Arduino
Das umfassende Handbuch

Kapitel 1
Arduino – was ist das?

Sie interessieren sich für den Arduino? In diesem Buch möchte ich Ihnen zeigen, warum der Arduino so viele Dinge verändert hat und wie Sie an dieser Entwicklung teilhaben und an ihr mitwirken können. Und um es vorwegzunehmen: Mit den neuen Arduinos wird es weitere Veränderungen geben. Es bleibt sehr spannend!

Der Name Arduino steht für ein weltweit führendes System an Open-Source-Hard- und -Software. Das Unternehmen Arduino bietet Software-Tools, Hardware-Plattformen und Dokumentationen, mit denen nahezu jeder mit Technologie kreativ umgehen kann.

Die Plattform Arduino ist ein beliebtes Tool zur Produktentwicklung für eine Vielzahl unterschiedlicher Anwendungen und das *Internet of Things* (IoT) sowie eines der erfolgreichsten Tools für die Ausbildung. Hunderttausende von Designern, Ingenieuren, Studenten, Entwicklern und Herstellern auf der ganzen Welt nutzen den Arduino, um Innovationen in den Bereichen Musik, Spiele, Spielzeug, Robotik, Smarthomes, Landwirtschaft, autonome Fahrzeuge und mehr zu entwickeln.

Soweit erst einmal eine verkürzte Erklärung dazu, wie der Hersteller selbst sein Arduino-System positioniert. Ich möchte diese Auflistung um die wichtigen Themen Mess-, Steuerungs- und Regelungstechnik, Kommunikation und Automatisierung ergänzen.

1.1 Arduino – etwas Hintergrund

In den Jahren 2001 bis 2006 war das *Ivrea Interaction Design Institute* in Italien aktiv. Dort wurde der Arduino als einfaches Werkzeug für schnelles Prototyping entwickelt. Er richtete sich an Studenten ohne Kenntnisse in Elektronik und Programmierung, die die physische Welt mit der digitalen Welt zu verbinden suchten.

Dieses grundlegende Anliegen der Mikrocontroller-Technologie, physikalische Größen in Daten umzusetzen, sie zu verarbeiten und die Ergebnisse dieser Verarbeitung wieder zurückfließen zu lassen, ist heute so gültig wie vor 15 oder mehr Jahren.

Sobald das im Jahr 2005 eingeführte Arduino-Board – so nennen wir die Leiterplatte, die den Arduino ausmacht – eine größere Community erreicht hatte, stellte sich die

Arduino-Mannschaft auf neue Anforderungen und Herausforderungen ein und differenzierte ihr Angebot von einfachen Boards auf der Basis von 8-Bit-Mikrocontrollern hin zu Produkten für IoT-Anwendungen, Wearables, 3D-Druck und Embedded Systems.

Alle Arduino-Boards sind vollständig *Open Source* und ermöglichen es den Benutzern, sie unabhängig voneinander zu erstellen und auch an ihre jeweiligen Bedürfnisse anzupassen. Auch die Software ist Open Source und wächst durch die Beiträge von Anwendern weltweit.

Arduino ist seitdem das beliebteste Prototyping-Tool für die Elektronik, das von einer breit gefächerten Nutzergemeinschaft – sowohl von Makern als auch von Ingenieuren und sogar großen Unternehmen – eingesetzt wird. Heute umfasst das Angebotsspektrum auf der Arduino-Website (*https://store.arduino.cc/collections/boards*) allein mehr als 34 unterschiedlich ausgestattete Arduino-Boards, die die Auswahl nicht immer einfach machen. Dazu komme ich später in Kapitel 2, »Arduino-Hardware«, noch detaillierter.

Der *Arduino Uno Rev3* ist nach wie vor das beste Board, um mit Elektronik und der Mikrocontroller-Programmierung zu beginnen. Der *Arduino Uno*, wie ich ihn hier kurz nenne, ist ein robustes Board, das übersichtlich und mit weitgehend konventionellen Bauteilen aufgebaut ist. Es verzeiht in der Regel auch kleinere Missgeschicke, die bei Inbetriebnahme und Test schon einmal vorkommen können.

Zudem ist der Arduino Uno das am häufigsten verwendete und am besten dokumentierte Board der gesamten Arduino-Familie. Ich komme gleich noch detaillierter darauf zurück.

In diesem einleitenden Kapitel sind zwei Begriffe gefallen, die für das Gesamtverständnis der Arduino-Thematik wichtig sind und hier deshalb noch gesondert betrachtet werden: *Open Source* und *Mikrocontroller*.

1.2 Open Source: Die Lizenzen des Arduino-Projekts

Als Open Source (wörtliche Übersetzung: *offene Quelle*) wurde ursprünglich Software bezeichnet, deren Quelltext offengelegt ist, der also unter bestimmten Bedingungen (Lizenzen) eingesehen, geändert, genutzt und weitergegeben werden kann.

Als Open-Source-Hardware oder *Open Hardware* wird demgegenüber eine Hardware bezeichnet, die nach freien Bauplänen von Dritten hergestellt werden kann.

Was heißt das für unseren Arduino?

Für die Hardware ist das recht einfach geregelt. Die Referenzdesigns für die Arduino-Boards finden Sie auf den jeweiligen Produktseiten. Sie sind unter einer *Creative Commons Attribution-ShareAlike License* (CC-BY-SA) lizenziert, sodass Sie sie verwen-

den und an Ihre eigenen Bedürfnisse anpassen können, ohne um Erlaubnis bitten oder eine Gebühr zahlen zu müssen.

Werden Anpassungen am Referenzdesign vorgenommen, dann müssen der Name des Lizenzgebers genannt und die Veränderungen unter der gleichen Lizenz weitergegeben werden. Kenntlich gemacht wird die Lizenz durch das folgende Symbol (siehe Abbildung 1.1).

Abbildung 1.1 Das Logo der Lizenz »CC-BY-SA«

Für den Arduino Uno sind die Dateien des Referenzdesigns auf der Webseite *https://store.arduino.cc/arduino-uno-rev3* unter DOCUMENTATION zu finden. Sie können dann verschiedene Dateien (Eagle, PDF, DXF) herunterladen.

Die im Archiv *UNO-TH_Rev3e-reference.zip* vorhandenen Dateien *UNO-TH_Rev3e-reference.sch* und *UNO-TH_Rev3e-reference.brd* sind Eagle-CAD-Daten, während in der Datei *ArduinoUno.dxf* die Abmessungen des Arduino-Uno-Boards abgelegt sind. Mit diesen Dateien kann ein Arduino Uno nachgebaut werden. Die Datei *UNO-TH_Rev3e_sch.pdf* enthält den Schaltplan, und das in der Datei *A000066-datasheet.pdf* mitgelieferte, 13-seitige Datenblatt bietet alle Informationen, die Sie zur Arbeit mit dem Arduino Uno Rev3 benötigen.

Hinweis

Arduino-Boards sind Open Hardware und können von Dritten nachgebaut werden. Es gibt also keinen Grund, nicht auch auf alternative Hardware zurückzugreifen.

Ob die alternative Hardware dann auch 100-prozentig kompatibel zum originalen Arduino-Board ist, wird sich erst im Nachhinein feststellen lassen. Für Neueinsteiger ist der Start mit einem originalen Arduino empfehlenswert, um genau diese Unsicherheit von vornherein auszuschließen.

Die Diskussionen im Internet zum Thema Arduino-Clones sind vielfältig und nicht immer zielführend. Dass Arduino-Clones a priori eine schlechtere Qualität aufweisen, ist inzwischen eine Legende. In den Anfangszeiten kann das aber durchaus so gewesen sein.

Der aktuelle Preis eines Arduinos, der im Allgemeinen höher ist als der eines Klons, hilft bei der Finanzierung der folgenden Aktivitäten des Arduino-Projekts:

- Entwicklung neuer Open-Source-Hardware
- Hardware-Dokumentation
- CE- und FCC-Zertifizierung

- Qualitätskontrolle
- Management der Arduino-Community
- Veröffentlichung von Tutorials
- Spenden an andere Open-Source-Projekte
- Hosting und Wartung der Arduino-Website und des Arduino-Forums mit Millionen von Nutzern

Auf der Software-Seite ist die Lizenzierung nicht ganz so einfach. Die Arduino-Software unterliegt der *GNU Public License* (GPL) bzw. der *Lesser GNU Public License* (LGPL).

Der Quellcode für die Arduino-Entwicklungsumgebung (*Arduino IDE*) wird von der GPL abgedeckt, die erfordert, dass alle Änderungen unter derselben Lizenz ausgeführt werden und der geänderte Quelltext offengelegt wird. Änderungen an der Arduino IDE werden Sie, zumindest vorerst, nicht vornehmen – Sie werden diese nur nutzen. In diesem Fall unterliegt die Nutzung keinen Einschränkungen.

Wenn Sie den Arduino-Core und die Bibliotheken für die Firmware eines kommerziellen Produkts verwenden, müssen Sie den Quellcode für die Firmware nicht freigeben. Die LGPL erfordert jedoch, dass Sie Objektdateien zur Verfügung stellen, die das erneute Verknüpfen der Firmware mit aktualisierten Versionen des Arduino-Kerns und der Bibliotheken ermöglichen. Alle Änderungen am Core und den Bibliotheken müssen unter der LGPL veröffentlicht werden (siehe »LGPL and Arduino in Commercial Products« unter *https://forum.arduino.cc/index.php?topic=240434.0*).

Schwieriger wird es dann bei Bibliotheken (*Librarys*), die nicht von Arduino selbst in das Arduino-Gesamtpaket integriert wurden. So sind derzeit (Stand: September 2022) in der *Arduino Library List* (*https://www.arduinolibraries.info*) Arduino-Librarys mit den folgenden Lizenzen zu finden:

- 0BSD (2)
- AGPL 3.0 (29)
- Apache 2.0 (219)
- Artistic 2.0 (6)
- BSD 2 Clause (27)
- BSD 3 Clause (138)
- BSD 3 Clause Clear (3)
- BSL 1.0 (2)
- CC BY SA 3.0 (1)
- CC BY SA 4.0 (5)
- CC0 1.0 (20)
- GPL 2.0 (74)
- GPL 3.0 (547)
- GPL 3.0 only (1)
- GPL 3.0 or later (2)
- ISC (4)
- LGPL 2.1 (166)
- LGPL 3.0 (202)
- MIT (2095)
- MPL 2.0 (11)
- MS PL (1)
- NOASSERTION (549)
- OSL 3.0 (1)
- Unlicense (30)
- WTFPL (3)

NOASSERTION bedeutet, dass keine gültigen Lizenzangaben gefunden wurden. Unlicense und WTFPL sind praktisch Public-Domain-Lizenzen und stellen keine weiteren Anforderungen.

Erklärungen zu den einzelnen Lizenzen finden Sie unter:

https://opensource.org/licenses/alphabetical

1.3 Maker und die Arduino-Community

In diesem Abschnitt erläutere ich Ihnen den Begriff des *Makers* und gehe auf die Arduino-Community ein.

Wenn Sie sich mit dem Arduino beschäftigen, werden Sie rasch von den Vorteilen profitieren, die Ihnen die Maker-Bewegung und die Community bieten. Lernen Sie sie also kennen, und engagieren Sie sich vielleicht selbst dort.

Gemäß der Definition laut Wikipedia.de bezeichnet der Begriff Maker eine Subkultur, die man auch als »Do-it-yourself-Kultur unter Nutzung aktueller Technik« beschreiben kann.

Es gibt übrigens auch eine alte Definition des Bastlers von Erich Kästner: »Ein Bastler ist jemand, der mit völlig unzureichenden Mitteln völlig überflüssige Dinge herstellt.«

Davon sind die Maker jedoch meilenweit entfernt: Maker sind die Entrepreneure der Stunde. Den Begriff des Bastlers werde ich deshalb hier nicht verwenden.

Die Maker-Bewegung bewegt sich vom ursprünglichen »Do it yourself« (DIY, dt. »Mach es selbst«) hin zum »Do it together« (DIT, dt. »Macht es gemeinsam«) als Modell für Gegenwart und Zukunft. Voraussetzung für die erfolgreiche Umsetzung von Projekten ist der Kontakt zu Gleichgesinnten. Zusätzliche Intelligenz und Erfahrung (auch aus anderen Wissensgebieten) sind herzlich willkommen.

Selbstvertrauen, Neugier, die Bereitschaft, sich einzubringen und im Team ein Projekt voranzubringen – das sind unabdingbare Voraussetzungen für einen Maker. Mit diesen Charakteristika und einigen technologischen Voraussetzungen sind die Triebkräfte und Eigenschaften der Maker-Bewegung bereits umrissen.

Die Maker-Bewegung profitiert außer von den geschilderten Fähigkeiten von ihrer heterogenen Zusammensetzung und den verfügbaren Tools zur Umsetzung ihrer Produktideen. Beide Einflussfaktoren bilden gleichsam die Basis für unkonventionelle Konzepte und Produkte.

Überzeugen Sie sich auf einer der zahlreichen *Maker-Faires* (*https://maker-faire.de/*) von den interessanten Lösungsansätzen, die die Maker vorstellen. Im Beitrag »Maker – Startups – Unternehmen«[1] habe ich das Zusammenspiel der verschiedenen Beteiligten beschrieben.

Bei den zur Verfügung stehenden Tools können wir unterschiedliche Einflussebenen erkennen, die in starkem Maße durch die Open-Source-Aspekte im weiteren Sinne geprägt werden. Die Offenlegung der Quellcodes von Software begann in der Hacker- und Hobbyisten-Bewegung und ist zentraler Bestandteil der Open-Source-Initiative.

Neue Projekte lassen sich auf einer breiten Palette von vorhandenen Software-Lösungen aufsetzen, sodass der Projektumfang bei sinkendem Entwicklungsrisiko wesentlich erweitert werden kann. Für Maker steht ein umfangreiches Portfolio an Software zur Verfügung. Neben den klassischen Tools für die Bearbeitung von Texten, Bildern, Videos und Musik stehen die Design-Tools im Vordergrund. Für die Software-Entwicklung stehen leistungsfähige Compiler und Skript-Interpreter ebenfalls als Open Source zur Verfügung.

Zur Software-Entwicklung für die bei Makern beliebten Arduino-Boards wird die Arduino IDE eingesetzt, die die Programmierung der Arduino-Boards in C++ ermöglicht (*https://www.arduino.cc*).

Eine weitere Klasse von Software-Tools sind die CAD- und CAM-Tools, die den Designprozess für Leiterplatten und mechanische Konstruktionen unterstützen. Die FH Potsdam initiierte das PCB-Designtool *Fritzing* (*https://fritzing.org/*), das alle notwendigen Schritte unterstützt – vom Zeichnen von Schaltplänen über das Visualisieren eines Breadboards (Steckbretts) bis zur Erstellung der Fertigungsdaten für die Leiterplatte. In der *FritzingFab* kann dann der Prototyp auch gleich bestellt und in Deutschland innerhalb Tagesfrist auch geliefert werden.

Mechanische Konstruktionen lassen sich mithilfe von *FreeCAD* (*https://www.freecadweb.org/*), *OpenSCAD* (*http://www.openscad.org/*) und anderen Tools erstellen und direkt in die noch zu betrachtenden Manufacturing-Lösungen überführen.

Der Grundgedanke der Open-Source-Software-Entwicklung wurde auch auf die Hardware übertragen: Die Hardware-Grundlagen, wie Layoutdaten für Leiterplatten, Konstruktionsunterlagen oder 3D-druckbare CAD-Dateien, werden offengelegt und zur weiteren Verwendung zur Verfügung gestellt. So können beispielsweise über die Plattform *Thingiverse.com* (*https://www.thingiverse.com*) 3D-Drucker, Laser-Cutter,

1 Kühnel, C.: »Maker – Startups – Unternehmen. Design & Elektronik«, 1/2016 (Teil 1), 2/2016 (Teil 2).
Online-Version von Teil 1: *http://www.elektroniknet.de/embedded/hardware/artikel/127588/*
Online-Version von Teil 2: *http://www.elektroniknet.de/embedded/hardware/artikel/130181/*

CNC-Fräsen und andere Maschinen mit den dort zur Verfügung gestellten Dateien zur Bearbeitung von Werkstücken benutzt werden. Besonders bekannt wurde die Seite Thingiverse.com durch Web-Communitys, die sich um die 3D-Drucker-Projekte *RepRap* und *MakerBot* versammeln.

