zurückgeben
int getTemperature(int pin)
{
  int value = analogRead(pin);
  int celsius = (value * 500L) / 1024; // 10mv pro Grad
  return celsius;
}
```

Diskussion

Das Rezept ähnelt Rezept 15.15, verwendet aber die `putFeed`-Methode, um Daten an Pachube zu senden. Im Beispiel werden Informationen von Temperatursensoren gesendet. Für Code, der zu Ihrer Anwendung passt, sehen Sie sich das Kapitel an, in dem dieser Sensor behandelt wird.

Pachube benötigt die Anzahl der Zeichen der Daten, bevor der eigentliche Inhalt gesendet wird. Sie wird über die Stringverkettungs-Funktion (siehe Rezept 2.5) bestimmt. Zuerst wird ein String erzeugt, der alle Felder enthält, und dann wird `String.length()`genutzt, um dessen Länge zu bestimmen.

Der folgende Sketch nutzt eine andere Technik, die kein RAM für Stringdaten benötigt. Er nutzt eine neue, bei Arduino 1.0 eingeführte Fähigkeit, die die Zahl der ausgegebenen Zeichen zurückliefert. Die Funktion `outputCSV` zählt die Anzahl der ausgegebenen Zeichen und gibt sie zurück. Sie wird zuerst aufgerufen, um die Gesamtzahl der über den seriellen Port ausgegebenen Zeichen zu berechnen. Sie wird dann erneut aufgerufen, um die Zeichen an den Ethernet-Client zu senden, der mit Pachube verbunden ist:

```
/*
 * Update Pachube feed
 * Sendet Temperatur im Fließkomma-Format von (bis zu) sechs LM35-Sensoren
 * V2-API
 */

#include <SPI.h>
#include <Ethernet.h>

const int feedID      =  2955; // ID des Feeds
const int streamCount   =   6; // Zahl zu sendender Daten-Streams (Sensoren)
const long REFRESH_INTERVAL = 60000;  // Jede Minute aktualisieren
// Bei Verbindungsfehler/-reset 10 Sekunden warten
// Darf nicht unter 5 Sekunden liegen
const long RETRY_INTERVAL  = 10000;

#define PACHUBE_API_KEY "Your key here . . . " // Ihren API-Schlüssel eintragen
```

```
// Muss innerhalb des Netzwerks eindeutig sein
byte mac[] = { 0xCC, 0xAC, 0xBE, 0xEF, 0xFE, 0x91 };
char serverName[] = "www.pachube.com";

EthernetClient client;

void setup()
{
 Serial.begin(9600);
 Serial.println("Bereit");
 if(Ethernet.begin(mac) == 0) { // Ethernet starten
  Serial.println("Ethernet-Konfiguration ueber DHCP fehlgeschlagen");
  while(true)  // Weitermachen zwecklos, in Endlosschleife warten
   ;
 }
}

void loop()
{
 int contentLen = outputCSV(Serial); // Zahl der Zeichen ermitteln
 if( putFeed(feedID, contentLen) == true){
  Serial.println("Feed aktualisiert");
  delay(REFRESH_INTERVAL);
 }
 else {
   Serial.println("Konnte Feed nicht aktualisieren");
   delay(RETRY_INTERVAL);
 }
}

// true zurückgeben, wenn Verbindung hergestellt und Daten gesendet werden konnten
boolean putFeed(int feedId, int length)
{
boolean result = false;
 if (client.connect(serverName, 80)>0) {
  Serial.print("Verbinde Feed "); Serial.println(feedId);
  client.print("PUT /v2/feeds/");
  client.print(feedId);
  client.print(".csv HTTP/1.1\r\nHost: api.pachube.com\r\nX-PachubeApiKey: ");
  client.print(PACHUBE_API_KEY);
  client.print("\r\nUser-Agent: Arduino 1.0");
  client.print("\r\nContent-Type: text/csv\r\nContent-Length: ");
  client.println(length+2, DEC); // allow for cr/lf
  client.println("Connection: close");
  client.println("\r\n");
  outputCSV(client);
  client.println("\r\n");
 }
 else {
  Serial.println("Verbindung fehlgeschlagen");
 }
 // response string
 if (client.connected()) {
  Serial.println("Verbunden");
  if(client.find("HTTP/1.1") && client.find("200 OK") ){
   result = true;
  }
```

```
  else
    Serial.println("Trenne Verbindung - kein 200 OK");
 }
 else {
  Serial.println("Verbindung getrent");
 }
 client.stop();
 client.flush();
 return result;
}

int outputCSV(Stream &stream)
{
 int count = 0;
 for(int id = 0; id < streamCount; id++) {
  float temperature = getTemperature(id);
  count += stream.print(id,DEC);
  count += stream.print(',');
  count += stream.print(temperature,1); // Eine Stelle hinterm Komma
  count += stream.print("\n");
 }
 return count;
}

float getTemperature(int inPin)
{
  int value = analogRead(inPin);
  float millivolts = (value / 1024.0) * 5000;  //  Siehe Rezept 6.8
  return millivolts / 10;              // 10mV pro Grad Celsius
}
```

KAPITEL 16

Bibliotheken nutzen, ändern und aufbauen

16.0 Einführung

Bibliotheken erweitern die Funktionalität der Arduino-Umgebung. Sie erweitern die zur Verfügung stehenden Befehle um Fähigkeiten, die im Arduino-Kern nicht zur Verfügung stehen. Mit Bibliotheken können Sie zusätzliche Features hinzufügen, die jedem Sketch zur Verfügung stehen, sobald die Bibliothek installiert ist.

Die Arduino-Software-Distribution enthält fest integrierte Bibliotheken, die gängige Aufgaben übernehmen. Diese Bibliotheken werden in Rezept 16.1 diskutiert.

Bibliotheken stellen auch eine gute Möglichkeit dar, Code zu teilen, der für andere nützlich sein könnte. Viele Bibliotheken von Drittanbietern stellen spezialisierte Fähigkeiten zur Verfügung. Sie können im Arduino Playground und über andere Sites heruntergeladen werden. Viele der in den früheren Kapitel behandelten Bauelemente nutzen Bibliotheken, um den Zugriff auf diese Geräte zu vereinfachen.

Bibliotheken können aber auch komplexen Code kapseln, um dessen Verwendung zu vereinfachen. Ein Beispiel ist die bei Arduino mitgelieferte Wire-Bibliothek, die einen Großteil der Komplexität der Hardware-Kommunikation auf unterster Ebene vor uns versteckt (siehe Kapitel 13).

Dieses Kapitel erläutert, wie man Bibliotheken nutzt und anpasst. Es zeigt auch, wie man eigene Bibliotheken aufbaut.

16.1 Mitgelieferte Bibliotheken nutzen

Problem

Sie wollen Bibliotheken in Ihrem Sketch nutzen, die bei der Arduino-Distribution mitgeliefert werden.

Lösung

Dieses Rezept zeigt, wie Sie Funktionen einer Arduino-Bibliothek in Ihrem Sketch nutzen können.

Eine Liste der verfügbaren Bibliotheken können Sie sich im IDE-Menü über Sketch→Import Library anzeigen lassen. Es erscheint eine Liste mit allen verfügbaren Bibliotheken. Etwa das erste Dutzend sind die Bibliotheken, die mit Arduino mitgeliefert werden. Eine horizontale Linie trennt diese von den Bibliotheken, die Sie selbst heruntergeladen und installiert haben.

Klicken Sie eine Bibliothek an, wird sie in den aktuellen Sketch eingebunden, indem die folgende Zeile zu Beginn des Sketches eingefügt wird:

```
#include <NamedergewähltenBibliothek.h>
```

Das sorgt dafür, dass die Funktionen der Bibliothek in Ihrem Sketch zur Verfügung stehen.

Die Arduino-IDE aktualisiert die Liste der verfügbaren Bibliotheken nur beim Start. Wenn Sie eine Bibliothek installieren, nachem die IDE bereits läuft, müssen Sie die IDE beenden und neu starten, damit die neue Bibliothek erkannt wird.

Die Arduino-Bibliotheken sind in der Referenz auf *http://arduino.cc/en/Reference/Libraries* dokumentiert und jede Bibliothek enthält Beispiel-Sketches, die ihre Verwendung demonstrieren. Kapitel 1 zeigt, wie man in der IDE an die Beispiele gelangt.

Die bei Arduino Version 1.0 enthaltenen Bibliotheken sind:

EEPROM
: Liest und schreibt Daten aus/in Speicher, dessen Inhalt auch erhalten bleibt, wenn der Strom ausgeschaltet wird. Siehe Kapitel 17.

Ethernet
: Wird zur Kommunikation mit dem Arduino Ethernet-Shield oder Arduino Ethernet-Board genutzt. Siehe Kapitel 15.

Firmata
: Ein Protokoll, das die serielle Kommunikation und die Steuerung des Boards vereinfacht.

LiquidCrystal
: Zur Steuerung kompatibler LC-Displays; siehe Kapitel 11.

SD
: Lesen und Schreiben von Dateien von/an SD-Karten über externe Hardware.

Servo
: Steuerung von Servomotoren; siehe Kapitel 8.

SoftwareSerial
: Stellt zusätzliche serielle Ports bereit.

SPI
: Wird für Ethernet- und SPI-Hardware genutzt; siehe Kapitel 13.

Stepper
: Steuerung von Schrittmotoren; siehe Kapitel 8.

Wire
: Zur Steuerung von mit dem Arduino verbundenen I2C-Geräten; siehe Kapitel 13.

Die beiden folgenden Bibliotheken finden sich in Releases vor Arduino 1.0, sind aber nicht länger Teil der Arduino-Distribution:

Matrix
: Hilft bei der Steuerung einer LED-Matrix; siehe Kapitel 7.

Sprite
: Sprites für eine LED-Matrix.

Diskussion

Mit einer bestimmten Hardware innerhalb des Arduino-Chips arbeitende Bibliotheken funktionieren nur mit vordefinierten Pins. Beispiele für diese Art Bibliotheken sind Wire und SPI. Bibliotheken, die dem Benutzer die Wahl der Pins erlauben, erledigen das üblicherweise in `setup`; Servo, LiquidCrystal und Stepper sind Beispiele für diese Art von Bibliotheken. Informationen zur Konfiguration finden Sie in der Dokumentation der jeweiligen Bibliothek.

Das `#include` einer Bibliothek fügt den Bibliothekscode hinter den Kulissen in Ihren Sketch ein. Das bedeutet, dass sich die Größe Ihres Sketches (wie sie am Ende der Kompilierung gemeldet wird) erhöht, aber der Arduino Build-Prozess ist clever genug, nur den Teil der Bibliothek einzufügen, den der Sketch tatsächlich verwendet. Sie müssen sich also keine Gedanken um den Speicherbedarf von Methoden machen, die gar nicht verwendet werden. Daher müssen Sie sich auch keine Gedanken darum machen, ob ungenutzte Funktionen die Codemenge reduzieren, die Sie in den Sketch packen können.

Die bei Arduino mitgelieferten Bibliotheken (und viele Bibliotheken von Drittanbietern) enthalten Beispiel-Sketches, die die Verwendung der Bibliothek demonstrieren. Diese sind über das Menü File→Examples zugänglich.

Siehe auch

Die Arduino-Referenz für Bibliotheken: *http://arduino.cc/en/Reference/Libraries*

16.2 Bibliotheken von Drittanbietern installieren

Problem

Sie wollen eine Bibliothek für den Arduino nutzen, die nicht in der Standard-Distribution enthalten ist.

Lösung

Laden Sie die Bibliothek herunter, die häufig in Form einer *.zip*-Datei vorliegt. Entpacken Sie diese Datei und Sie erhalten einen Ordner, der der gleichen Namen hat wie die Bibliothek. Dieser Ordner muss in einen Ordner namens *libraries* im Arduino-Dokumentenordner kopiert werden. Um diesen Ordner zu finden, öffnen Sie Preferences (Arduino→Preferences auf dem Mac; File→Preferences unter Windows) und notieren sich die Position des Sketchbooks. Wechseln Sie mit einem Dateisystem-Browser (Windows Explorer oder Mac OS X Finder) oder über das Terminal in dieses Verzeichnis. Gibt es keinen *libraries*-Ordner, legen Sie einen an und kopieren den entpackten Ordner dort hinein.

Wenn die Arduino IDE noch läuft, beenden und starten Sie sie neu. Die IDE durchsucht diesen Ordner nur beim Start nach Bibliotheken. Wenn Sie sich nun das Menü Sketch→Import Library ansehen, erscheint am unteren Rand (unter der grauen Linie und dem Wort *Contributed*) die gerade hinzugefügte Bibliothek.

Enthält die Bibliothek Beispiel-Sketches, können Sie diese über das IDE-Menü ansehen. Klicken Sie auf File→Examples und Sie finden die Beispiele unter dem Namen der Bibliothek in einem Abschnitt zwischen den allgemeinen Beispielen und den Beispielen zu den mit Arduino ausgelieferten Bibliotheken.

Diskussion

Eine große Zahl von Bibliotheken steht von Drittanbietern zur Verfügung. Viele sind von hoher Qualität, werden aktiv gepflegt und umfassen eine gute Dokumentation sowie Beispiel-Sketches. Der Arduino Playground ist ein guter Ort, um nach Bibliotheken zu suchen: *http://www.arduino.cc/playground/*.

Achten Sie darauf, Bibliotheken mit einer guten Dokumentation und mit Beispielen zu nutzen. Schauen Sie in den Arduino-Foren, ob es Threads (Diskussionen) gibt, die diese Bibliothek behandeln. Bibliotheken, die für frühe Arduino-Versionen entwickelt wurden, können Probleme bereiten, wenn man sie mit der neuesten Arduino-Version nutzt. Möglicherweise müssen Sie sehr viel lesen (die Threads beliebter Bibliotheken enthalten Hunderte Postings), um Informationen darüber zu finden, wie man ältere Bibliotheken mit der neuesten Arduino-Release nutzen kann.

Wenn die Bibliotheks-Beispiele nicht im Examples-Menü erscheinen, oder wenn Sie die Meldung erhalten, dass die Bibliothek nicht gefunden werden konnte (»Library not found«), dann prüfen Sie, ob der Bibliotheksordner am richtigen Ort liegt und ob der

Name richtig geschrieben wurde. Ein Bibliotheksordner namens *<BibliotheksName>* muss eine Datei namens *<LibraryName>.h* in genau der gleichen Schreibweise enthalten. Stellen Sie sicher, dass auch alle weiteren Dateien im Ordner vorhanden sind, die von der Bibliothek benötigt werden.

16.3 Eine Bibliothek anpassen

Problem

Sie möchten das Verhalten einer Bibliothek anpassen, etwa um deren Fähigkeiten zu erweitern. So unterstützt die TimeAlarms-Bibliothek aus Kapitel 12 nur sechs Alarme, Sie benötigen aber mehr (siehe Rezept 12.5).

Lösung

Die Time- und TimeAlarms-Bibliotheken werden in Kapitel 12 beschrieben. Sehen Sie sich Rezept 12.5 an, um sich mit der Standard-Funktionalität vertraut zu machen. Die Bibliotheken können von der Website zu diesem Buch (*http://shop.oreilly.com/product/0636920022244.do*), oder von *http://www.arduino.cc/playground/uploads/Code/Time.zip* (dieser Download umfasst beide Bibliotheken) heruntergeladen werden.

Wenn Sie die Time- und TimeAlarms-Bibliotheken installiert haben, kompilieren Sie den folgenden Sketch und laden ihn auf den Arduino hoch. Der Sketch versucht, sieben Alarme einzurichten – einen mehr, als die Bibliothek unterstützt. Jeder `Alarm`-Task gibt einfach die Task-Nummer aus:

```
/*
 multiple_alarms Sketch
 Verwendet mehr Timer, als die Bibliothek standardmäßig unterstützt -
 Sie müssen die Header-Datei editieren, um mehr als 6 Alarme zu unterstützen
 */

#include <Time.h>
#include <TimeAlarms.h>

int currentSeconds = 0;

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

 // create 7 alarm tasks
 Alarm.timerRepeat(1, repeatTask1);
 Alarm.timerRepeat(2, repeatTask2);
 Alarm.timerRepeat(3, repeatTask3);
 Alarm.timerRepeat(4, repeatTask4);
 Alarm.timerRepeat(5, repeatTask5);
 Alarm.timerRepeat(6, repeatTask6);
 Alarm.timerRepeat(7, repeatTask7); // 7ter Timer
}
```

```
void repeatTask1()
{
 Serial.print("Task 1 ");
}

void repeatTask2()
{
 Serial.print("Task 2 ");
}
void repeatTask3()
{
 Serial.print("Task 3 ");
}

void repeatTask4()
{
 Serial.print("Task 4 ");
}

void repeatTask5()
{
 Serial.print("Task 5 ");
}

void repeatTask6()
{
 Serial.print("Task 6 ");
}

void repeatTask7()
{
 Serial.print("Task 7 ");
}

void loop()
{
 if(second() != currentSeconds)
 {
  // Zeit jede Sekunde ausgeben
  // Task-Nummer wird ausgegeben, wenn der Alarm für diesen Task angestoßen wird
   Serial.println();
   Serial.print(second());
   Serial.print("->");
   currentSeconds = second();
   Alarm.delay(1); // Alarm.delay muss aufgerufen werden, um die Alarme zu verarbeiten
 }
}
```

Öffnen Sie den seriellen Monitor und sehen Sie sich die Ausgabe an. Nach neun Sekunden sieht die Ausgabe so aus:

```
1->Task 1
2->Task 1  Task 2
3->Task 1  Task 3
4->Task 1  Task 2  Task 4
5->Task 1  Task 5
```

```
6->Task 1 Task 2 Task 3 Task 6
7->Task 1
8->Task 1 Task 2 Task 4
9->Task 1 Task 3
```

Der für sieben Sekunden angesetzte Task wird nicht ausgeführt, weil die Bibliothek nur sechs Timer-»Objekte« zur Verfügung stellt.

Sie können das erweitern, indem Sie die Bibliothek anpassen. Wechseln Sie in den Bibliotheksordner in ihrem Arduino *Documents*-Ordner.

Sie können die Lage des Verzeichnisses mit dem Sketchbook-Ordner ermitteln, indem Sie File→Preferences (unter Windows) oder Arduino →Preferences (auf einem Mac) in der IDE auswählen. Es erscheint eine Dialogbox, die die Lage des Sketchbooks zeigt.

Wenn die Time- und TimeAlarms-Bibliotheken installiert sind (beide Bibliotheken befinden sich in der heruntergeladenen Datei), wechseln Sie in den Ordner *Libraries\TimeAlarms*. Öffnen Sie die Header-Datei *TimeAlarms.h* (weitere Details zu Header-Dateien finden Sie in Rezept 16.4). Sie können die Datei mit einem beliebigen Texteditor bearbeiten, z.B. mit Notepad unter Windows oder TextEdit auf einem Mac.

Am Anfang der *TimeAlarms.h*-Datei sehen Sie die folgenden Zeilen:

```
#ifndef TimeAlarms_h
#define TimeAlarms_h

#include <inttypes.h>
#include "Time.h"
#define dtNBR_ALARMS 6
```

Die maximale Anzahl der Alarme wird durch `dtNbr_ALARMS` definiert.

Ändern Sie:

```
#define dtNBR_ALARMS 6
```

in:

```
#define dtNMBR_ALARMS 7
```

und speichern Sie die Datei.

Laden Sie den Sketch erneut auf den Arduino hoch, und diesmal sollte die Ausgabe wie folgt aussehen:

```
1->Task 1
2->Task 1 Task 2
3->Task 1 Task 3
4->Task 1 Task 2 Task 4
5->Task 1 Task 5
6->Task 1 Task 2 Task 3 Task 6
7->Task 1 Task 7
8->Task 1 Task 2 Task 4
9->Task 1 Task 3
```

Wie Sie sehen, wird Task 7 jetzt nach sieben Sekunden aktiviert.

Diskussion

Die von einer Bibliothek gebotenen Möglichkeiten sind häufig ein Kompromiss aus den von der Bibliothek verwendeten Ressourcen und den für den Rest Ihres Sketches zur Verfügung stehenden Ressourcen. Häufig ist es möglich, das an Ihren Bedarf anzupassen. Zum Beispiel könnten Sie die Speichernutzung der Serial-Bibliothek reduzieren müssen, damit Ihrem Sketch mehr RAM zur Verfügung steht. Oder Sie könnten den Speicherplatz erhöhen müssen, den eine Bibliothek in Ihrer Anwendung nutzt. Die Entwickler legen die Bibliotheken üblicherweise so an, dass sie typische Szenarieren abdecken. Benötigt Ihre Anwendung Fähigkeiten, die die Entwickler nicht berücksichtigt haben, können Sie die Bibliothek möglicherweise so anpassen, dass sie Ihren Anforderungen genügt.

In diesem Beispiel stellt die TimeAlarms-Bibliothek im RAM Platz für sechs Alarme bereit. Jeder benötigt etwa ein Dutzend Bytes und der Platz wird auch dann reserviert, wenn weniger genutzt werden. Die Anzahl der Alarme ist in der Header-Datei der Bibliothek definiert (der Header ist eine Datei namens *TimeAlarms.h* im Ordner *TimeAlarms*). Hier die ersten Zeilen aus *TimeAlarms.h*:

```
#ifndef TimeAlarms_h
#define TimeAlarms_h

#include <inttypes.h>

#include "Time.h"

#define dtNBR_ALARMS 6
```

Bei der TimeAlarms-Bibliothek wird die maximale Anzahl von Alarmen in einer `#define`-Anweisung festgelegt. Da Sie diesen Wert erhöht und die Header-Datei gespeichert haben, nutzt der Sketch die neue Obergrenze, wenn er neu kompiliert und hochgeladen wird.

Manchmal werden Konstanten genutzt, um Eigenschaften wie die Taktfrequenz eines Boards zu definieren. Arbeitet das Board mit einer anderen Geschwindigkeit, kann es zu unerwarteten Ergebnissen kommen. Sie können das Problem dann üblicherweise beheben, indem Sie die Werte in der Header-Datei an ihr Board anpassen.

Wenn Sie die Header-Datei ändern und die Bibliothek dann gar nicht mehr funktioniert, können Sie die Bibliothek erneut herunterladen und den Originalzustand wiederherstellen.

Siehe auch

Rezept 16.4 erläutert, wie man die Funktionalität von Bibliotheken erweitern kann.

16.4 Eine eigene Bibliothek entwickeln

Problem

Sie wollen eine eigene Bibliothek entwickeln. Bibliotheken sind eine bequeme Möglichkeit, Code wiederzuverwenden, und stellen eine gute Lösung dar, um ihn mit anderen Nutzern zu teilen.

Lösung

Eine Bibliothek ist eine Sammlung von Methoden und Variablen, die in einem Format kombiniert werden, die es Nutzern ermöglicht, in standardisierter Form auf Funktionen und Variablen zuzugreifen.

Die meisten Arduino-Bibliotheken sind als Klassen geschrieben. Wenn Sie mit C++ oder Java vertraut sind, kennen Sie sich auch mit Klassen aus. Bibliotheken lassen sich aber auch ohne Klassen entwickeln und dieses Rezept zeigt, wie das geht.

Das Rezept erläutert, wie Sie die `BlinkLED`-Funktion im Sketch aus Rezept 7.1 in eine Bibliothek umwandeln.

In Rezept 7.1 finden Sie ein Schaltdiagramm und eine Erklärung der Schaltung. Die Bibliothek enthält die `blinkLED`-Funktion aus diesem Rezept. Hier der Sketch, der zu Testen der Bibliothek verwendet wird:

```
/*
 * blinkLibTest
 */

#include "blinkLED.h"

const int firstLedPin  = 3;     // LED-Pins
const int secondLedPin = 5;
const int thirdLedPin  = 6;

void setup()
{
 pinMode(firstLedPin, OUTPUT);   // LED-Pins aus Ausgang delarieren
 pinMode(secondLedPin, OUTPUT);
 pinMode(thirdLedPin, OUTPUT);
}

void loop()
{
 // Jede LED für 1000 Millisekunden (1 Sekunde) blinken lassen
 blinkLED(firstLedPin, 1000);
 blinkLED(secondLedPin, 1000);
 blinkLED(thirdLedPin, 1000);
}
```

Die blinkLED-Funktion aus Rezept 7.1 muss aus dem Sketch entfernt und in eine separaten Datei namens *blinkLED.cpp* verschoben werden (in der Diskussion werden Details zu den *.cpp*-Dateien erläutert):

```
/* blinkLED.cpp
 * einfache Bibliothek, die eine LED für die angegebene Dauer in Millisekunden blinken lässt
 */
#include "Arduino.h"  // Wprogram.h für Arduino-Versionen vor 1.0
#include "blinkLED.h"

// LED am angebenen Pin für die angegebene Dauer in Millisekunden blinken lassen
void blinkLED(int pin, int duration)
{
 digitalWrite(pin, HIGH);   // LED einschalten
 delay(duration);
 digitalWrite(pin, LOW);    // LED ausschalten
 delay(duration);
}
```

Die meisten Bibliotheksautoren sind Programmierer, die den von ihnen bevorzugten Programmier-Editor nutzen, doch Sie können jeden beliebigen Texteditor verwenden, um diese Dateien anzulegen.

Legen Sie die Header-Datei *blinkLED.h* wie folgt an:

```
/*
 * blinkLED.h
 * Headerdatei für BlinkLED-Bibliothek
 */
#include "Arduino.h"

void blinkLED(int pin, int duration); // Funktions-Prototyp
```

Diskussion

Wir nennen die Bibliothek »blinkLED« und speichern sie im Bibliotheksordner (siehe Rezept 16.2). Legen Sie ein Unterverzeichnis namens *blinkLED* im Bibliotheksordner an und kopieren Sie *blinkLED.h* und *blinkLED.cpp* in dieses Verzeichnis.

Die Funktion blinkLED aus Rezept 7.1 wird aus dem Sketch entfernt und in eine Bibliotheksdatei namens *blinkLED.cpp* ausgelagert (Die Erweiterung *.cpp* steht für »C Plus Plus« und enthält den ausführbaren Code).

Die Begriffe *Funktion* und *Methode* werden in der Dokumentation von Arduino-Bibliotheken genutzt, um Codeblöcke wie blinkLED zu referenzieren. Der Begriff *Methode* wird für Funktionsblöcke in Klassen verwendet. Beide Begriffe stehen für die Funktionsblöcke, die durch die Bibliothek zugänglich gemacht werden.

Die Datei *blinkLED.cpp* enthält die Funktion blinkLED, die mit dem Code aus Rezept 7.1 identisch ist. Zusätzlich wurden am Anfang die folgenden beiden Zeilen hinzugefügt:

```
#include "Arduino.h" // Arduino include
#include "blinkLED.h"
```

Die Zeile `#include "Arduino.h"` wird von Bibliotheken benötigt, die Arduino-Funktionen oder -Konstanten verwenden. Ohne diese Zeile meldet der Compiler Fehler für alle in Ihrem Sketch genutzten Arduino-Funktionen.

Arduino.h wurde mit der Release 1.0 eingeführt und ersetzt *WProgram.h*. Wenn Sie Sketches mit älteren Releases kompilieren, können Sie die folgenden bedingten Includes nutzen, um die richtige Variante einzubinden:

```
#if ARDUINO >= 100
#include "Arduino.h // für 1.0 und höher
#else
#include "WProgram.h" // für ältere Releases
#endif
```

Die nächste Zeile, `#include "blinkLED.h"`, bindet die Funktonsdefinitionen (auch *Prototypen* genannt) in Ihre Bibliothek ein. Der Arduino Build-Prozess erzeugt bei der Kompilierung automatisch Prototypen für alle Funktionen – für Bibliotheken werden aber keine Prototypen erzeugt, d.h. Sie müssen für diese Prototypen eine Header-Datei anlegen. Es ist eben diese Header-Datei, die in einen Sketch eingefügt wird, wenn Sie eine Bibliothek über die IDE importieren (siehe Rezept 16.1).

Jede Bibliothek muss eine Datei besitzen, die die Namen der bereitgestellten Funktionen deklariert. Diese Datei wird als *Header-Datei* (oder auch *Include*-Datei) bezeichnet und hat die Form *<BibliotheksName>.h*. In unserem Beispiel heißt die Header-Datei *blinkLED.h* und liegt im gleichen Ordner wie *blinkLED.cpp*.

Die Header-Datei für unsere Bibliothek ist simpel. Sie deklariert nur eine Funktion:

```
void blinkLED(int pin, int duration); // Funktions-Prototyp
```

Das sieht der Funktionsdefinition in *blinkLED.cpp* sehr ähnlich:

```
void blinkLED(int pin, int duration)
```

Doch es gibt einen kleinen, aber feinen Unterschied. An den Prototypen in der Header-Datei ist ein Semikolon angehangen. Das teilt dem Compiler mit, dass es sich nur um eine Deklaration der Form der Funktion handelt, und nicht um den Code. Die Quelldatei *blinkLED.cpp* enthält dieses Semikolon nicht, und das informiert den Compiler darüber, dass es sich um den Quellcode der Funktion handelt.

Bibliotheken können mehr als eine Header- und mehr als eine Implementierungsdatei verwenden. Es muss aber zumindest einen Header geben, der mit dem Namen der Bibliothek übereinstimmt. Eben diese Datei wird am Anfang Ihres Sketches eingefügt, wenn Sie eine Bibliothek importieren.

Ein gutes Buch zu C++ wird beschreiben, wie man Header- und *.cpp*-Dateien zur Entwicklung von Modulen einsetzt. Im Siehe auch-Abschnitt dieses Rezepts finden Sie eine Auswahl.

Sind *blinkLED.cpp* und *blinkLED.h* am richtigen Platz, schließen Sie die IDE und starten Sie sie erneut.

Die Arduino-IDE aktualisiert die verfügbaren Bibliotheken nur beim Start. Wenn Sie eine Bibliothek anlegen, während die IDE läuft, müssen Sie die IDE beenden und wieder neu starten, damit die Bibliothek erkannt wird.

Wenn Sie den blinkLibTest-Sketch hochladen, sollten die drei LEDs blinken.

Die Bibliothek um zusäzliche Funktionen zu erweitern, ist einfach. Zum Beispiel können Sie einige Konstanten aufnehmen. Die Benutzer der Bibliothek können dann diese selbsterklärenden Konstanten anstelle von Millisekunden-Angaben nutzen.

Fügen Sie die drei folgenden Zeilen mit Konstanten hinzu, die traditionell vor dem ersten Funktionsprototypen stehen:

```
// Konstanten für Blinkdauer
const int BLINK_SHORT = 250;
const int BLINK_MEDIUM = 500;
const int BLINK_LONG = 1000;

void blinkLED(int pin, int duration); // Funktions-Prototyp
```

Ändern Sie den `loop`-Code wie folgt und laden Sie den Sketch hoch, um sich die unterschiedlichen Blinkgeschwindigkeiten anzusehen:

```
void loop()
{
 blinkLED(firstLedPin, BLINK_SHORT);
 blinkLED(secondLedPin, BLINK_MEDIUM);
 blinkLED(thirdLedPin, BLINK_LONG);
}
```

Sie müssen die IDE nur schließen und neu starten, wenn Sie den Bibliotheksordner neu angelegt haben, nicht aber bei weiteren Änderungen an der Bibliothek. Ab Arduino-Release 0017 (und höher) eingebundene Bibliotheken werden jedesmal neu kompiliert, wenn der Sketch kompiliert wird. Bei Arduino-Releases vor 0017 musste die Objektdatei der Bibliothek gelöscht werden, damit die Bibliothek neu kompiliert und Änderungen integriert wurden.

Neue Funktionen können einfach hinzugefügt werden. Das folgende Beispiel fügt eine Funktion hinzu, die eine LED n mal blinken lässt, wobei der entsprechende Wert an die Funktion übergeben wird. Hier der `loop`-Code:

```
void loop()
{
 blinkLED(firstLedPin,BLINK_SHORT, 5); // blinke 5 mal
```

```
 blinkLED(secondLedPin,BLINK_MEDIUM, 3); // blinke 3 mal
 blinkLED(thirdLedPin, BLINK_LONG);      // blinke 1 mal
}
```

Um diese Funktionalität in die Bibliothek aufzunehmen, fügen wir den Prototyp wie folgt in *blinkLED.h* ein:

```
/*
 * blinkLED.h
 * Header-Datei für BlinkLED-Bibliothek
 */
#include "Arduino.h"

// Konstanten für Blinkdauer
const int BLINK_SHORT = 250;
const int BLINK_MEDIUM = 500;
const int BLINK_LONG = 1000;

void blinkLED(int pin, int duration);

// Neue Funktion mit Zähler
void blinkLED(int pin, int duration, int repeats);
```

Fügen Sie die Funktion in *blinkLED.cpp* ein:

```
/*
 * blinkLED.cpp
 * einfache Bibliothek, die eine LED für die angegebene Dauer in Millisekunden blinken lässt
 */
#include "Arduino.h"  // Wprogram.h für ältere Arduino-Versionen
#include "blinkLED.h"

// LED am angebenen Pin für die angegebene Dauer in Millisekunden blinken lassen
void blinkLED(int pin, int duration)
{
 digitalWrite(pin, HIGH);   // LED an
 delay(duration);
 digitalWrite(pin, LOW);    // LED aus
 delay(duration);
}

/* Funktion mit Zähler */
void blinkLED(int pin, int duration, int repeats)
{
 while(repeats)
 {
 blinkLED(pin, duration);
 repeats = repeats -1;
 }
}
```

Sie können eine Datei namens *keywords.txt* anlegen, wenn Sie eine *Syntaxhervorhebung* wünschen (das Einfärben von Schlüsselwörtern beim Bearbeiten eines Sketches in der IDE). Das ist eine Textdatei, die den Namen des Schlüsselwortes und seinen Typ angibt – jeder Typ verwendet eine andere Farbe. Schlüsselwort und Typ müssen durch einen

Tabulator (kein Leerzeichen) getrennt sein. Speichern Sie zum Beispiel folgende Datei als *keywords.txt* im *blinkLED*-Ordner:

```
#######################################
# Methoden und Funktionen (KEYWORD2)
#######################################
blinkLED KEYWORD2
#######################################
# Konstanten (LITERAL1)
#######################################
BLINK_SHORT  LITERAL1
BLINK_MEDIUM  LITERAL1
BLINK_LONG   LITERAL1
```

Sie müssen die IDE beenden und neu starten, wenn Sie eine neue Bibliothek anlegen oder die *keywords.txt*-Datei modifizieren. Sie müssen sie nicht neu starten, wenn Sie Code- (*.c* oder *.cpp*) oder Header-Dateien (*.h*) bearbeiten.

Siehe auch

Weitere Beispiele zur Entwicklung einer Bibliothek finden Sie in Rezept 16.5.

»Writing a Library for Arduino«: *http://www.arduino.cc/en/Hacking/LibraryTutorial*

Beachten Sie auch folgende Bücher zu C++:

- *Practical C++ Programming* von Steve Oualline (O'Reilly; Suchen Sie danach auf www.oreilly.de)
- *C++ Primer Plus* von Stephen Prata (Sams)
- *C++ Primer* von Stanley B. Lippman, Josée Lajoie und Barbara E. Moo (Addison-Wesley Professional)

16.5 Eine Bibliothek entwickeln, die andere Bibliotheken nutzt

Problem

Sie wollen eine Bibliothek entwickeln, die die Funktionalität einer oder mehrerer existierender Bibliothek(en) nutzt. Zum Beispiel wollen Sie die Wire-Bibliothek verwenden, um Daten von einem Wii Nunchuck-Controller abzurufen.

Lösung

Dieses Rezept nutzt die in Rezept 13.2 beschriebenen Funktionen, um über die Wire-Bibliothek mit einem Wii Nunchuck zu kommunizieren.

Legen Sie einen Ordner namens *Nunchuck* im Bibliotheksverzeichnis an (Details zur Dateistruktur einer Bibliothek finden Sie in Rezept 16.4). Legen Sie eine Datei namens *Nunchuck.h* an, die den folgenden Code enthält:

```
/*
 * Nunchuck.h
 * Arduino-Bibliothek für Wii Nunchuck
 */

#ifndef Nunchuck_included
#define Nunchuck_included

// Identitäten für alle vom Wii Nunchuck zurückgelieferten Felder
enum nunchuckItems { wii_joyX, wii_joyY, wii_accelX, wii_accelY, wii_accelZ,
            wii_btnC, wii_btnZ, wii_ItemCount };

// Pins neben I2C als Spannung und Masse für Nunchuck verwenden
void nunchuckSetPowerpins();

// I2C-Interface für Nunchuck initialisieren
void nunchuckInit();

// Daten vom Nunchuck anfordern
void nunchuckRequest();

// Daten vom Nunchuck abrufen,
// gibt true bei Erfolg zurück, sonst false
bool nunchuckRead();

// Daten in ein Format umwandeln, das die meisten wiimote-Treiber erwarten
char nunchuckDecode (uint8_t x);

// angeforderten Wert zurückliefern
int nunchuckGetValue(int item);

#endif
```

Legen Sie eine Datei namens *Nunchuck.cpp* im *Nunchuck*-Ordner mit folgendem Inhalt an:

```
/*
 * Nunchuck.cpp
 * Arduino-Bibliothek für wii Nunchuck
 */

#include "Arduino.h"  // Arduino

#include "Wire.h"    // Wire (I2C)
#include "Nunchuck.h" // Defines dieser Bibliothek

// Definitionen für Standard Arduino-Board (19 und 18 für Mega)
const int vccPin = 17; // +v und Masse über diese Pins
const int gndPin = 16;


const int dataLength = 6;      // Anzahl anzufordernder Bytes
static byte rawData[dataLength];  // Array für Nunchuck-Daten
```

```
// Pins neben I2C als Spannungsversorgung und Masse für Nunchuck verwenden
void nunchuckSetPowerpins()
{
 pinMode(gndPin, OUTPUT); // Versorgungspins einstellen
 pinMode(vccPin, OUTPUT);
 digitalWrite(gndPin, LOW);
 digitalWrite(vccPin, HIGH);
 delay(100); // Warten, dass sich die Spannungsversorgung stabilisiert
}

// I2C-Interface für Nunchuck initialisieren
void nunchuckInit()
{
 Wire.begin();          // I2C-Bus als Master betreten
 Wire.beginTransmission(0x52);// Übertragung an Gerät 0x52
 Wire.write((byte)0x40);    // Speicheradresse senden
 Wire.write((byte)0x00);    // Null senden.
 Wire.endTransmission();    // Übertragung beenden
}

// Daten vom Nunchuck anfordern
void nunchuckRequest()
{
 Wire.beginTransmission(0x52);// Übertragung an Gerät 0x52
 Wire.write((byte)0x00);// Ein Byte senden
 Wire.endTransmission();// Übertragung beenden
}