Mit den geschilderten Werkzeugen stehen auch dem Maker heute Hilfsmittel zur Verfügung, mit denen er eine Idee recht schnell auf ihre Umsetzbarkeit hin überprüfen kann. Dieser als *Proof of Concept* bezeichnete Entwicklungsschritt wird nicht nur von den Makern, sondern auch im industriellen Umfeld genutzt. Hier spielen die erreichbare Einsparung von Entwicklungszeit (*Time-to-Market*) und die Reduzierung des Entwicklungsrisikos die entscheidende Rolle.

Die Maker-Bewegung ist gemäß diesen Schilderungen also eher ein interdisziplinärer Zusammenschluss Gleichgesinnter, während die Arduino-Community diesen Zusammenschluss auf Arduino-relevante Themen bezieht. Die Grundsätze sind dabei aber vergleichbar.

Auf der Arduino-Website *https://www.arduino.cc/* gibt es Foren und Blogs, die dem Erfahrungsaustausch dienen und bei Problemen Hilfestellung leisten.

Im *Arduino Blog* werden die Informationen nach bestimmten Themen zusammengefasst. Durch Auswahl von BOARDS, CATEGORIES oder ARCHIVE bündeln Sie die Informationen. Wenn Sie unter BOARDS den ARDUINO UNO auswählen, bekommen Sie die spezifischen Informationen zusammengestellt angezeigt.

Die Arduino-Website ist ein guter Startpunkt, um mit der Arduino-Community in Kontakt zu treten.

Auch in den sozialen Medien bildet sich die Arduino-Community ab. Sie finden verschiedene Arduino-Gruppen auf Facebook und eine auf MeWe.

1.4 Arduino Uno Rev3 – der Standard

Arduino Uno ist das Arduino-Board, dessen Kern der von *Atmel* entwickelte Mikrocontroller *ATmega328P* ist. Auch nach der Übernahme von Atmel durch *Microchip* wird dieser verbreitete Controller unverändert gefertigt. Auf dem Board ist alles vorhanden, was zur Unterstützung des Mikrocontrollers benötigt wird. Abbildung 1.2 zeigt einen Arduino Uno Rev3.

Die spezielle Form des Arduino Uno Rev3, der Arduino-Uno-Formfaktor, ist praktisch standardisiert, sodass andere Boards, die sogenannten *Arduino-Shields*, egal von welchem Hersteller sie stammen, über die Arduino-Buchsenleisten kontaktiert werden können.

Abbildung 1.2 Arduino Uno Rev3

1.4.1 Ein- und Ausgangspins

Wie Sie in Abbildung 1.2 sehen können, verfügt der Arduino Uno über 20 digitale Eingangs- bzw. Ausgangspins, sechs analoge Eingänge, einen 16-MHz-Quarz, einen USB-Anschluss, eine Netzbuchse, einen ICSP-Header für das *In-Circuit Serial Programming* (ICSP) und eine Reset-Taste.

ICSP ist eine der verschiedenen Methoden zur Programmierung von Arduino-Boards. Normalerweise wird ein Arduino-Bootloader-Programm zum Programmieren eines Arduinos verwendet, der einen seriellen Programm-Upload ermöglicht. Wenn der Bootloader jedoch fehlt oder beschädigt ist, kann stattdessen ICSP verwendet werden.

Die nach außen hin verfügbaren Anschlüsse sind den beiden Buchsenleisten (Header) zugeordnet ❶. An der linken Seite des Arduino Uno sind oben eine USB-Buchse vom Typ B (*USB Jack*) ❷ und darunter die Buchse zur Spannungsversorgung (*Power Jack*) ❸ angeordnet.

Die Pinbelegung der nach außen führenden Buchsenleisten entnehmen Sie Abbildung 1.3. Deutlich zu sehen ist, dass jedem Anschluss mehrere Funktionen zugeordnet sind. Welche Funktion wirksam wird, bestimmt die vor dem Programmstart vorzunehmende Initialisierung.

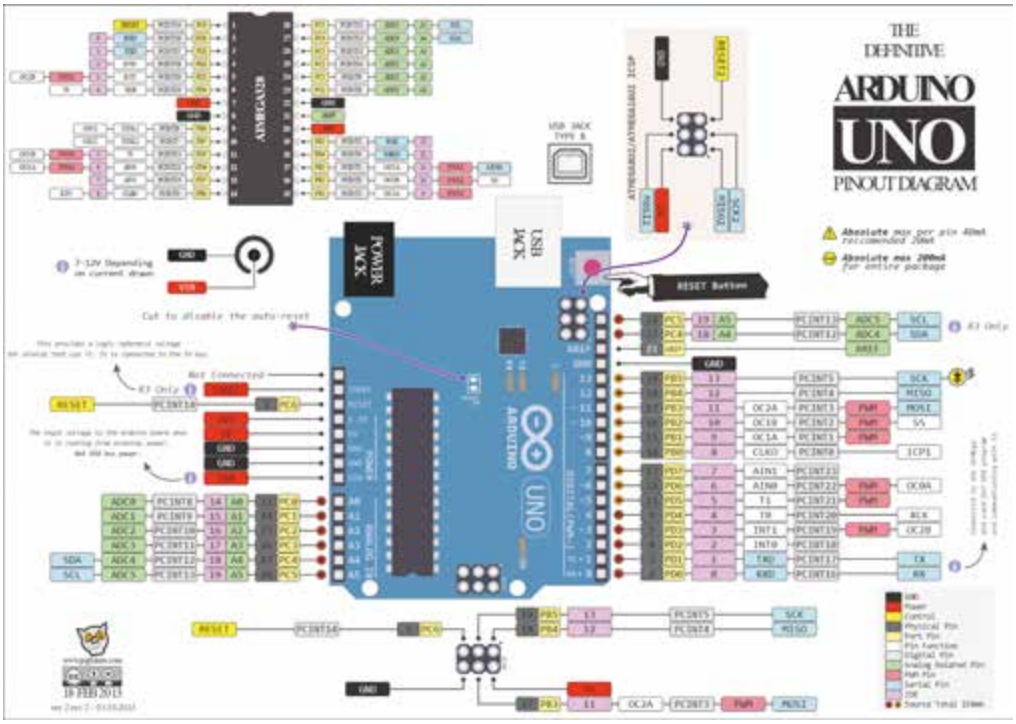

Abbildung 1.3 Pinbelegung beim Arduino Uno Rev3

Von den insgesamt 20 digitalen I/O-Pins können sechs als PWM-Ausgang und sechs als analoger Eingang fungieren. *PWM* bezeichnet die Pulsweitenmodulation (*Pulse Width Modulation*), bei der die Information im Tastverhältnis (*Duty Cycle*) gemäß Abbildung 1.4 liegt.

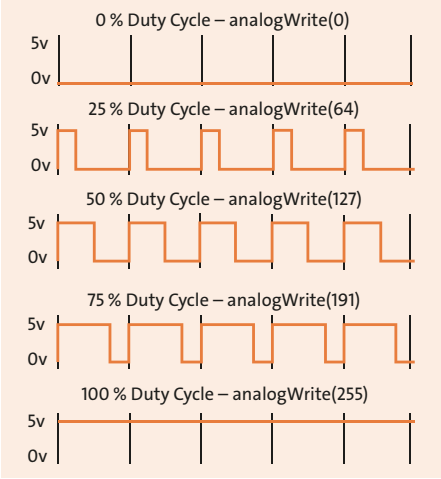

Abbildung 1.4 PWM

PWM-Signale lassen sich sehr gut für die Helligkeitssteuerung von LEDs oder für die Steuerung der Drehzahl von DC-Motoren verwenden. Mit `analogWrite(value)` steht in Abbildung 1.4 auch bereits die zugehörige Anweisung zur Ausgabe – was es damit genau auf sich hat, erfahren Sie in Abschnitt 4.6.2.

1.4.2 Serielle Schnittstellen

Die Pinbelegung zeigt drei serielle Schnittstellen, die für den Einsatz des Arduino sehr wichtig sind:

Pins	Bedeutung
TX, RX	Serielle Schnittstelle für Programm-Upload und Kommunikation (UART)
SCL, SDA	I^2C-Bus
SCK, MISO, MOSI	SPI-Bus

Tabelle 1.1 Arduino Uno – serielle Schnittstellen

Mit diesen Kommunikationsschnittstellen werden Sie immer wieder in Kontakt kommen, denn neben den analogen und digitalen Ein-/Ausgängen stellen diese Schnittstellen Verbindungen zu anderen Komponenten oder Controllern her und ermöglichen erst den für Mikrocontroller-Anwendungen so wichtigen Datenaustausch.

1.4.3 Spannungsversorgung

Den Arduino Uno können Sie auf unterschiedlichen Wegen mit Spannung versorgen. Die Spannungsversorgung kann am einfachsten über den USB-Anschluss erfolgen, der ohnehin für den Programm-Upload benötigt wird. Bedenken Sie allerdings, dass die USB-2.0-Ports, die heute noch in vielen PCs verbaut werden, gemäß Spezifikation nur einen Strom von maximal 500 mA liefern. Der USB-Anschluss kann also den Strombedarf in vielen Fällen nicht decken.

Spannungsversorgung über USB

Die Spannungsversorgung über den USB-Anschluss eines Rechners kann nicht immer zuverlässig die benötigte Spannung bereitstellen. Zum Ausprobieren ist es ein guter Anfang, aber wenn Sie den Arduino mit Zusatzschaltungen erweitern, brauchen Sie ein richtiges Netzteil.

Bei der Spannungsversorgung über den *Power Jack* (Hohlstecker) sollte das eingesetzte Steckernetzteil eine Gleichspannung zwischen 7 und 12 V liefern. Die Stromaufnahme des Arduino Uno hängt stark von den angeschlossenen Lasten ab und kann ohne diese nicht zuverlässig angegeben werden.

Ein Steckernetzteil, das in der Lage ist, einen Strom von 1 A zu liefern, ist aber erst einmal ein sicherer Anfang. Reichelt bietet ein solches Netzteil unter der Bezeichnung HNP 12-120L6 für unter 10 € an (*https://cdn-reichelt.de/documents/datenblatt/D400/HNP12.pdf*).

Die Spannungszuführung kann aber auch über den Anschluss VIN erfolgen.

Wenn Sie den Arduino Uno über ein USB-Kabel mit einem Computer verbinden und die Spannungsversorgung über ein Netzteil oder einen Akku vornehmen, dann sind Sie bereits für das Abenteuer Arduino gerüstet.

1.4.4 Mikrocontroller ATmega328P

Der Mikrocontroller ATmega328P ist in einem klassischen DIL-Gehäuse (*Dual-in-Line*) auf dem Board gesockelt und kann daher ersetzt werden, wenn wirklich einmal alles schiefgegangen ist.

Für einen Austausch 1:1 müssen Sie einen ATmega328P mit programmiertem Bootloader[2] verwenden. Den bekommen Sie (eigentlich) für 6,10 € bei Reichelt. Der ATmega328P ohne Bootloader[3] kostet bei Reichelt nur 3,45 €.

In der aktuellen Situation müssen Sie allerdings unerfreuliche Lieferfristen berücksichtigen. Der ATmega328P mit programmiertem Bootloader war bei Reichelt nicht lieferbar, beim ATmega328P ohne Bootloader wurde im September 2022 als voraussichtlicher Liefertermin der 11.01.2023 angegeben. Hoffen wir, dass sich Versorgungslage bessert, bis Sie dieses Buch in den Händen halten.

Wie Sie später noch sehen werden, gibt es auch einen Arduino Uno mit einem ATmega328P im SMD-Gehäuse. Da ist ein Wechsel aber nicht so einfach.

1.4.5 Warum eigentlich die Bezeichnung »Uno«?

Nachdem Sie einen ersten Eindruck von der Hardware des Arduino Uno gewinnen konnten und auch schon angedeutet wurde, dass es weitere Arduinos gibt, noch ein Wort zur Namensgebung.

2 *https://www.reichelt.de/arduino-atmega328-mit-arduino-bootloader-ard-atmega-328-p230602.html*

3 *https://www.reichelt.de/8-bit-atmega-avr-mikrocontroller-32-kb-20-mhz-pdip-28-atmega-328p-pu-p119685.html*

Vor dem Arduino Uno gab es bereits andere Arduinos. Mit dem Uno (»uno« bedeutet auf Italienisch »eins«) wurde die Bereitstellung der Arduino Entwicklungsumgebung 1.0 (*Arduino IDE 1.0*) markiert. Arduino Uno und die Version 1.0 der Arduino IDE sind also die Referenzversionen von Arduino, die im Laufe der Jahre zu neueren Versionen weiterentwickelt wurden.

Heute existiert eine breite Palette von Arduino-Boards, von denen ich Ihnen wegen ihrer unterschiedlichen Ausstattungsmerkmale und Performance noch einige vorstellen werde.

Die jetzt als Legacy Arduino bezeichnete Arduino IDE lag im September 2022 in Version 1.8.19 für alle gängigen Betriebssysteme vor.

Der Arduino Uno bleibt damit immer noch die Hardware-Basis des Arduino-Universums und ist mit dem ATmega328P eine ausgezeichnete Ausgangsbasis für die Umsetzung von Programmideen in laufende Mikrocontroller-Anwendungen.

Wenn Sie bei der Arbeit mit dem Arduino Uno eine gewisse Sicherheit gewonnen haben, dann ist der Schritt hin zu komplexeren Arduinos wesentlich einfacher.

Keine Angst beim Umgang mit dem Arduino Uno!

Der Arduino Uno ist ein robustes Board, weshalb Sie keine Angst beim Ausprobieren von Neuem und bislang Unbekanntem aufkommen lassen sollten.

1.5 Details zum Mikrocontroller

Um den Arduino besser verstehen zu können, will ich Ihnen einige Grundlagen zum Mikrocontroller allgemein erläutern. Wenn Ihnen das an dieser Stelle zu theoretisch erscheint, dann können Sie diesen Abschnitt auch überblättern – Sie können auch wunderbar mit dem Arduino arbeiten, ohne diese fortgeschrittenen Informationen zu kennen. Vielleicht kommen Sie später doch noch hierher zurück, um das eine oder andere zu vertiefen.

Den Arduino Uno hatte ich als Hardware-Basis des Arduino-Universums bezeichnet. Die folgenden Ausführungen beziehen sich deshalb auch auf den im Arduino Uno eingesetzten Mikrocontroller ATmega328P von *Atmel* bzw. heute *Microchip*.

Ich werde in späteren Kapiteln noch Arduino-kompatible Mikrocontroller auf der Basis anderer Architekturen vorstellen – die grundlegenden Aussagen gelten aber unabhängig davon.

1.5.1 Mikrocontroller-Kern

Abbildung 1.5 zeigt das Blockdiagramm des ATmega328P, entnommen aus dem immerhin 662 Seiten umfassenden Datenblatt der Reihe ATmega48/88/168/328.

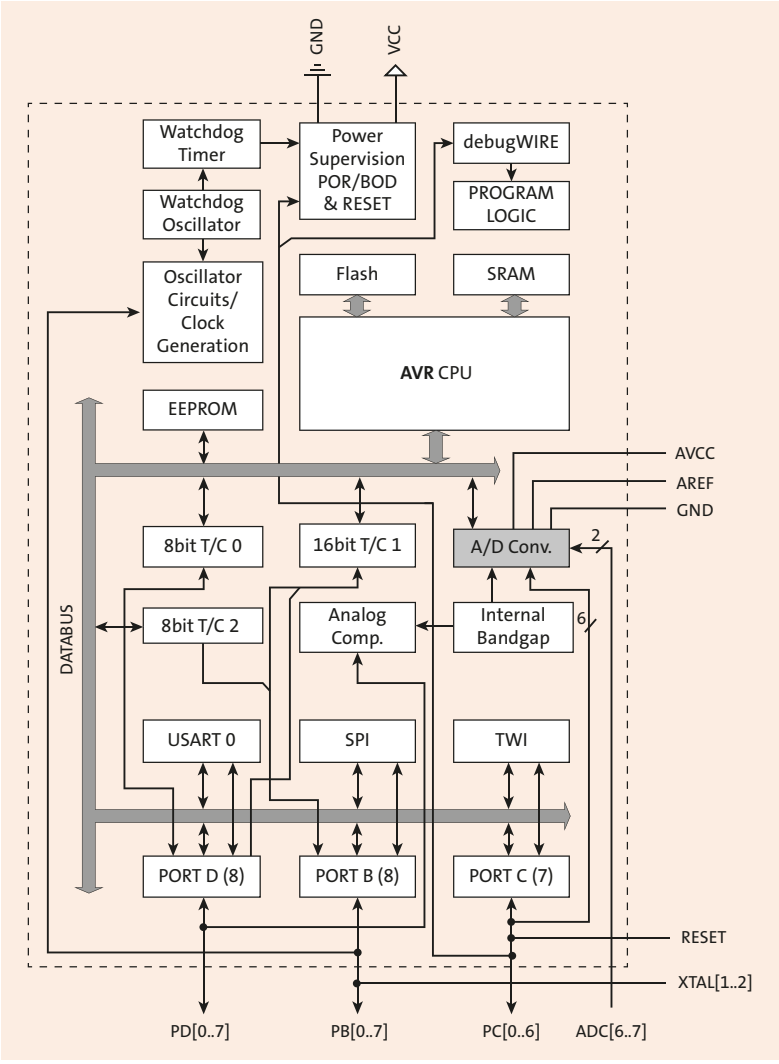

Abbildung 1.5 ATmega328P-Blockdiagramm (Quelle: ATmega328 Data Sheet)

Das *ATmega328P Data Sheet* können Sie von der URL *http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf* herunterladen.

In der oberen Hälfte des Blockdiagramms sind die eigentliche AVR-CPU (*Central Processing Unit*) sowie Programm- und Datenspeicher angeordnet.

Die CPU ist das zentrale Rechen- und Steuerelement eines jeden Mikrocontrollers oder Rechners allgemein.

Der Programmspeicher ist als Flash-Memory ausgebildet und kann garantiert bis zu 100.000-mal programmiert werden. Flash-Memory ist eine Variante des ROM (*Read-Only Memory*, dt. Nur-Lese-Speicher), eines nichtflüchtigen Speichers, dessen Inhalt nach der Programmierung nur noch gelesen werden kann.

Getrennte Programm- und Datenspeicher kennzeichnen die sogenannte *Harvard-Architektur*, die den gleichzeitigen Zugriff auf beide Speicher erlaubt. Des Weiteren besteht die Möglichkeit, die Speicher mit unterschiedlicher Bitbreite auszulegen. Beim ATmega328P sind die Instruktionen 16 und 32 Bit breit, weshalb der Programmspeicher mit einer Breite von 16 Bit angelegt wurde. Mit 16 K (16.384) Zellen Programmspeicher umfasst er dann 32 KByte Flash-Memory.

Für die Assembler-Instruktion `ADD` (»Addiere zwei Registerinhalte«) ist im Folgenden die Befehlsdecodierung im Programmspeicher beispielhaft gezeigt. Die Inhalte der beiden Register `r` und `d` werden in einem Zyklus addiert, und das Ergebnis wird im Register `d` abgelegt. Mit den Bits `rrrrr` und `ddddd` wird jeweils eines der Register 0 bis 31 adressiert.

15	14	13	12	11	10	9	8	**7**	6	5	4	3	2	1	0	**Bit**
0	0	0	0	1	1	r	d	d	d	d	d	r	r	r	r	**ADD**

Der Datenspeicher umfasst eine Breite von 8 Bit. Es kann also genau ein Byte pro Speicherplatz abgelegt werden. Für das Datenbyte `0xA5` (=10100101) sehen Sie das hier:

7	6	5	4	3	2	1	0	**Bit**
1	0	1	0	0	1	0	1	**0xA5**

Hinweis

In der Folge werden Zahlen häufig als Hexadezimalzahlen in der Form `0xA5` (= 165 dezimal) angegeben. Wenn Ihnen diese Art der Darstellung fremd ist, dann finden Sie bei den Materialien zum Buch eine hilfreiche Übersicht.

Der als statisches RAM des Speichers (*Random Access Memory*, dt. *Speicher mit wahlfreiem Zugriff zum Lesen und Schreiben*) ausgelegte Datenspeicher umfasst das *Register File*, Ein-/Ausgabe-Register (I/O-Register) und das interne SRAM zur Datenspeicherung mit der in Tabelle 1.2 gezeigten Aufteilung des Speichers (*Memory Map*).

Speicher	Adressbereich
32 Register	0x0000–0x001F
64 I/O-Register	0x0020–0x005F
160 Extended I/O-Register	0x0060–0x00FF
Internes SRAM (2048 × 8 Bit)	0x0100–0x08FF

Tabelle 1.2 Speicheraufteilung des SRAM beim ATmega328P

Aus Gründen der Softwaresicherheit ist der Programmspeicher in zwei Teile aufgeteilt. Die *Boot Loader Section* beinhaltet den sogenannten *Bootloader*, der den Programm-Upload (das ist das Programmieren des Flash-Memorys über die serielle Schnittstelle) ermöglicht.

Die *Application Program Section*, der eigentliche Programmspeicher, enthält dann das über den Bootloader hochgeladene Programm. Das ist weniger komplex, als es hier vorerst erscheinen mag. Dennoch ist es wichtig, dass Sie die beiden Speicherbereiche und deren unterschiedliche Funktion verstanden haben.

Beim Arduino Uno ist der eingesetzte ATmega328P bereits mit dem Bootloader ausgestattet. Sollten Sie den ATmega328P mal ersetzen müssen, dann können Sie beispielsweise einen mit dem Bootloader vorprogrammierten ATmega328P von einem Distributor beziehen oder Sie programmieren den Bootloader selbst in einen »nackten« ATmega328P.

Hinweise, wie Sie hier vorgehen müssen, können Sie unter *https://netzmafia.ee.hm.edu/skripten/hardware/Arduino/Bootloader_Flashen/index.html* nachlesen. Eine gewisse Erfahrung beim Umgang mit einem Arduino ist hierzu aber Voraussetzung, weshalb ich zumindest anfangs die wenigen Euro für den bereits programmierten Bootloader ausgeben würde.

Zurück zum ATmega328P-Blockdiagramm: Zunächst möchte ich Ihnen die Funktion der AVR-CPU näher erläutern. Abbildung 1.6 zeigt die Architektur des AVR-RISC-Mikrocontrollers wiederum anhand eines Blockdiagramms.

Die Hauptfunktion der AVR-CPU besteht darin, die korrekte Programmausführung sicherzustellen. Die CPU muss daher in der Lage sein, auf Programm- und Datenspeicher zuzugreifen, Berechnungen durchzuführen, Peripheriegeräte zu steuern und Interrupts (das sind Unterbrechungen des laufenden Programms) zu bearbeiten.

Anweisungen im Programmspeicher führt die AVR-CPU im sogenannten *Single Level Pipelining* aus – d. h., während eine Anweisung ausgeführt wird, wird die nächste Anweisung bereits aus dem Programmspeicher abgerufen. Dieses Konzept ermöglicht die Ausführung von Anweisungen in jedem Taktzyklus.

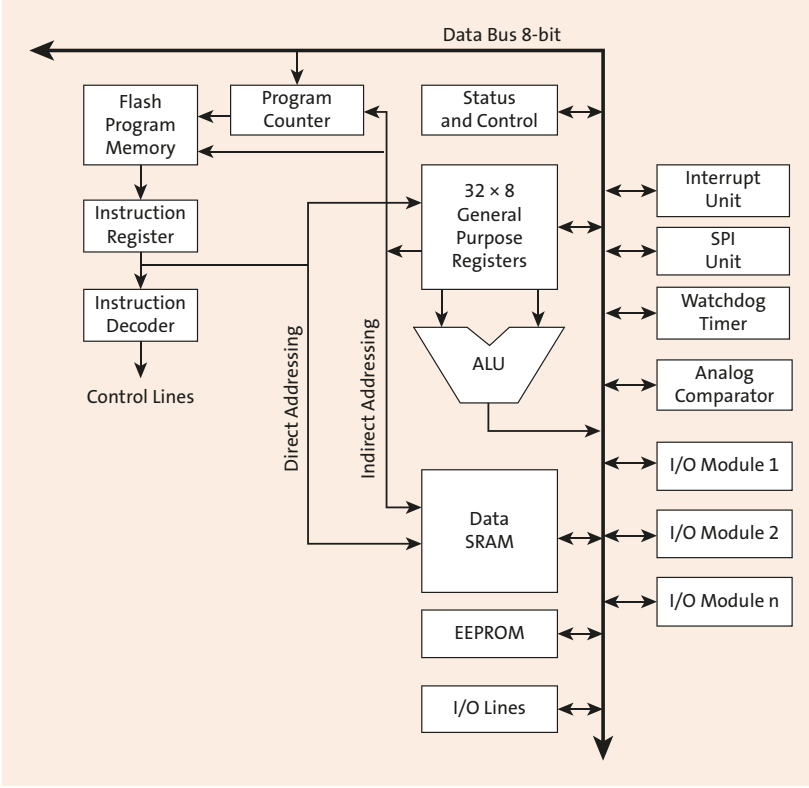

Abbildung 1.6 AVR-Architektur (Quelle: ATmega328P Data Sheet)

Das Pipelining ist heute in allen modernen Prozessoren meist sogar mehrstufig vorzufinden (siehe Abbildung 1.7).

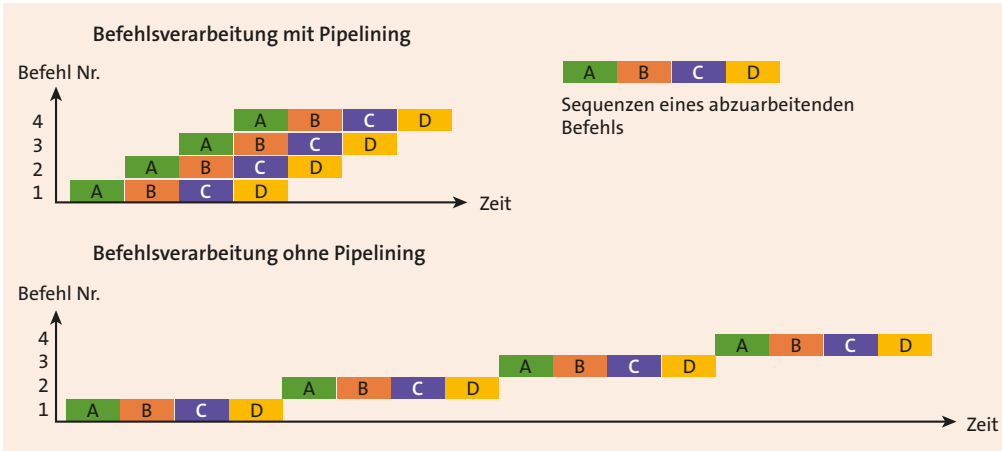

Abbildung 1.7 Das Prinzip des Pipelinings (Quelle: https://de.wikipedia.org/wiki/Pipeline_(Prozessor))

Dies ermöglicht den Betrieb der ALU (*Arithmetic Logic Unit*) in einem Taktzyklus der CPU (siehe Abbildung 1.8). Bei einer typischen ALU-Operation werden zwei Operanden aus dem *Register File* geladen, die Operation ausgeführt und das Ergebnis ins *Register File* zurückgeschrieben. Am Beispiel der `ADD`-Instruktion hatte ich die Adressierung der Register bereits gezeigt.

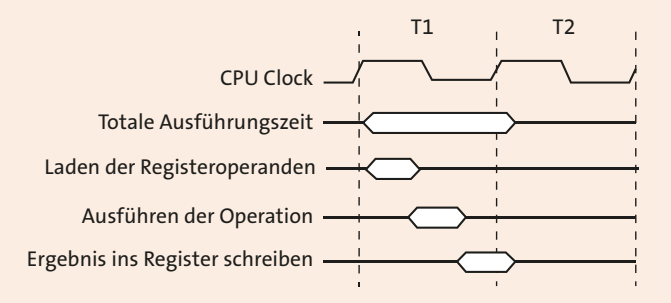

Abbildung 1.8 Single-Cycle-ALU-Operation

Die ALU unterstützt arithmetische und logische Operationen zwischen Registern oder zwischen einer Konstanten und einem Register. Einzelregisteroperationen können auch in der ALU ausgeführt werden.

Nach einer arithmetischen Operation ist das *Statusregister* aktualisiert und beinhaltet Informationen über das Ergebnis der Operation.

Bedingte und unbedingte Sprung- und Aufrufbefehle steuern den Programmablauf. Diese Befehle sind in der Lage, direkt auf den ganzen Adressraum zuzugreifen.

Während Interrupts und Unterprogrammaufrufen wird die jeweilige Rücksprungadresse auf dem *Stack* gespeichert. Der Stack ist ein Stapelspeicher nach dem FILO-Prinzip (*First In Last Out*) – d. h., der zuerst abgelegte Inhalt steht erst nach dem Zurücklesen aller nachträglich abgelegten Inhalte wieder zur Verfügung. Daher auch der Name *Stapel*.

Der Stack wird im statischen RAM (*SRAM*) eingerichtet und ist folglich nur durch die Gesamtgröße des SRAM und dessen Verwendung durch das Programm begrenzt.

Der I/O-Speicherbereich enthält 64 Adressen für CPU-Peripheriefunktionen wie Steuerregister, SPI und andere I/O-Funktionen. Der ATmega328P verfügt darüber hinaus noch über 160 Extended I/O-Register.

1.5.2 Mikrocontroller-Peripherie

Wie aus Abbildung 1.5 und Abbildung 1.6 bereits ersichtlich war, beinhaltet der ATmega328P zahlreiche Peripheriefunktionen, die die Leistungsfähigkeit und Flexibilität dieses Mikrocontrollers gleichermaßen bestimmen.

Das in der Folge vorgestellte Interrupt-System ist spezifisch für die AVR-Mikrocontroller. Die anderen Module hingegen haben funktional durchaus eher allgemeingültigen Charakter. Ihre Implementierung in anderen Mikrocontrollern wird sich natürlich unterscheiden.

Interrupt-System

Der ATmega328P kann eine Vielzahl unterschiedlicher Interrupts verarbeiten. *Interrupts* sind Unterbrechungen des laufenden Programms und dienen der speziellen Behandlung von Ereignissen (*Events*), die einer unmittelbaren Bearbeitung bedürfen.

Diese Interrupts sowie der Reset haben einen sogenannten *Interruptvektor* im Programmspeicher.

Jeder Interrupt kann individuell freigegeben werden. Die Interruptvektoren sind in Form einer Tabelle am unteren Ende des Programmspeichers angeordnet (siehe Tabelle 1.3). Die Priorität des jeweiligen Interrupts ist mit der Position in dieser Tabelle verknüpft. Je niedriger die Adresse des Interruptvektors ist, desto höher ist die Priorität. *Reset* hat also die höchste Priorität.

Wird ein Interrupt angefordert (*Interrupt Request*), dann wird das *Global Interrupt Enable Bit* zurückgesetzt. Alle (weiteren) Interrupts sind damit gesperrt (*disabled*). Das Anwendungsprogramm kann diese Sperre aufheben. Automatisch aufgehoben wird diese Sperre beim Verlassen der betreffenden *Interrupt-Serviceroutine* (ISR) durch ein *Return from Interrupt* (`reti`), wodurch ein Rücksprung ins Hauptprogramm erfolgt.

Es gibt grundsätzlich zwei unterschiedliche Typen von Interrupts. Der erste Typ wird durch ein Ereignis getriggert, das das betreffende Interrupt-Flag setzt. Der Programmzähler wird mit der Adresse des betreffenden Interruptvektors geladen, und die zugehörige Interrupt-Serviceroutine wird abgearbeitet. Tritt eine Interrupt-Anforderung auf, wenn der betreffende Interrupt nicht freigegeben ist, dann wird diese Interrupt-Anforderung gespeichert und erst nach Freigabe des Interrupts bearbeitet. Das gleiche Verhalten gilt auch für den globalen Interrupt.