// Daten vom Nunchuck empfangen,
// gibt true bei Erfolg zurück, anderenfalls false
bool nunchuckRead()
{
 byte cnt=0;
 Wire.requestFrom (0x52, dataLength);// Daten vom Nunchuck anfordern
 while (Wire.available ()) {
  byte x = Wire.read();
  rawData[cnt] = nunchuckDecode(x);
  cnt++;
 }
 nunchuckRequest();  // Nächste Nutzdaten anfordern
 if (cnt >= dataLength)
  return true;  // Erfolg, wenn alle 6 Bytes empfangen wurden
 else
  return false; // Fehler
}

// Daten in ein Format umwandeln, das die meisten wiimote-Treiber erwarten
char nunchuckDecode (byte x)
{
 return (x ^ 0x17) + 0x17;
}

// Angeforderten Wert abrufen
int nunchuckGetValue(int item)
{
 if( item <= wii_accelZ)
  return (int)rawData[item];
```

```
 else if(item == wii_btnZ)
  return bitRead(rawData[5], 0) ? 0: 1;
 else if(item == wii_btnC)
  return bitRead(rawData[5], 1) ? 0: 1;
}
```

Schließen Sie den Nunchuck wie in Rezept 13.2 an, und nutzen Sie den folgenden Sketch, um die Bibliothek zu testen (wenn die IDE bereits läuft, während Sie diese beiden Dateien anlegen, beenden und starten Sie die IDE neu, damit die neue Bibliothek auftaucht):

```
/*
 * WiichuckSerial
 *
 * Nutzt Nunchuck-Bibliothek, um Sensorwerte an den seriellen Port zu senden
 */

#include <Wire.h>
#include "Nunchuck.h"

void setup()
{
 Serial.begin(9600);
 nunchuckSetPowerpins();
 nunchuckInit(); // Initialisierungs-Handshake
 nunchuckRead(); // Ersten Aufruf ignorieren
 delay(50);
}

void loop()
{
 nunchuckRead();
 Serial.print("H,");    // Header
 for(int i=0; i < 5; i++) // Werte des Beschleunigungsmessers und der Buttons ausgeben
 {
  Serial.print(nunchuckGetValue(wii_accelX+ i), DEC);
  Serial.write(',');
 }
 Serial.println();
 delay(20); // Pause in Millisekunden
}
```

Diskussion

Um eine andere Bibliothek einzubinden, verwenden Sie wie in einem normalen Sketch die `include`-Anweisung. Es ist sinnvoll, in Ihrer Dokumentation Informationen zu allen von Ihnen verwendeten zusätzlichen Bibliotheken aufzuführen, insbesondere dann, wenn eine Bibliothek benötigt wird, die nicht in der Standard-Distribution enthalten ist.

Der Hauptunterschied zwischen den Bibliotheks-Code um dem Sketch in Rezept 13.2 besteht im Einbinden von *Nunchuck.h*, das die Funktions-Prototypen enthält. (Arduino Sketch-Code erzeugt diese Prototypen stillschweigend für Sie, während Prototypen für Arduino-Bibliotheken explizit angelegt werden müssen).

Hier ein weiteres Beispiel für eine Bibliothek. Sie verwendet eine C++-Klasse, um die Bibliotheksfunktionen zu kapseln. Eine Klasse ist eine Programmiertechnik zur Gruppierung von Funktionen und Variablen, die von den meisten Arduino-Bibliotheken verwendet wird.

Die Bibliothek kann beim Debugging helfen, indem sie Daten über die Wire-Bibliothek an einen zweiten Arduino sendet. Das ist besonders nützlich, wenn der serielle Hardware-Port nicht verfügbar ist und ein Software-Port aufgrund der zeitlichen Abläufe nicht in Frage kommt. Hier wird die print-Funktionalität des Arduino-Kerns zum Aufbau einer neuen Bibliothek genutzt, die Ausgaben an I2C sendet. Die Verbindungen und der Code werden in Rezept 13.9 behandelt. Die folgende Beschreibung zeigt, wie man diesen Code in eine Bibliothek umwandelt.

Legen Sie einen Ordner namens *i2cDebug* im Bibliotheksverzeichis an (Details zur Dateistruktur einer Bibliothek finden Sie in Rezept 16.4). Legen Sie eine Datei namens *i2cDebug.h* mit dem folgenden Code an:

```
/*
 * i2cDebug.h
 */
#ifndef i2cDebug_included
#define i2cDebug_included

#include <Arduino.h>
#include <Print.h>  // Arduino print-Klasse

class i2cDebugClass : public Print
{
 private:
  int i2cAddress;
  byte count;
  size_t write(byte c);
 public:
  i2cDebugClass();
  boolean begin(int id);
};

extern i2cDebugClass i2cDebug;  // das I2C-Debug-Objekt
#endif
```

Erzeugen Sie die folgende Datei namens *i2cDebug.cpp* im *i2cDebug*-Ordner:

```
/*
 * i2cDebug.cpp
 */

#include <i2cDebug.h>

#include <Wire.h>  // Arduino I2C-Bibliothek


i2cDebugClass::i2cDebugClass()
{
}
```

```
boolean i2cDebugClass::begin(int id)
{
 i2cAddress = id;  // Slave-Adresse speichern
 Wire.begin(); // mit I2C-Bus verbinden (Adresse für Master optional)
 return true;
}

size_t i2cDebugClass::write(byte c)
{
 if( count == 0)
 {
  // Erstes Zeichen der Übertragung?
  Wire.beginTransmission(i2cAddress); // An Gerät senden
 }
 Wire.write(c);
 //  Daten senden, wenn I2C-Puffer voll oder Zeilenende erreicht
 // BUFFER_LENGTH ist in der Wire-Bibliothek definiert
 if(++count >= BUFFER_LENGTH || c == '\n')
 {
  // Sende Daten, wenn Puffer voll oder Newline
  Wire.endTransmission();
  count = 0;
 }
 return 1; // ein Zeichen geschrieben
}

i2cDebugClass i2cDebug;  // I2C-Debug-Objekt erzeugen
```

Die `write`-methode liefert `size_t` zurück. Dieser Wert ermöglicht es der print-Funktion, die Zahl der ausgegebnen Zeichen zurückzugeben. Das wurde it Arduino 1.0 eingeführt-davor haben `write` und `print` keine Werte zurückgegeben. Wenn ihre Bibliothek auf Stream oder Print basiert, müssen Sie den Rückgabetyp in `size_t` ändern.

Laden Sie den Beispiel-Sketch in die IDE:

```
/*
 * i2cDebug
 * Beispiel-Sketch für i2cDebug-Bibliothek
 */

#include <Wire.h>   // Arduino I2C-Bibliothek
#include <i2cDebug.h>

const int address = 4;   // Adresse des Kommunikationsgerätes
const int sensorPin = 0;  // Analoger Eingangspin des Sensors
int val;          // Variable für Sensorwert


void setup()
{
 Serial.begin(9600);
 i2cDebug.begin(address);
}
```

```
void loop()
{
 // Spannung am Poti einlesen (val liegt zwischen 0 und 1023)
 val = analogRead(sensorPin);
 Serial.println(val);
 i2cDebug.println(val);
}
```

Vergessen Sie nicht, die IDE neu zu starten, nachdem Sie den Bibliotheksordner angelegt haben. Weitere Details zur Entwicklung einer Bibliothek finden Sie in Rezept 16.4.

Laden Sie den I2C-Slave-Sketch auf ein anderes Arduino-Board hoch und verschalten Sie die Boards wie in Rezept 13.9 beschrieben. Die Ausgabe des Arduino-Boards mit Ihrer Bibliothek sollte auf dem zweiten Board zu sehen sein.

Falls Sie mit C++-Klassen nicht vertraut sind, bieten folgende Bücher eine gute Einführung:

- *Programming Interactivity* von Joshua Noble (O'Reilly; suche Sie danach www.oreilly.de)
- *C++ Primer* von Stanley B. Lippman, Josée Lajoie und Barbara E. Moo (Addison-Wesley Professional)

16.6 Bibliotheken von Drittanbietern an Arduino 1.0 anpassen

Problem

Sie wollen die Bibliothek eines Drittanbieters nutzen, die für Arduino-Releases vor 1.0 entwickelt wurde.

Lösung

Bei den meisten Bibliotheken müssen nur wenige Zeilen geändert werden, damit sie unter Arduino 1.0 laufen. Zum Beispiel muss einer oder mehrere dieser Includes:

```
#include "wiring.h"
#include "WProgram.h"
#include "WConstants.h"
#include "pins_arduino.h"
```

durch ein einzelnes Include ersetzt werden:

```
#include "Arduino.h"
```

Die Dateinamen können zwischen spitzen Klammern oder Anführungszeichen stehen.

Diskussion

Ältere Bibliotheken, die sich unter Arduino 1.0 nicht kompilieren lassen, erzeugen eine oder mehrere der folgenden Fehlermeldungen:

```
source file: error: wiring.h: No such file or directory
source file: error: WProgram.h: No such file or directory
source file: error: WConstants.h: No such file or directory
source file: error: pins_arduino.h: No such file or directory
```

»Source file« (Quelldatei) ist der vollständige Pfad auf die Bibliotheksdatei, die aktualisiert werden muss. Es wird eine Reihe weiterer Fehler geben, die aber daher rühren, dass die angegebenen Dateien in der 1.0-Release nicht enthalten sind. Diese Fehler sollten verschwinden, sobald Sie die alten Header-Dateien durch `Arduino.h` ersetzt haben. Die Definitionen in diesen Dateien sind nun in `Arduino.h` enthalten, d.h. die Lösung besteht darin, alle obigen Dateien durch ein einzelnes Include von `Arduino.h` zu ersetzen.

Soll Arduino 1.0 neben früheren Kompilaten genutzt werden, können Sie ein bedingtes include nutzen (siehe Rezept 16.6):

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
// Diese Dateinamen werden in der Originalversion der Bibliothek verwendet
#include "wiring.h"
#include "pins_arduino.h"
#endif
```

Siehe auch

Bibliotheken von Drittanbietern, die Serial, Ethernet und andere Funktionalitäten nutzen, deren Syntax sich bei Arduino 1.0 geändert hat, verlangen zusätzliche Anpassungen am Code. Details finden Sie in Anhang H und in den Kapiteln dieses Buches, die diese Funktionalität behandeln.

Index

Symbole

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Über den Autor

Michael Margolis ist Technologieexperte im Bereich Echtzeitsysteme mit dem Schwerpunkt Hardware- und Software-Entwicklung für die Umgebungs-Interaktion. Er hat über 30 Jahre Erfahrung auf Führungsebene bei Sony, Microsoft und Lucent/Bell Labs. Er hat Bibliotheken und Kernsoftware geschrieben, die in der Arduino 1.0-Distribution enthalten ist.

Kolophon

Das Tier auf dem Cover des *Arduino Kochbuchs* ist ein Spielzeug-Hase. Mechanisches Spielzeug wie dieser Hase werden über Federn, Getriebe, Riemenräder, Hebel oder andere einfache Maschinen bewegt, angetrieben durch mechanische Energie. Solche Spielsachen haben eine lange Geschichte. Antike Beispiele sind aus Griechenland, China und der arabischen Welt bekannt.

Die Herstellung mechanischen Spielzeugs florierte im frühen modernen Europa. Im späten 14. Jahrhundert demonstrierte der deutsche Erfinder Karel Grod fliegendes Aufzieh-Spielzeug. Prominente Wissenschaftler jener Tage, darunter Leonardo da Vinci, Descartes und Galileo Galilei, waren für ihr mechanisches Spielzeug berühmt. Da Vincis berühmter mechanischer Löwe, 1509 für Louis XII gebaut, ging zum König hoch und öffnete seinen Brustkasten, um eine Fleur-de-Lis zu offenbaren.

Die Kunst der Herstellung mechanischen Spielzeugs erreichte ihren Höhepunkt im späten 18. Jahrhundert mit den berühmten »Automaten« des Schweizer Uhrmachers Pierre Jaquet-Droz und seines Sohnes Henri-Louis. Die menschlichen Figuren konnten so lebensechte Dinge tun wie eine Feder in ein Tintenfass tauchen, ganze Sätze schreiben, zeichnen und vom Papier Radiergummi-Reste wegblasen. Im 19. Jahrhundert brachten europäische und amerikanische Unternehmen beliebtes Spielzeug zum Aufziehen heraus, das zu Sammlerstücken geworden ist.

Da die Herstellung dieses Originalspielzeuges mit seiner komplizierten Mechanik und aufwendigen Dekoration sehr zeit- und kostenintensiv war, blieb es Königshäusern oder der Unterhaltung Erwachsener vorbehalten. Erst seit dem späten 19. Jahrhundert, mit dem Aufkommen der Massenproduktion und billiger Materialien (Blech und später Kunststoff), wurde mechanisches Spielzeug auch zu Spielsachen für Kinder. Die günstigen, sich bewegenden Neuheiten waren über ein Jahrhundert sehr beliebt, bis sie von batteriebetriebenen Spielsachen abgelöst wurden.

Die Abbildung auf dem Cover stammt vom Dover Pictorial Archive. Die Schriftart auf dem Cover ist Adobe ITC Garamond. Als Textschrift wird Linotype Birka verwendet, als Überschriftenschrift Adobe Myriad Condensed und als Code-Schrift LucasFonts TheSansMonoCondensed.