Der zweite Typ triggert nur so lange, wie die Bedingung für den Interrupt existiert. Wird eine solche Bedingung vor der Interrupt-Freigabe beendet, dann geht dieser Interrupt verloren. Wenn der ATmega328P eine ISR beendet hat, dann wird die Programmabarbeitung nach der Unterbrechungsstelle fortgesetzt. Das Statusregister wird nicht automatisch gesichert, sodass dieser Vorgang manuell vorgenommen werden muss.

Vektor Nr.	Programmadresse	Interrupt	Interrupt-Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interupt 0
3	0x0004	INT1	External Interupt 1
4	0x0006	PCINT0	Pin Change Interrupt 0
5	0x0008	PCINT1	Pin Change Interrupt 1
6	0x000A	PCINT2	Pin Change Interrupt 2
7	0x000C	WDT	Watchdog Interupt
8	0x000E	TIMER2 COMPA	Timer2 CompareA Interrupt
9	0x0010	TIMER2 COMPB	Timer2 CompareB Interrupt
10	0x0012	TIMER2 OVF	Timer2 Overflow Interrupt
11	0x0014	TIMER1 CAPT	Timer1 Capture Interrupt
12	0x0016	TIMER1 COMPA	Timer1 CompareA Interrupt
13	0x0018	TIMER1 COMPB	Timer1 CompareB Interrupt
14	0x001A	TIMER1 OVF	Timer1 Overflow Interrupt
15	0x001C	TIMER0 COMPA	Timer0 CompareA Interrupt
16	0x001E	TIMER0 COMPB	Timer0 CompareB Interrupt
17	0x0020	TIMER0 OVF	Timer0 Overflow Interrupt
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART Tx Complete
22	0x002A	ADC	ADC Conversion Complete

Tabelle 1.3 ATmega328P-Interruptvektortabelle

Vektor Nr.	Programmadresse	Interrupt	Interrupt-Definition
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	I^2C Bus Interface
26	0x0032	SPM READY	Store Program Memory Ready

Tabelle 1.3 ATmega328P-Interruptvektortabelle (Forts.)

Aus den Blockdiagrammen (siehe Abbildung 1.5 und Abbildung 1.6) und der Interruptvektortabelle (siehe Tabelle 1.3) können Sie entnehmen, welche Ereignisse einen Interrupt auslösen können.

Die Anschlussbelegungen (engl. *Pinout*) haben beim Arduino und beim ATmega328P leider andere Bezeichnungen. Abbildung 1.9 zeigt die Zuordnung dieser Bezeichnungen zu den Anschlüssen (*I/O-Pins*) des ATmega328P.

Interrupts anfordernde Ereignisse (in abnehmender Priorität) sind:

- **Reset** – ausgelöst über den Reset-Pin (/RESET), ein Reset beim Zuschalten der Betriebsspannung (*Power-on Reset*) oder beim Unterschreiten der Betriebsspannung eines bestimmten Schwellenwertes (*Brown-out Reset*) bzw. ausgelöst durch den Watchdog Timer.
- **External Interrupt 0/1** – ausgelöst durch eine Pegeländerung an einem der digitalen Eingänge INT0 bzw. INT1.
- **Pin Change Interrupt 0/1/2** – ausgelöst durch eine Pegeländerung an einem der digitalen Eingänge PCINT23 bis PCINT0.
- **Watchdog Interrupt** – ausgelöst durch einen Überlauf des Watchdog Timers. Dieser Überlauf signalisiert einen Fehlerzustand im Programmablauf, denn im Normalfall wird der Watchdog Timer innerhalb seiner einstellbaren Timer-Periode zyklisch zurückgesetzt, was einen solchen Überlauf verhindert.
- **Timer2/Timer1/Timer0 Interrupts** – ausgelöst durch verschiedene Timer-Ereignisse wie Vergleiche mit bestimmten Werten oder Überlauf.
- **SPI Serial Complete Interrupt** – ausgelöst durch eine abgeschlossene Datenübertragung über SPI (*SPI-Transfer*). Das *Serial Peripheral Interface* (SPI) erlaubt eine synchrone High-Speed-Datenübertragung zwischen einem ATmega328P und peripheren Komponenten oder zwischen verschiedenen Mikrocontrollern.
- **USART Interrupts** – ausgelöst durch verschiedene Zustände des seriellen Interface. USART bezeichnet ein komplexes Modul für die synchrone und asynchrone serielle Kommunikation (*Universal Synchronous and Asynchronous Serial Receiver*

and Transmitter). Dieses Modul ist vor allem die Basis für die eingesetzte serielle Schnittstelle (RS232).

- **ADC Conversion Complete Interrupt** – ausgelöst durch das Ende einer Analog/Digital-Umsetzung (ADC).
- **EEPROM Ready Interrupt** – ausgelöst durch das Ende eines Schreibvorgangs auf den internen EEPROM.
- **Analog Comparator Interrupt** – ausgelöst durch einen Vergleich des Analog-Comparators.
- **I²C-Bus Interrupt** – ausgelöst durch verschiedene Zustände der I²C-Kommunikation.
- **Store Program Memory Ready Interrupt** – ausgelöst durch das Ende eines Schreibvorgangs in den Programmspeicher.

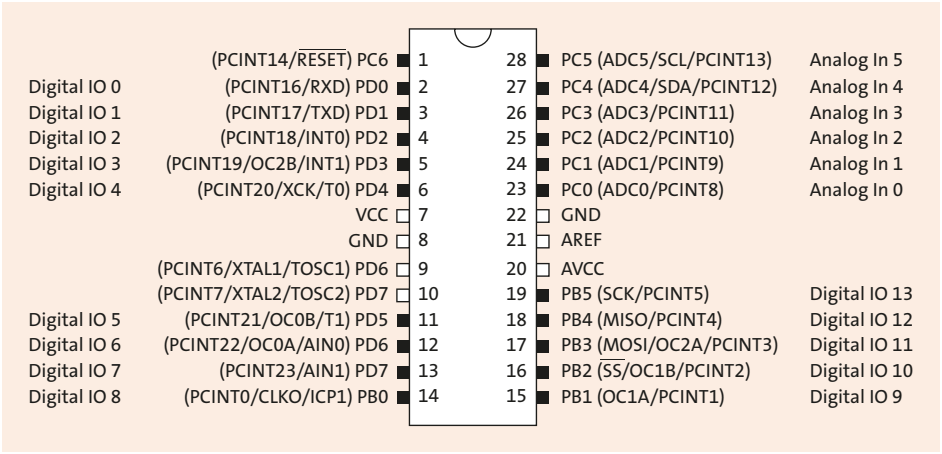

Abbildung 1.9 Anschlussbezeichnungen beim Arduino und ATmega328P

Nicht alle hier vorgestellten Interrupts sind für die Arbeit mit dem Arduino wichtig, einige können die Funktionalität jedoch erweitern. Ich werde bei der Programmierung des Arduino noch darauf zurückkommen.

I/O-Ports

Neben den Anschlüssen für verschiedene Spannungen (VCC, AVCC und AREF) und Masse (GND) ist eine Vielzahl von I/O-Ports am Mikrocontroller außen verfügbar (siehe Abbildung 1.9).

VCC bezeichnet die Versorgungsspannung für die digitalen Schaltungsteile des Mikrocontrollers, AVCC die Versorgungsspannung der analogen. AREF bezeichnet die analoge Referenzspannung für den AD-Umsetzer, und GND steht für das Bezugspotenzial – die Masse. Über diese I/O-Ports können Sie digitale Signale ein- und ausge-

ben und teilweise sogar analoge Spannungswerte erfassen. Abbildung 1.10 zeigt den recht komplexen Aufbau eines digitalen I/O-Ports im Blockschaltbild. Die Steuerung der Funktionen des I/O-Ports erfolgt durch eine umfangreiche interne Logik.

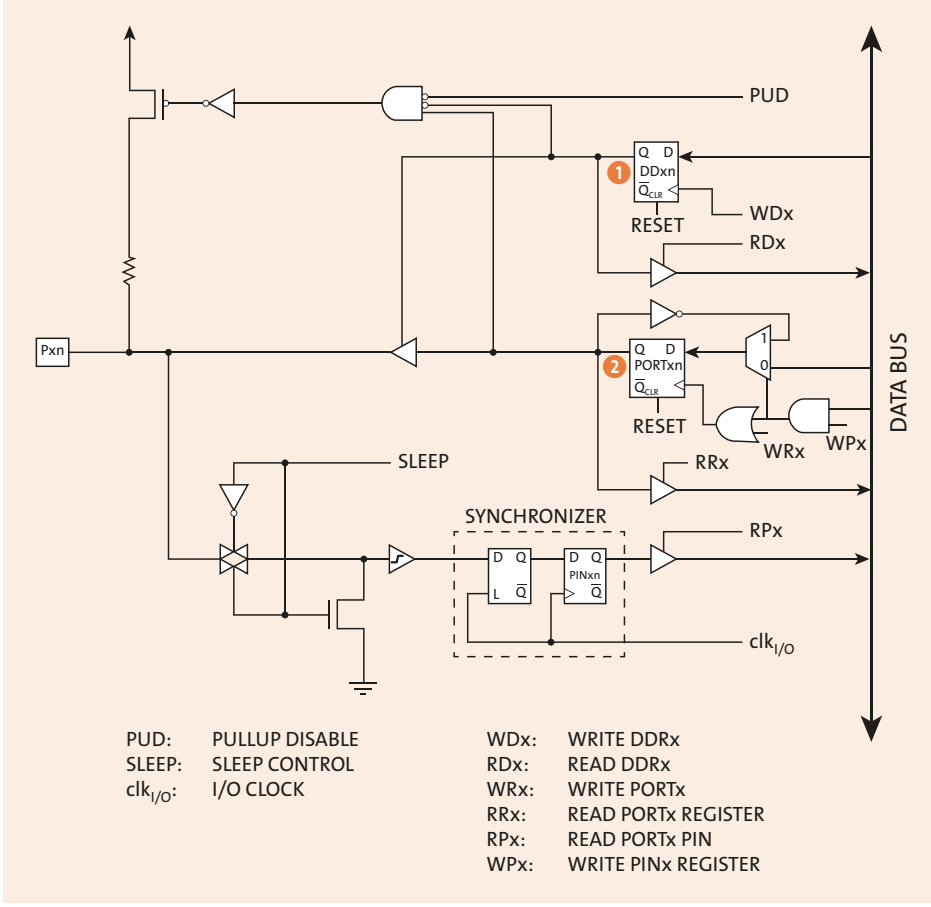

Abbildung 1.10 Digitaler I/O-Port (Quelle: ATmega328P Data Sheet)

Ob ein Pin als digitaler Eingang oder Ausgang arbeitet, entscheidet das *Data Direction Register* DDxn ❶. Das Zeichen x bezeichnet den betreffenden Port (A bis D) und das Zeichen n die betreffende Pinnummer (0 bis 7).

Das Ausgangssignal ist durch das *PORTxn Latch* ❷ zwischengespeichert (gepuffert). Die Zeichen x und n haben die gleiche Bedeutung wie eben.

Mit diesen beiden Latches können vier verschiedene Zustände des digitalen I/O-Ports eingestellt werden (siehe Tabelle 1.4).

DDxn	PORTxn	I/O	Pull-up	Kommentar
0	0	Eingang	Nein	Tri-State (hochohmiger Ausgang)
0	1	Eingang	Ja	Der Pin liefert Strom, wenn der Widerstand gegen GND liegt.
1	0	Ausgang	Nein	Ausgang Low
1	1	Ausgang	Nein	Ausgang High

Tabelle 1.4 Logische Zustände am I/O-Port

Ist der betreffende Pin als Eingang konfiguriert (DDxn = 0), dann entscheidet der Zustand von PORTxn, ob der interne Pull-up-Widerstand ein- oder ausgeschaltet ist. Der Wert des Pull-up-Widerstands kann dabei zwischen 20 und 50 kΩ variieren.

Hinweis

Als Pull-up-Widerstand bezeichnet man einen vom Pin gegen die Betriebsspannung geschalteten Widerstand (pull up = hochziehen). Ein Pull-down-Widerstand wird hingegen vom Pin nach Masse geschaltet (pull down = herunterziehen).

Beide Latches können beschrieben (gesetzt oder zurückgesetzt) werden, und ihr Status kann zurückgelesen werden. Der I/O-Pin kann auch direkt (ohne Latch) gelesen werden. Die verschiedenen Zustände sind deshalb wichtig, da Sie später bei der Konfiguration von I/O-Pins genau diese Zustände einstellen werden.

Jeder Ausgang kann einen Strom von 20 mA liefern. Sie können also eine LED (über einen Vorwiderstand) direkt von einem I/O-Pin ansteuern.

Serial Peripheral Interface (SPI)

Das *Serial Peripheral Interface* (SPI) erlaubt eine synchrone serielle Kommunikation mit hoher Geschwindigkeit zwischen dem Mikrocontroller und den peripheren Komponenten. Diese peripheren Komponenten können beispielsweise Speicher, Displays oder ein anderer Mikrocontroller sein.

Der ATmega328P weist die folgenden Features auf:

- Full-Duplex, Drei-Draht-Interface
- Master oder Slave Mode
- LSB-First- oder MSB-First-Übertragung
- sieben programmierbare Bitraten
- End of Transmission Interrupt

Kapitel 3
Das Experimentierumfeld

In den letzten Kapiteln habe ich Ihnen Grundlagen zum Mikrocontroller allgemein und die Arduino-Familie einschließlich der Shields zur Erweiterung vorgestellt. Bevor wir zum Software-Teil kommen, auf den Sie sicher schon warten, will ich Ihnen aber noch unser Experimentierumfeld näherbringen.

Um Ihren Arduino herum brauchen Sie noch Zusatzschaltungen aus elektronischen Bauelementen, die nach bestimmten Regeln zusammengeschaltet werden sollten, um die gewünschte Funktion sicherzustellen.

Bestimmtes Zubehör unterstützt den elektronischen Aufbau. Bei der Inbetriebnahme helfen Messmittel, und schließlich soll das resultierende Schaltbild dokumentiert und vielleicht sogar eine Leiterplatte daraus gefertigt werden.

All diese spannenden Themen erwarten Sie in diesem Kapitel. Danach können Sie gut vorbereitet in den Software-Teil einsteigen.

3.1 Elektronische Bauteile

Mit den bislang vorgestellten Hardwarekomponenten kennen Sie schon einige elektronische Bauteile. In diesem Kapitel geht es aber darum, dass Sie die grundlegenden Eigenschaften der Bauteile kennenlernen, mit denen wir später unseren Arduino erweitern. Außerdem helfen die Kenntnisse dabei, ein vorgegebenes Schaltbild zu verstehen.

3.1.1 Widerstand, Kondensator und Spule

Mit Widerständen, Kondensatoren und Spulen wurden Sie bereits im Physikunterricht konfrontiert.

Widerstand

Sicher kennen Sie auch noch den Begriff des *ohmschen Widerstandes*. Er ist ein elektrischer Widerstand *R*, dessen Widerstandswert im Gleichstromkreis genauso groß ist wie im Wechselstromkreis. Für ihn gilt das *ohmsche Gesetz*: $U = I \times R$

Das ohmsche Gesetz besagt, dass die Stromstärke I in einem Leiter und die Spannung U zwischen den Enden des Leiters direkt proportional sind.

Außerdem erzeugt der Stromfluss eine Erwärmung des Widerstands. Es wird Leistung P umgesetzt, die nach der Beziehung

$$P = U \times I = I^2 \times R = \frac{U^2}{R}$$

berechnet werden kann.

Das Schaltbild für den Widerstand zeigt Abbildung 3.1. Die linke Darstellung ❶ entspricht der EU-Norm und bezeichnet einen Widerstand R1 mit einem Wert von 1 kΩ (kOhm). Die rechte Darstellung ❷ entspricht der US-Norm und bezeichnet einen Widerstand R2 mit einem Widerstandswert von 2,2 kΩ. Ich zeige hier beide Normen, da beispielsweise im weitverbreiteten CAD-Programm *Fritzing*, das ich Ihnen in Abschnitt 3.7.1 vorstelle, die US-Symbole verwendet werden.

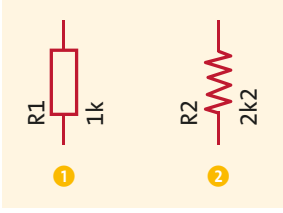

Abbildung 3.1 Schaltbild für den Widerstand

Widerstände sind durch die Kenngrößen *Nennwiderstand*, *Belastbarkeit* und *Toleranz* bestimmt.

Berechnungsbeispiel

Ein Widerstand soll bei einer Spannung von 5 V den Stromfluss auf 25 mA begrenzen. Nach dem ohmschen Gesetz bekommen wir einen Widerstandswert von 200 Ω.

In der Widerstandsreihe E48 finden wir aber nur einen Widerstand von 205 Ω (siehe Abschnitt A.3.2), wodurch der Strom 24,39 mA betragen wird. Die umgesetzte Leistung beträgt 122 mW.

Berechnete Kenngrößen:

- Nennwiderstand 205 Ω
- Belastbarkeit 0,125 W (SMD-Bauform min. 0805)
- Toleranz 2 % (E48)

Ohmsche Widerstände als Bauteil gibt es in verschiedenen Bauformen. Sicher jedem bekannt sind bedrahtete Widerstände, deren Widerstandswert durch farbige Ringe gekennzeichnet wird. Abbildung 3.2 zeigt hierfür ein Beispiel.

Abbildung 3.2 Bedrahtete Widerstände (Quelle: Afrank99 – Eigenes Werk, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=456666)

Auf den bisher vorgestellten Boards war diese Bauform nicht vorhanden, obwohl Sie gerade für Versuchsaufbauten diese Widerstände sicher bevorzugen werden, weil sie sich gut handhaben lassen. Weil man Boards mit ihnen besser maschinell bestücken kann, werden auf den Boards heute praktisch ausschließlich Widerstände in SMD-Bauform (*Surface Mounted Device*) eingesetzt (siehe Abbildung 3.3). Sie sind als Dünnschicht- bzw. Metallschicht-, Metallglasur- und Kohleschichtwiderstände erhältlich.

Abbildung 3.3 SMD-Widerstand (Quelle: Haragayato, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=199075)

Technische Daten zur Kennzeichnung und zur Bauform habe ich im Anhang in Abschnitt A.3 zusammengestellt. Neben den hier betrachteten Festwiderständen (Widerstand mit festem Wert) gibt es Widerstände, die von einem Parameter abhängig sind. Zu ihnen zählen:

- Potenziometer
- Fotowiderstand
- temperaturabhängiger Widerstand (Thermistor)
- spannungsabhängiger Widerstand (Varistor)
- druck- und dehnungsabhängiger Widerstand

Schaltungstechnisch sind diese parameterabhängigen Widerstände als ohmsche Widerstände zu betrachten, nur dass ihr Widerstandswert eine Funktion des betreffenden Parameters darstellt.

Kondensator

Auch der Kondensator wird Ihnen aus dem Physikunterricht bekannt sein. Der Kondensator verhält sich im Gleichstromkreis anders als im Wechselstromkreis. Für unseren Anwendungsfall in einer digitalen Schaltung steht der Gleichstromkreis im Vordergrund, den wir hier auch vorrangig betrachten.

Der Kondensator kann eine Ladung *Q* aufnehmen, die zu seiner Kapazität *C* proportional ist:

$$Q = C \times U$$

Der Spannung *U* am Kondensator sind durch die Spannungsfestigkeit des Dielektrikums Grenzen gesetzt.

Der Ladevorgang eines Kondensators *C* über einen Vorwiderstand *R* folgt einer Exponentialfunktion:

$$u(t) = U \times (1 - e^{\frac{-t}{RC}})$$

Dieses Verhalten ist deshalb wichtig, da hiervon zeitabhängige Größen *f(t)* abgeleitet werden.

Es gibt sehr viele verschiedene Kondensatortypen und jeder hat seine spezifischen Eigenschaften und Anwendungen. Man unterscheidet die Kondensatortypen in der Regel anhand des Dielektrikums.

Abbildung 3.4 zeigt Keramikkondensatoren unterschiedlicher Bauformen.

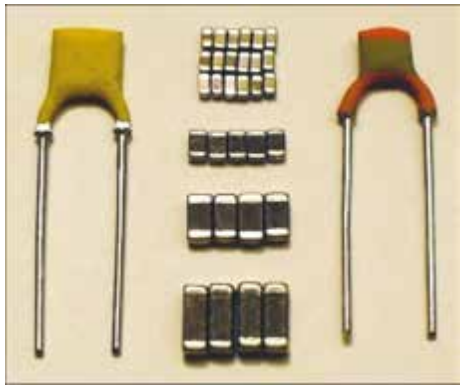

Abbildung 3.4 Keramikkondensatoren (Quelle: Elcap [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)])

Der *Keramikkondensator* weist ein keramisches Dielektrikum auf. Die Zusammensetzung der Inhaltsstoffe des Dielektrikums ermöglicht die Herstellung von Kondensatoren mit vorher bestimmbaren elektrischen Eigenschaften. Keramikkondensatoren werden überwiegend mit Kapazitäten von 1 pF bis 100 µF hergestellt. In der Bauform

des Keramikvielschicht-Chipkondensators (*Multi Layer Ceramic Capacitor*, MLCC) sind sie die am häufigsten eingesetzten diskreten Kondensatoren in der Elektronik. Sie werden als Entstör-, Durchführungs- oder als Leistungskondensatoren verwendet. Wenn Sie Ihre Kenntnisse zum Keramikkondensator vertiefen wollen, dann finden Sie unter *https://de.wikipedia.org/wiki/Keramikkondensator* eine detaillierte Beschreibung zahlreicher Aspekte des Keramikkondensators.

Folienkondensatoren weisen isolierende Kunststofffolien als Dielektrikum auf. Nach den Keramikkondensatoren und Elektrolytkondensatoren gehören Folienkondensatoren zu den am häufigsten eingesetzten Kondensatorbauarten in der Elektronik.

Die Hauptvorteile von metallisierten Kunststofffolienkondensatoren sind sehr niedrige ohmsche Verluste und eine kleine Induktivität. Sie zeichnen sich durch eine hohe Impulsbelastbarkeit aus.

Bei Einsatz von Polypropylenfolie kommt außerdem noch eine geringe Temperaturabhängigkeit der Kapazität und des Verlustfaktors hinzu. Sie werden in Oszillatoren, Schwingkreisen, Frequenzfiltern, Abstimmkreisen und temperaturstabilen Zeitgliedern eingesetzt. Darüber hinaus werden sie in AD-Umsetzern (*Sample-and-Hold-Schaltung* für AD-Umsetzer) eingesetzt. Die beiden letzten Einsatzgebiete sind durchaus für unsere Anwendungen interessant.

Wenn Sie Ihre Kenntnisse zum Folienkondensator vertiefen wollen, dann finden Sie unter *https://de.wikipedia.org/wiki/Kunststoff-Folienkondensator* eine detaillierte Beschreibung zahlreicher Aspekte des Folienkondensators. Abbildung 3.5 zeigt Folienkondensatoren unterschiedlicher Bauform.

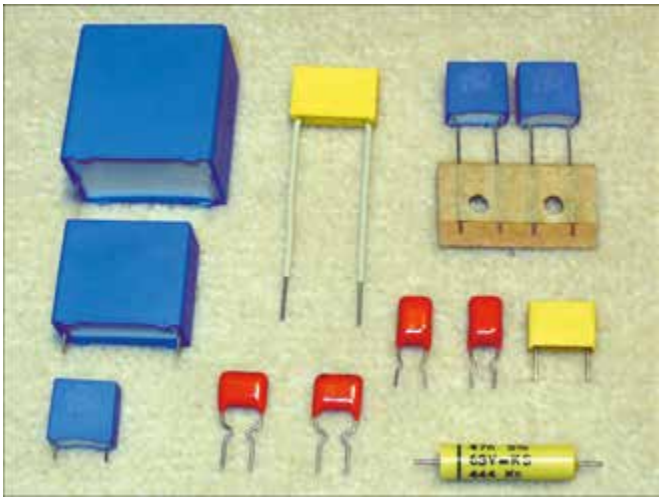

Abbildung 3.5 Folienkondensatoren (Quelle: Elcap [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)])

Ein *Elektrolytkondensator* ist ein gepolter Kondensator, dessen Anode aus Metall besteht, auf dem durch anodische Oxidation eine gleichmäßige, äußerst dünne, elektrisch isolierende Oxidschicht erzeugt wird, die das Dielektrikum des Kondensators bildet. Ein flüssiger oder fester Elektrolyt, der sich geometrisch der Oberflächenstruktur der Anode anpasst, bildet die Kathode des Elektrolytkondensators.

Aluminium-Elektrolytkondensatoren sind preiswerter als Tantal- bzw. Niob-Elektrolytkondensatoren und werden überall in der Elektronik eingesetzt.

Gegenüber den Keramik- und Folienkondensatoren weisen Elektrolytkondensatoren eine hohe volumenbezogene Kapazität auf.

Elektrolytkondensatoren sind gepolte Bauteile, die nur mit Gleichspannung betrieben werden dürfen. Falschpolung, zu hohe Spannung (oberhalb der Nennspannung) oder Rippelstrom-Überlastung können den Kondensator zerstören. Die Zerstörung kann katastrophale Folgen (Explosion, Brand) nach sich ziehen!

Durch die große spezifische Kapazität eignen sich Elektrolytkondensatoren besonders zum Entkoppeln unerwünschter Frequenzen vom zweistelligen Hertz-Bereich bis hin zu einigen Megahertz, zum Glätten gleichgerichteter Spannungen in Netzteilen, Schaltnetzteilen und Gleichspannungswandlern. Sie puffern auch Versorgungsspannungen bei plötzlichen Lastspitzen in digitalen Schaltungen.

Auch hier darf der Hinweis auf die ausgezeichnete Darstellung der Thematik in der deutschen Wikipedia nicht fehlen. Wenn Sie also mehr Informationen zum Elektrolytkondensator benötigen, dann finden Sie unter *https://de.wikipedia.org/wiki/Elektrolytkondensator* wieder eine detaillierte Beschreibung zahlreicher Aspekte des Elektrolytkondensators. Abbildung 3.6 zeigt Elektrolytkondensatoren unterschiedlicher Bauform.

Abbildung 3.6 Elektrolyt- und Tantal-Kondensatoren (Quelle: Elcap [CC0 (https://creativecommons.org/publicdomain/zero/1.0/deed.de)])

Das Schaltbild für den Kondensator zeigt Abbildung 3.7. Die linke Darstellung ❶ entspricht der EU-Norm und bezeichnet einen nicht polarisierten Kondensator C1 mit einem Wert von 10 nF. Die Darstellung rechts daneben ❷ entspricht der US-Norm und bezeichnet einen nicht polarisierten Kondensator C2 mit einem Wert von 100 pF. Auf der rechten Seite von Abbildung 3.7 sind die Schaltbilder polarisierter (Elektrolyt-)Kondensatoren zu sehen. Die Darstellung ❸ entspricht wieder der EU-Norm und zeigt einen Elektrolytkondensator C3 mit einem Wert von 10 µF, während die Darstellung ❹ der US-Norm entspricht und einen Elektrolytkondensator C4 mit einem Wert von 1 µF zeigt.

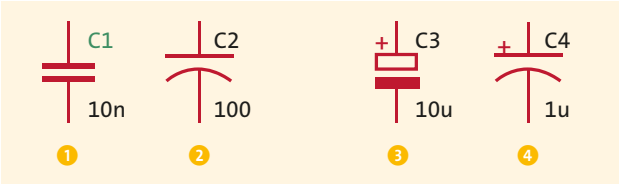

Abbildung 3.7 Schaltbild für Kondensatoren

Spule

Spulen können Teile eines Gerätes, z. B. eines Relais, eines Elektromagneten oder Elektromotors, eines Transformators oder Lautsprechers sein. Bei diesen Anwendungen wird das erzeugte Magnetfeld unmittelbar ausgenutzt.

Spulen sind aber auch passive, induktive Bauelemente mit einer definierten Induktivität, die sich im Gleichstromkreis anders als im Wechselstromkreis verhalten. Eingesetzt werden Spulen unter anderem in Filterschaltungen, zur Störungsunterdrückung oder als Energiespeicher in Schaltnetzteilen.

In einer Spule mit n Windungen, deren Länge l größer als ihr Durchmesser ist, erzeugt ein fließender Strom I ein Magnetfeld H nach folgender Beziehung:

$$H = \frac{n}{l} \times I$$

Die Strom-Spannungsbeziehung an der Spule lautet:

$$u(t) = L \frac{di}{dt}$$

Hier bezeichnen L die Induktivität der Spule und di/dt die Stromänderung über die Zeit.

Mit anderen Worten erzeugt eine Stromänderung an einer Spule eine Spannung. Beim Schalten von Spulen und Relais müssen Sie dieses Verhalten berücksichtigen, was durch sogenannte Freilaufdioden geschieht. Abbildung 3.8 zeigt typische Ringkern-Spulen.

Abbildung 3.8 Typische Ringkern-Spulen

Bei den in diesem Buch betrachteten Arduino-Anwendungen spielen die Spulen als eigenständiges Bauteil eine untergeordnete Rolle. Ganz anders sieht das bei Relais, Magneten und Elektromotoren aus.

3.1.2 Taster, Schalter und Relais

Taster und Schalter sind mechanische Elemente, die den Stromfluss ermöglichen oder unterbrechen. Der Unterschied zwischen einem Taster und einem Schalter liegt in der mechanischen Verriegelung. Ein Taster ist nur bei Betätigung aktiv (offen oder geschlossen) und kehrt nach der Betätigung wieder in seinen Ausgangszustand zurück, während ein Schalter bei Betätigung geschlossen wird und sich erst bei erneuter Betätigung wieder öffnet oder umgekehrt.

Abbildung 3.9 zeigt die Schaltbilder für Taster 1, Schalter 2, DIP-Schalter 3, Drehschalter 4 und ein Relais mit einem Umschaltkontakt 5. Taster und Schalter werden durch die Kennzeichnung der mechanischen Betätigung an der Schaltzunge unterschieden.

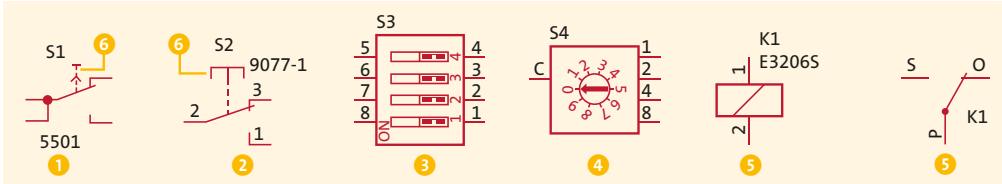

Abbildung 3.9 Taster, Schalter und Relais – Schaltbilder

Abbildung 3.10 zeigt beispielhaft für die Leiterplattenbestückung geeignete Bauformen für Taster, Schalter und Relais. Die Vielfalt ist hier außerordentlich groß. Sicher kennen Sie auch andere Bauformen für diese Gruppe von Bauelementen, die wir in unseren Anwendungen immer wieder vorfinden werden.

Abbildung 3.10 Taster, Schalter und Relais – Beispiele für Bauformen

3.1.3 Dioden

Dioden möchte ich insgesamt betrachten, da sie eine wesentliche gemeinsame Eigenschaft aufweisen: Sie lassen den Stromfluss in eine Richtung durch und sperren ihn in der anderen Richtung. In der Halbleitertechnik bezieht sich der Begriff *Diode* in der Regel auf Siliziumdioden mit einem p-n-Übergang. Das sind auch die Dioden, mit denen wir in unseren Anwendungen zu tun haben werden. Andere Varianten werden z. B. speziell als Schottky-Diode oder Germanium-Diode bezeichnet.

Dioden werden wegen ihrer richtungsabhängigen Leitfähigkeit zur Gleichrichtung von Wechselspannung zu Gleichspannung, für logische Verknüpfungen, als Freilaufdioden und für viele weitere Zwecke eingesetzt.

Abbildung 3.11 zeigt die Strom-Spannungskennlinie einer Diode des Typs 1N914. Deutlich erkennbar ist das von einem idealen Schalter abweichende Verhalten.

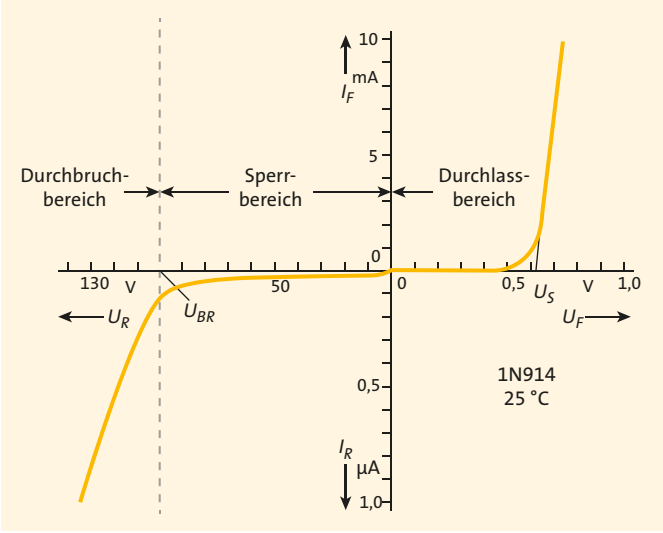

Abbildung 3.11 Dioden-Kennlinie

Im *Durchlassbereich* muss erst eine Flussspannung von ca. 0,5 V überwunden werden, bevor ein hinreichend großer Strom durch die Diode fließen kann. Im Bereich um 0,65 V ändert sich bei einer Si-Diode die Spannung nur noch wenig.

In diesem Bereich kann das Verhalten der Diode sehr gut durch die Shockley-Gleichung beschrieben werden:

$$I_D = I_S \times (\exp\left(\frac{U_D}{n\,U_T}\right) - 1)$$

Hier gilt:

- Strom durch die Diode in Flussrichtung I_D
- (Sättigungs-)Sperrstrom $I_S = 10^{-12} \ldots 10^{-6}$ A
- Emmisionskoeffizient $n = 1 \ldots 2$
- Temperaturspannung $U_T = kT/q$ (ca. 25 mV bei Raumtemperatur)
- Boltzmann-Konstante $k = 1381 \times 10^{-21}$ Ws/K
- Elementarladung $e = 1602 \times 10^{-19}$ As

Im *Sperrbereich*, das bedeutet umgekehrte Polarität der anliegenden Spannung, fließt ein kleiner Sperrstrom. Erreicht die in Sperrrichtung an der Diode angelegte Spannung die sogenannte *Durchbruchspannung*, dann ist die Feldstärke im p-n-Übergang so groß, dass durch den Durchbrucheffekt bedingt der Stromfluss stark zunimmt. Dieser Bereich ist für die normale Anwendung tabu. Nur bei Z-Dioden wird dieser Durchbruch speziell zur Erzeugung von Spannung-Referenzen genutzt.

Es gibt aber weitere Effekte am p-n-Übergang, die für spezielle Dioden genutzt werden. Bei Fotodioden wird die Lichtempfindlichkeit des p-n-Übergangs ausgenutzt. Der Sperrstrom ist proportional zum einfallenden Licht, weshalb die Fotodiode sehr gut zur Lichtmessung eingesetzt werden kann.

Der prinzipielle Aufbau einer *Leuchtdiode* (*LED*) entspricht dem einer p-n-Halbleiterdiode, wodurch LEDs die gleichen Grundeigenschaften besitzen. Dioden aus Silizium oder Germanium leuchten nicht. Ist das Ausgangsmaterial aber ein direkter Halbleiter (meist eine Gallium- oder Galliumarsenid-Verbindung), dann wird vom p-n-Übergang bei Stromfluss in Durchlassrichtung Licht abgegeben. LEDs haben als Anzeigeelemente eine herausragende Bedeutung. In Abschnitt 7.1 gehe ich auf deren Besonderheiten ein.

Abbildung 3.12 zeigt die Schaltbilder für verschiedene Dioden. D1 ❶ steht für eine »Allzweck«-Diode 1N4446, die es von verschiedenen Herstellern gibt. D2 ❷ ist eine Z-Diode, was durch die den Durchbruch kennzeichnende Katode verdeutlicht wird. D3 ❸ steht für eine Fotodiode. Die Lichtempfindlichkeit wird durch die Pfeile markiert, die einfallende Strahlung symbolisieren. Umgekehrt ist das bei der LED ❹. Alle diese Bauteile werden wir in den Anwendungen wiederfinden.

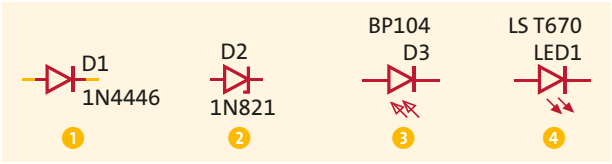

Abbildung 3.12 Dioden – Schaltbilder

Abbildung 3.13 zeigt unterschiedliche Bauformen der in Abbildung 3.12 dargestellten Dioden. Die »Allzweck«-Diode 1N4446 ❶ und die Z-Diode 1N821 ❷ haben Standardgehäuse mit axialen Anschlüssen (DO-204AA). Die Fotodiode BP104 ❸ weist ein für die SMD-Montage geeignetes DIL-Gehäuse (*Dual In Line*) auf. Die LED LS T670 ❹ mit ihrem PLCC-Gehäuse ist ebenfalls für die SMD-Montage geeignet. Die traditionelle LED-Bauform sind die 3-mm- oder 5-mm-Rundgehäuse ❺, die gerade für Experimente gut geeignet sind. Im Rundgehäuse gibt es auch zwei- oder dreifarbige LEDs, die durch mehrere verschiedenfarbige LED-Chips gebildet werden.

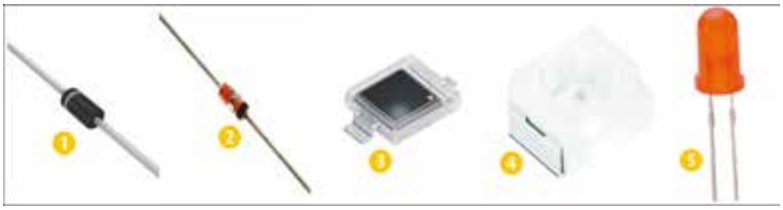

Abbildung 3.13 Dioden – Beispiele für Bauformen

Die Vielfalt ist auch hier außerordentlich groß. Sicher kennen Sie andere Bauformen für diese Gruppe von Bauelementen, die wir in unseren Anwendungen immer wieder vorfinden werden

3.1.4 Transistoren und FETs als Schalter

Bei Bipolar- und Feldeffekt-Transistoren (*FET*) mache ich bereits in der Überschrift eine Einschränkung. Im Umfeld des Arduino werden diese Bauteile im Wesentlichen als Schalter betrieben, was die Betrachtung dieser Bauteile deutlich vereinfacht.

Wenn von Transistoren gesprochen wird, dann sind in der Regel Bipolartransistoren gemeint, die es als npn-Transistoren bzw. pnp-Transistoren gibt. Beide Varianten bestehen aus jeweils zwei p-n-Übergängen.

Abbildung 3.14 zeigt schematisch den Aufbau eines npn-Transistors. Die beiden p-n-Übergänge sind vertikal angeordnet.

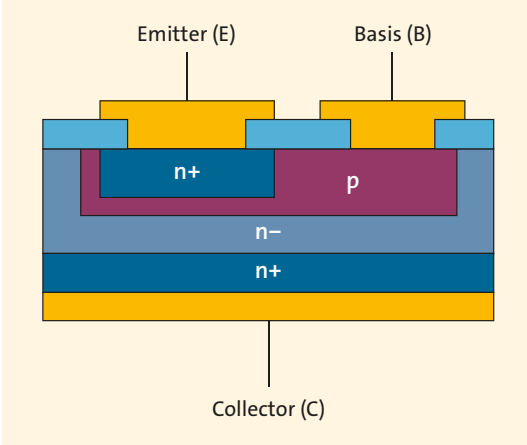

Abbildung 3.14 Aufbau des npn-Transistors

Beschrieben wird das Verhalten eines Transistors durch ein recht komplexes Kennlinienfeld (siehe Abbildung 3.15). Die im Kennlinienfeld verwendeten Bezeichnungen für Spannungen und Ströme sind gemäß Abbildung 3.16 definiert.

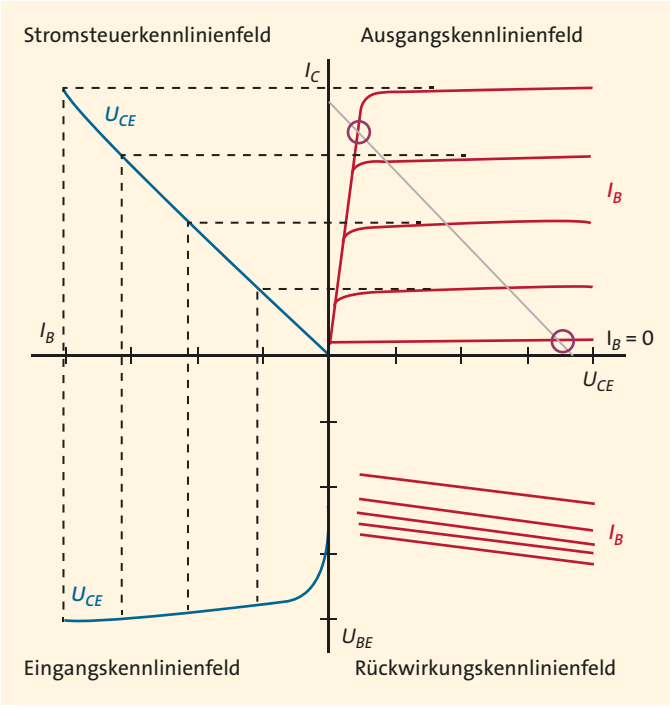

Abbildung 3.15 Transistor-Kennlinienfeld

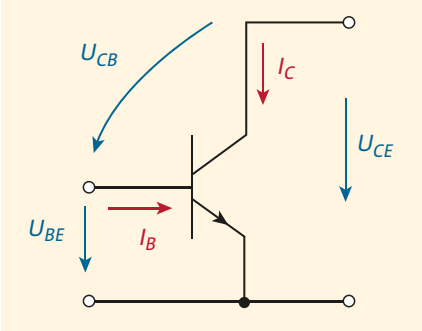

Abbildung 3.16 Spannungen am npn-Transistor

Das *Eingangskennlinienfeld* beschreibt das Verhalten des Basis-Emitter-Übergangs und zeigt uns eine Dioden-Kennlinie. Diese Kennlinie haben Sie in Abschnitt 3.1.3 bereits für den Durchlassbereich der Diode kennengelernt. Eine am Basis-Emitter-Übergang anliegende Spannung U_{BE} hat einen Basisstrom I_B zur Folge. Im eingeschalteten Zustand des Transistors wird die Basis-Emitter-Spannung ca. 0,65 V betragen und sich in Abhängigkeit vom Strom nur noch wenig ändern.

Durch den fließenden Basisstrom wird die Basis mit Ladungsträgern geflutet, was Auswirkungen auf die Sperrschicht des Collector-Basis-Übergangs hat und diesen Übergang ebenfalls leitend macht. Der dadurch fließende Kollektorstrom ist wesentlich größer als der Basisstrom, was in der nahezu linearen Stromverstärkung des Transistors zum Ausdruck kommt. Die *Stromsteuerkennlinie* bringt das zum Ausdruck. Stromverstärkungen von $B_N = 100$ und mehr sind üblich. Es gilt die Beziehung:

$$I_C = B_N \times I_B$$

Das *Ausgangskennlinienfeld* beschreibt das Verhalten der Collector-Emitter-Strecke. Als Steuergröße wirkt der Basisstrom. Für den aktiven Betrieb als Verstärkerelement ist das Kennlinienfeld insgesamt interessant, im Betrieb als Schalter nur das Gebiet nahe der Abszisse (x-Achse) und nahe der Ordinate (y-Achse).

Fließt kein Basisstrom (Sperrzustand des Transistors), dann verbleibt ein äußerst kleiner Collector-Emitter-Reststrom I_{CEO}, der meist im nA-Bereich liegt, und die Collector-Emitter-Spannung U_{CE} liegt sehr nahe an der Betriebsspannung. Fließt hingegen ein ausreichend großer Basisstrom, dann leitet der Transistor und kommt in die Übersteuerung ($U_{CB} < 0$). Die Collector-Emitter-Spannung verharrt bei der Collector-Emitter-Sättigungsspannung U_{CES}, die in der Regel unterhalb von 200 mV liegt. Diese beiden Arbeitspunkte des Transistors im Betrieb als Schalter habe ich ins Ausgangskennlinienfeld eingezeichnet (siehe Abbildung 3.15).

Der *Feldeffekttransistor* (FET) hat einen einfacheren Aufbau als der Bipolartransistor (siehe Abbildung 3.17).

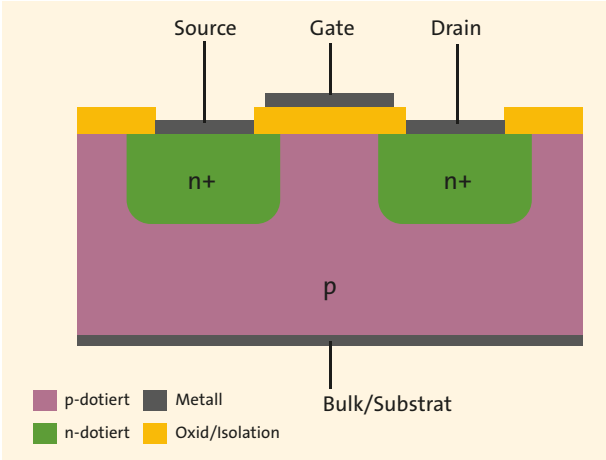

Abbildung 3.17 Aufbau eines n-Kanal-MOSFET

Die Leitfähigkeit des Kanals zwischen Drain- und Source-Anschluss wird durch die Gate-Source-Spannung gesteuert. Beim FET handelt es sich also um einen gesteuerten Widerstand. Der große Vorteil gegenüber Bipolartransistoren ist die verlustfreie Ansteuerung. Der Eingang des FETs ist hochohmig. Der Widerstand zwischen Drain und Source im eingeschalteten Zustand ist nicht zu vernachlässigen und ist stark bauelementeabhängig. Hier müssen Sie in jedem Fall das Datenblatt konsultieren! Einige Beispiele helfen, das zu verdeutlichen:

Typ	Bauteil	R_{DSon}	U_{DS} @ I_D = 100 mA
n-Kanal	BS170	5 Ω	500 mV
	BSS295	0,3 Ω	30 mV
	2SK2961	0,2 Ω	20 mV
	IRLD024	0,1 Ω	10 mV
p-Kanal	BS250	14 Ω	1400 mV
	VP0808L	5 Ω	500 mV
	VP0300L	2,5 Ω	250 mV
	IRFD9024	0,28 Ω	28 mV

Tabelle 3.1 R_{DSon} verschiedener FETs

Aufbaubedingt sind die Widerstände im eingeschalteten Zustand bei p-Kanal-FETs höher als bei n-Kanal-FETs.

Kommen wir zum praktischen Teil und betrachten die Schaltbilder in Abbildung 3.18. Der Pfeil am Emitter des Bipolartransistors zeigt, ob es sich um einen npn-Transistor ❶ oder um einen pnp-Transistor ❷ handelt. Beim FET zeigt die unterbrochene Strecke zwischen Drain und Source, dass der Kanal nicht selbstleitend ist (selbstsperrender bzw. Enhancement-Typ). Den selbstleitenden FET (Verarmungstyp bzw. Depletion Typ) habe ich hier nicht betrachtet. Der Pfeil am Gate kennzeichnet die Dotierung des Kanals (n-Kanal ❸, p-Kanal ❹).

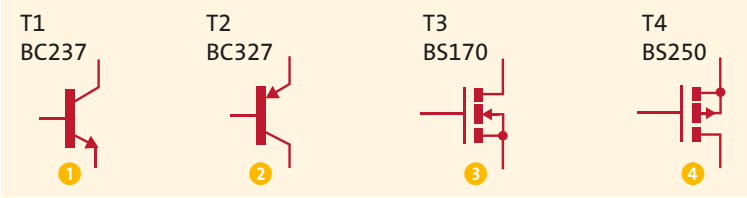

Abbildung 3.18 Schaltbilder für Transistor und FET

Auch für FETs wie für Bipolartransistoren gibt es wieder sehr unterschiedliche Bauformen. Abbildung 3.19 zeigt links TO-92-Gehäuse. Die Gehäuse rechts sind für die SMD-Bestückung geeignet und werden für leistungsstärkere Typen verwendet.

Abbildung 3.19 Bauformen für Bipolartransistoren und FETs

Wie die verschiedenen Transistoren und FETs als Schalter in Digitalschaltungen eingesetzt werden, zeigen Ihnen die folgenden Abbildungen.

Abbildung 3.20 zeigt einen npn-Transistor als Schalter. Diese Schaltung werden Sie sehr oft wiederfinden, z. B. zur Ansteuerung von LEDs, Relais und anderen Bauelementen. Geschaltet wird die Last am Kollektor, der hier durch den Widerstand R3 dargestellt und gegen die Betriebsspannung geschaltet ist. Eine hohe Spannung am Eingang (IN) schaltet den Transistor ein. Der Widerstand R1 dient zur Begrenzung des Eingangsstroms. Bei Einsatz des Widerstands R2 wird die Einschaltschwelle für den Transistor beeinflusst. Ich komme in einem Berechnungsbeispiel noch darauf zurück.

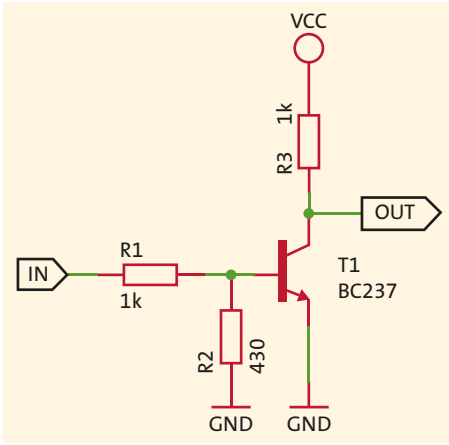

Abbildung 3.20 npn-Transistor als Schalter

Beim Einsatz eines pnp-Transistors kehren sich die Verhältnisse um. Die Last ist hier gegen Masse (GND) geschaltet, und eine niedrige Spannung am Eingang (IN) schaltet den Transistor ein (siehe Abbildung 3.21).

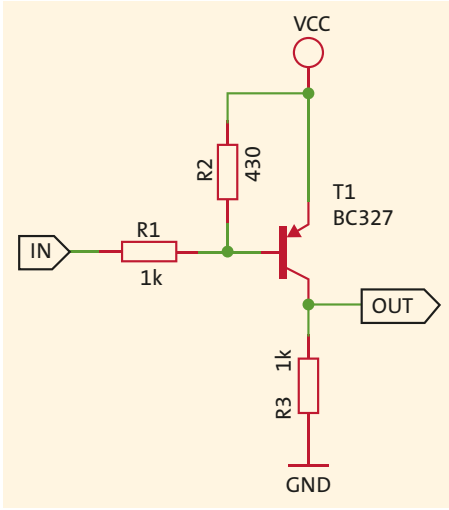

Abbildung 3.21 pnp-Transistor als Schalter

Die Schaltungen für FETs sind denen der Bipolartransistoren sehr ähnlich (siehe Abbildung 3.22 und Abbildung 3.23). Ich bin hier von FETs mit Schaltspannungen am Gate im Bereich von bis zu 5 V ausgegangen, wodurch eine aufwendigere Schaltung am Gate vermieden werden kann.

Da bauartbedingt die Kapazität zwischen Gate und Source nicht zu vernachlässigen ist, kann der Widerstand R1 dynamisch entkoppeln bzw. eine Schwingneigung unterdrücken. Statisch hat er wegen des hochohmigen Eingangs keine Bedeutung.

Der n-Kanal-FET wird durch eine hohe Spannung am Eingang (IN) eingeschaltet. Beim p-Kanal-FET geschieht das wiederum durch eine niedrige Spannung am Eingang (IN).

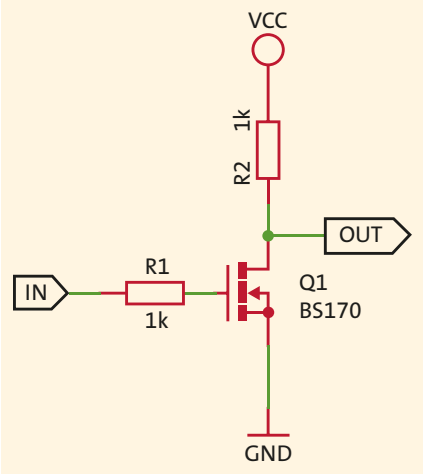

Abbildung 3.22 n-Kanal-FET als Schalter

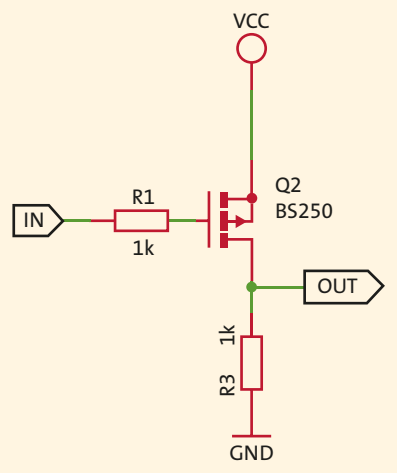

Abbildung 3.23 p-Kanal-FET als Schalter

3.1.5 Operationsverstärker

Operationsverstärker sind eine wichtige Klasse analoger Schaltungen und könnten ein eigenes Buch füllen. Ich erwähne sie hier nur deshalb, weil Sie die relevanten Grundlagen kennen sollten, um ihren Einsatz zur Anbindung von Sensoren oder in Verbindung mit AD-Umsetzern (*ADU*) zu verstehen.

Operationsverstärker gibt es in vielen verschiedenen Ausführungen. Allen gemeinsam sind aber die folgenden Eigenschaften (idealer Operationsverstärker):

- sehr hohe Verstärkung (unendliche Verstärkung)
- hoher Eingangswiderstand (unendlicher Eingangswiderstand)
- vernachlässigbare Eingangsströme (keine Eingangsströme)
- sehr niedriger Ausgangswiderstand (kein Ausgangswiderstand)

Durch die hohe Verstärkung des Operationsverstärkers bedingt, wird die Eingangsspannung – das ist die Differenz der Spannungen an den beiden Eingängen (+) und (–) – sehr klein bzw. geht gegen null.

Für den in Abbildung 3.24 gezeigten invertierenden Operationsverstärker bedeutet das, dass die Spannung am Eingang (–) scheinbar auf GND-Potenzial zu liegen kommt (virtueller GND). Der durch den Widerstand R1 fließende Strom wird also mit dem durch den Widerstand R2 fließenden Strom identisch sein.

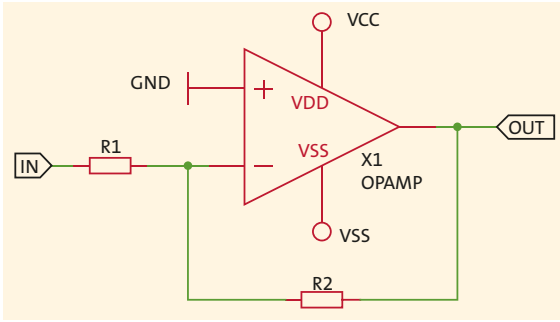

Abbildung 3.24 Invertierender Operationsverstärker

Ich greife hier dem nächsten Kapitel einmal kurz vor und gebe die Formel für das Übertragungsverhalten des invertierenden Operationsverstärkers an:

$$U_{OUT} = -\frac{R2}{R1} U_{IN}$$

Wenn wir die oben angegebenen Bedingungen für einen idealen Operationsverstärker annehmen, bestimmen die Widerstände R1 und R2 das Verhalten der Gesamtschaltung. Die Spannungsversstärkung entspricht dem Verhältnis –(R2/R1), daher der Name *invertierender Verstärker* (eine positive Eingangsspannung hat eine negative Ausgangsspannung zur Folge). Der Eingangswiderstand der Schaltung ist identisch mit dem Widerstand R1.

Die Betriebsspannungen VCC und VSS liegen hier symmetrisch zu GND. Beispielsweise sind oft VCC = 12 V und VSS = –12 V oder auch VCC = 5 V und VSS = –5 V.

Durch Vertauschen der Eingänge bekommt man einen nichtinvertierenden Operationsverstärker gemäß Abbildung 3.25.

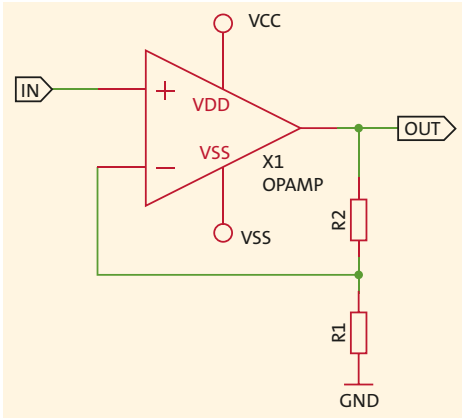

Abbildung 3.25 Nichtinvertierender Operationsverstärker

Der Eingang der Schaltung führt direkt an den Eingang (+) des Operationsverstärkers. Die Eingangsspannung liegt damit auch über dem Widerstand R1, die Ausgangsspannung hingegen über der Reihenschaltung von R1 und R2. Für das Übertragungsverhalten gilt dann:

$$U_{\mathrm{OUT}} = \left(\frac{R2}{R1} + 1\right) U_{\mathrm{IN}}$$

Die minimale Spannungsverstärkung beträgt hier also 1. Größere Spannungsverstärkungen sind über das Verhältnis R2/R1 einstellbar. Eine positive Eingangsspannung hat auch eine positive Ausgangsspannung zur Folge.

Der sogenannte *Spannungsfolger* (R2 = 0, R1 nicht vorhanden => v = 1) dient häufig zur Entkopplung. Der Eingangswiderstand ist durch den Operationsverstärker gegeben sehr hoch und der Ausgangswiderstand sehr niedrig.

Wenn Sie sich tiefer in diese Thematik einarbeiten möchten, dann kann ich Ihnen einen Blick in das Elektronikkompendium unter *http://www.elektronik-kompendium.de/sites/bau/0209092.htm* empfehlen.

3.2 Grundlagen zur Schaltungstechnik

An einigen Stellen hatte ich bereits auf Grundlagen zur Schaltungstechnik zurückgegriffen, die ich Ihnen in diesem Abschnitt aber zusammengefasst vorstellen möchte. Keine Angst: Es ist keine graue Theorie, und Sie werden Schaltungsdetails besser verstehen, wenn Ihnen diese Grundlagen bekannt sind.

3.2.1 Ohmsches Gesetz

Bei der Betrachtung des Widerstands *R* in Abschnitt 3.1.1 habe ich bereits das ohmsche Gesetz erwähnt. Es besagt, dass die Stromstärke *I* in einem Leiter und die Spannung *U* zwischen den Enden des Leiters direkt proportional sind:

$U = I \times R$

Dieser Zusammenhang ist fundamental für alle unsere schaltungstechnischen Betrachtungen.

3.2.2 Kirchhoffsche Regeln

Bei der Analyse elektrischer Schaltungen werden die kirchhoffschen Regeln eingesetzt. Die kirchhoffschen Regeln beschreiben den Zusammenhang zwischen Strömen und Spannungen in einer elektrischen Schaltung. Sie unterteilen sich in den *Knotenpunktsatz* und den *Maschensatz*.

Knotenpunktsatz

Der Knotenpunktsatz besagt, dass in einem Knotenpunkt einer elektrischen Schaltung die Summe der hineinfließenden Ströme gleich der Summe der herausfließenden Ströme ist. Mit anderen Worten: Die Summe der Ströme in einem Knotenpunkt einer elektrischen Schaltung ist gleich null.

$$\sum_{k=1}^{n} I_k = 0$$

Abbildung 3.26 zeigt einen Knotenpunkt mit den hineinfließenden Strömen I1 und I4 sowie den hinausfließenden Strömen *I2* und *I3* ($n = 4$).

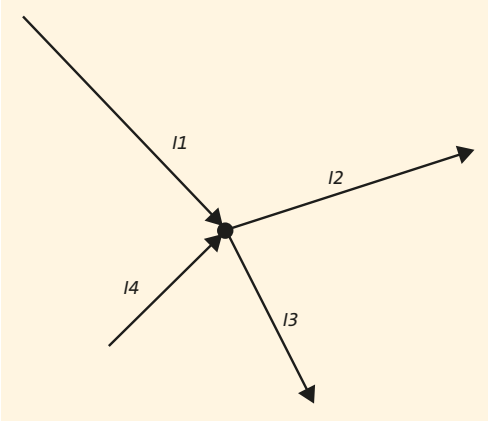

Abbildung 3.26 Knotenpunkt mit hinein- und hinausfließenden Strömen

Mit einem einfachen praktischen Beispiel zeige ich Ihnen die Anwendung des Knotenpunktsatzes. Abbildung 3.27 zeigt Ihnen noch einmal den invertierenden Verstärker aus Abbildung 3.24, diesmal aber etwas umgezeichnet.

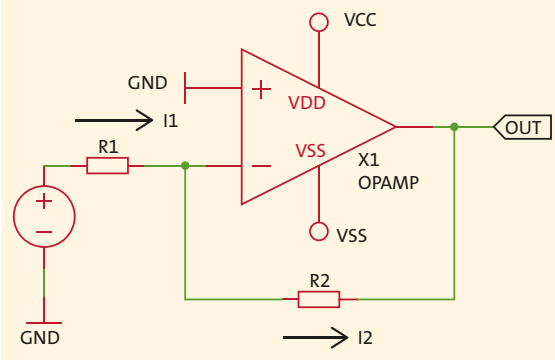

Abbildung 3.27 Invertierender Verstärker

Die Eingangsspannung U_{IN} treibt den Strom I_1 in den Knoten am Eingang (–) des Operationsverstärkers. Da der Eingangsstrom des Operationsverstärkers mit null angenommen wird, fließt nur der Strom I_2 aus dem Knoten heraus. Es gilt also $I_1 = I_2$ oder auch $I_1 - I_2 = 0$.

Jetzt ersetzen wir die Ströme dem ohmschen Gesetz folgend durch die Spannungsabfälle über dem jeweiligen Widerstand:

$$\frac{U_{IN}}{R1} = \frac{-U_{OUT}}{R2}$$

Das Minuszeichen vor der Ausgangsspannung steht deshalb, weil die Ausgangsspannung U_{OUT} nicht die gleiche Richtung wie der Strom I_2 hat. Wenn Sie die letzte Gleichung nun nach U_{OUT} umstellen, dann erhalten Sie die Gleichung aus Abschnitt 3.1.5, die das Übertragungsverhalten des invertierenden Operationsverstärkers beschreibt:

$$U_{OUT} = -\frac{R2}{R1} U_{IN}$$

Maschensatz

Der Maschensatz besagt, dass alle Spannungen in einem geschlossenen Stromkreis (*Masche*) gleich null sind:

$$\sum_{k=1}^{n} U_k = 0$$

Angewendet auf den in Abbildung 3.28 gezeigten Stromkreis, bedeutet das in Richtung der eingetragenen Orientierung für den Strom I:

$$U_1 - U_2 = I\,(R1 + R2)$$

$$I = \frac{U_1 - U_2}{R1 + R2}$$

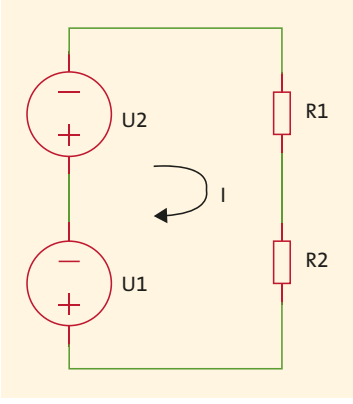

Abbildung 3.28 Maschensatz

Mit einem Beispiel möchte ich Ihnen wieder die Handhabung des Maschensatzes und anschließend die Kombination beider Regeln demonstrieren.

Abbildung 3.29 zeigt eine Ihnen bereits bekannte Schaltung – eine Inverterstufe mit einem npn-Transistor.

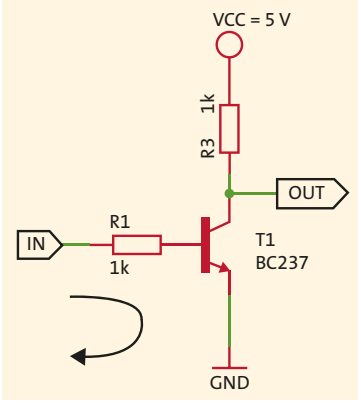

Abbildung 3.29 Inverterstufe

Wir betrachten den Eingangskreis in Richtung des eingezeichneten Pfeils und können nach dem Maschensatz folgende Gleichung aufstellen:

$$U_{\mathrm{IN}} = I_{\mathrm{B}} \times R1 + U_{\mathrm{BE}}$$

Der Basisstrom wird durch den Stromverstärkungsfaktor B_N und eine mögliche Übersteuerung m im Schalterbetrieb zu:

$$I_B = m \frac{I_C}{B_N}$$

Der Kollektorstrom I_C beträgt im durchgesteuerten Zustand des Transistors:

$$I_C = \frac{V_{\mathrm{CC}} - U_{\mathrm{CES}}}{R3}$$

Die Schwellenspannung U_{INS} für das Umschalten der Inverterstufe kann nun durch das Zusammenführen der drei Gleichungen berechnet werden:

$$U_{\mathrm{INS}} = \frac{m}{B_N} \frac{R1}{R3} (V_{\mathrm{CC}} - U_{\mathrm{CES}}) + U_{\mathrm{BE}}$$

Mit den Werten B_N = 100, m = 5 und U_{CES} = 0,1 V und den in Abbildung 3.29 angegebenen Widerstandswerten für R1 und R3 erhält man eine Schwellenspannung U_{INS} von 0,9 V.

Die Ausgangsspannung VOL des ATmega328P, die für den Lo-Pegel steht, liegt im Extremfall bei dem Wert der berechneten Schwellenspannung U_{INS} (siehe Tabelle 3.2).

Symbol	Parameter	Messbedingung	Wert
VOL	Output Low Voltage	IOL = 20 mA, VCC = 5 V	< 0,9 V
VOH	Output High Voltage	IOH = –20 mA, VCC = 5 V	> 4,2 V

Tabelle 3.2 ATmega328P – Spannungen am digitalen Ausgang

Das sind keine guten Voraussetzungen für ein sicheres Schaltverhalten der Inverterstufe. Durch Einfügen des Widerstands R2 (siehe Abbildung 3.30) soll die Schwellenspannung in einen sicheren Bereich verschoben werden.

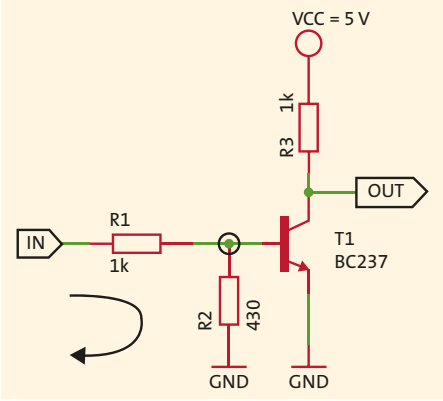

Abbildung 3.30 Erweiterte Inverterstufe

Um die Veränderung der Schwellenspannung U_{INS} zu berechnen, müssen wir den Knoten an der Basis des Transistors und den Eingangskreis betrachten. Die Formeln dazu kennen Sie bereits.

Mit dem Knotenpunktsatz erhält man:

$$I_1 = I_2 + I_B$$

Nach dem Maschensatz ergibt sich nun die folgende Änderung:

$$U_{\mathrm{INS}} = I_1 \times R1 + U_{\mathrm{BE}} = (I_2 + I_B) \times R1 + U_{\mathrm{BE}}$$

Mit den oben angegebenen Beziehungen für I_B und I_C und dem Strom $I_2 = U_{\mathrm{BE}}/\mathrm{R2}$ erhalten Sie nach der Umstellung der Formel folgende Beziehung:

$$U_{\mathrm{INS}} = \frac{m}{B_N} \times \frac{R1}{R3}(V_{\mathrm{CC}} - U_{\mathrm{CES}}) + \left(\frac{R1}{R2} + 1\right) \times U_{\mathrm{BE}}$$

Daraus können Sie für die Schwellenspannung U_{INS} einen Wert von $U_{\mathrm{INS}} = 2{,}4$ V berechnen.

Die Schwellenspannung für die an einen digitalen Ausgang angeschlossene Inverterstufe liegt nun in der Mitte des Spannungshubs und sollte damit wesentlich unempfindlicher gegenüber Störungen sein.

3.2.3 Reihen- und Parallelschaltung von Widerständen

Mit der Kenntnis des ohmschen Gesetzes und der kirchhoffschen Regeln können Sie nun auch die Reihen- und Parallelschaltung von Widerständen berechnen (siehe Abbildung 3.31).

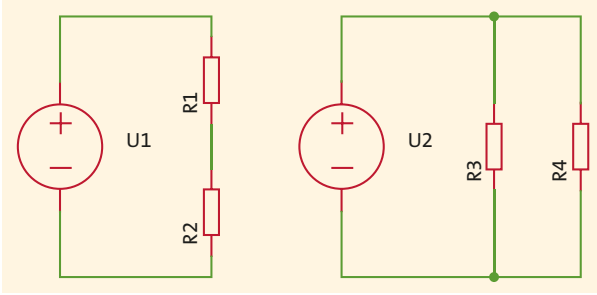

Abbildung 3.31 Reihen- und Parallelschaltung von Widerständen

Werden Widerstände in Reihe geschaltet, werden diese vom gleichen Strom durchflossen. Mit dem Maschensatz gilt die Beziehung $U_1 = U_{R1} + U_{R2} = I \times (R_1 + R_2)$ und für den Gesamtwiderstand dann $R = R_1 + R_2$.

Bei der Parallelschaltung von Widerständen liegt über allen Widerständen die gleiche Spannung an. Mit dem Knotenpunktsatz gilt:

$$I = \frac{U_2}{R3} + \frac{U_2}{R4}$$

Daraus folgt durch Umstellung:

$$\frac{I}{U_s} = \frac{1}{R} = \frac{1}{R3} + \frac{1}{R4}$$

Bei der Parallelschaltung von Widerständen addieren sich also deren Leitwerte ($G_i = 1/R_i$). Bei der Parallelschaltung von zwei Widerständen, wie oben angegeben, vereinfacht sich die Formel zu:

$$R = \frac{R3 \times R4}{R3 + R4}$$

Merke: Gesamtwiderstand

Der Gesamtwiderstand parallel geschalteter Widerstände ist immer kleiner als deren kleinster Widerstandswert.

So viel zu den theoretischen Grundlagen. Sie werden sie immer wieder heranziehen müssen, wenn es um konkrete Dimensionierungen von Bauteilen geht.

3.3 Breadboards und Zubehör

In den vorigen beiden Abschnitten haben Sie Schaltungsteile und die erforderlichen Grundlagen zu ihrer Berechnung kennengelernt. In diesem Abschnitt möchte ich Sie mit einigen empfehlenswerten Hardware-Erweiterungen bekannt machen.

Diese Hardware soll den Aufbau Ihrer Versuchsschaltungen oder Prototypen so einfach wie möglich machen. Nichts ist schlimmer, als einen Fehler in einem Drahtverhau zu suchen, dessen Ursachen möglicherweise an ganz anderer Stelle liegen.

3.3.1 Breadboards

Zum Aufbau von Versuchsschaltungen sind die sogenannten *Breadboards* (Steckbretter) sehr beliebt. Ich bleibe hier bei den originalen Begriffen, denn mit diesen können Sie entsprechende Angebote im Netz finden.

Die Breadboards sind in verschiedenen Größen bis hin zum Arduino Prototyping Shield zu haben. Abbildung 3.32 zeigt Ihnen Breadboards in verschiedenen Größen sowie deren Abmessungen.

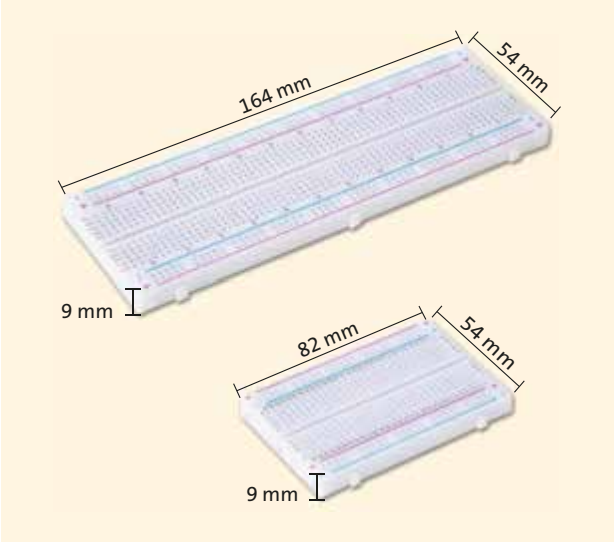

Abbildung 3.32 Breadboards verschiedener Größe

Allen Breadboards gemeinsam sind die Verbindungen der verschiedenen Lochraster. In Abbildung 3.33 habe ich einige verbundene Lochreihen farbig hervorgehoben.

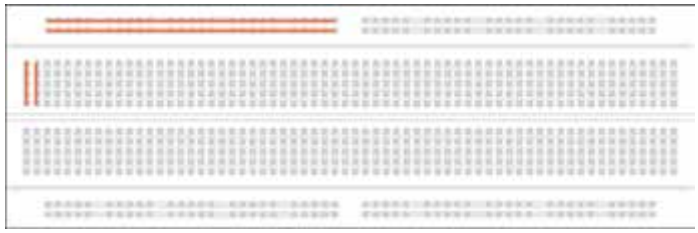

Abbildung 3.33 Verbundene Lochreihen

An der Ober- und Unterkante befinden sich Lochreihen, die der Zuführung der Betriebsspannung (+) und des Massepotenzials (–) dienen. Im mittleren Bereich sind die senkrechten Reihen verbunden, die dann die elektrischen Bauteile aufnehmen.

Die Breadboards weisen aber noch eine weitere wichtige Eigenschaft auf. Mechanisch ist dafür gesorgt, dass die Breadboards durch Zusammenstecken zu einer größeren Prototyping-Fläche kombiniert werden können. Abbildung 3.34 zeigt Ihnen verschiedene Kombinationsmöglichkeiten.

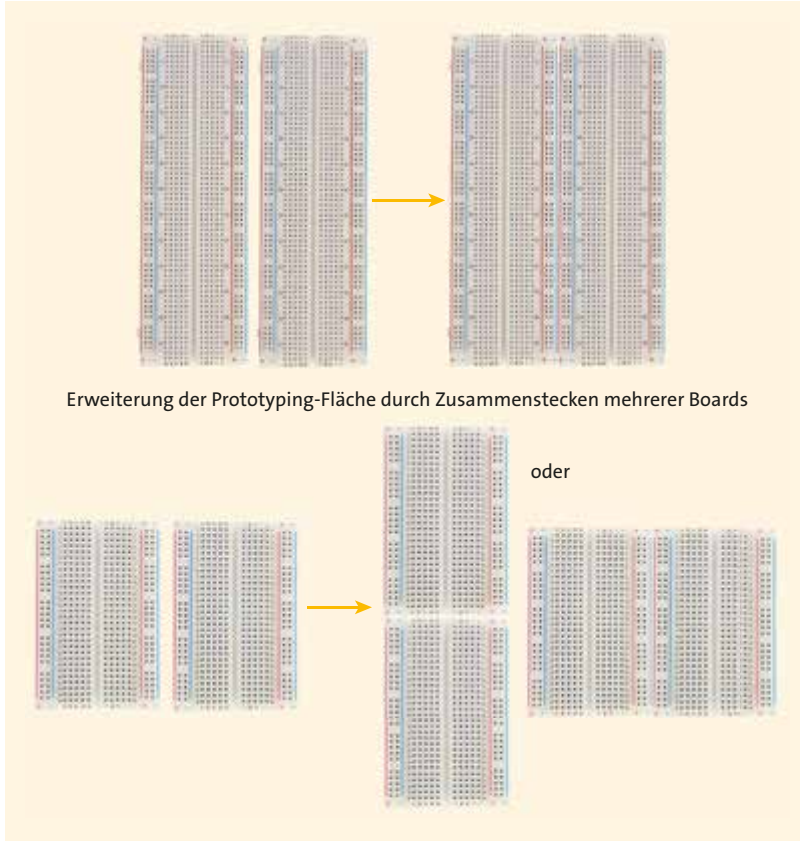

Abbildung 3.34 Kombination von Breadboards

Speziell für den Arduino gibt es noch ein sogenanntes *Prototype Shield* – eine Kombination aus einem Lochraster-Shield und einem kleinen Breadboard.

Abbildung 3.35 zeigt das *Prototype Shield v.5*, wie es von verschiedenen Lieferanten so oder in ähnlicher Form angeboten wird. Ich habe dieses Shield im Dezember 2022 für ca. 3 € erworben.

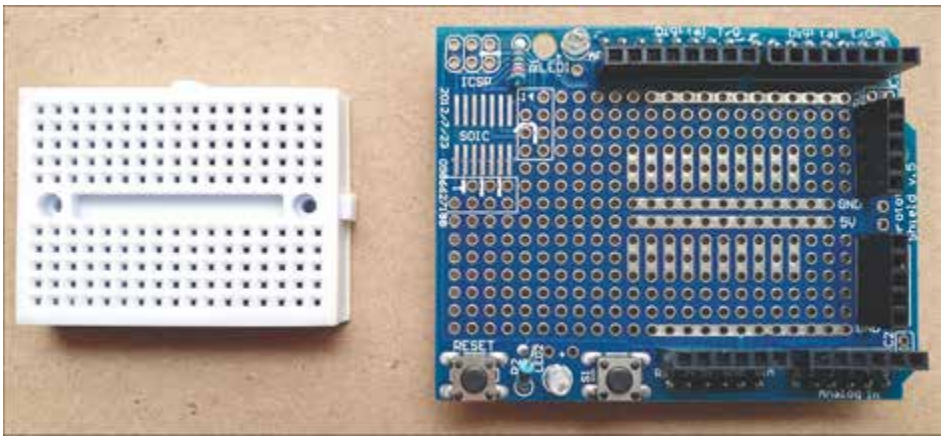

Abbildung 3.35 Prototype Shield v.5

Neben den Arduino-Steckerleisten weist das Shield zwei LEDs mit Vorwiderstand, zwei Taster (davon einer ein Reset) und einen Lochrasterbereich sowie einen Bereich zum Auflöten eines Bausteins im SOIC-Gehäuse auf.

Möchten Sie das Breadboard in Verbindung mit dem Shield nutzen, dann können Sie es mit dem dort bereits angebrachten doppelseitigen Klebeband (nach Lösen der Schutzfolie) auf das Prototype Shield aufkleben.

3.3.2 Breadboard Holder

Ein sehr einfacher Breadboard Holder (Steckbrett-Halter) entsteht bereits durch das Aufkleben des kleinen Breadboards auf das im vorigen Abschnitt vorgestellte Prototype Shield.

Abbildung 3.36 zeigt Ihnen diese Kombination, die ich gern für kleine Versuchsaufbauten nutze, z. B. für den Anschluss eines Sensor-Boards oder einiger LEDs. Die Kontakte des darunter liegenden Arduinos stehen auf dem Prototype Shield 1:1 zur Verfügung und können mit den auf dem Breadboard erstellten Schaltungsteilen einfach zusammengeschaltet werden.

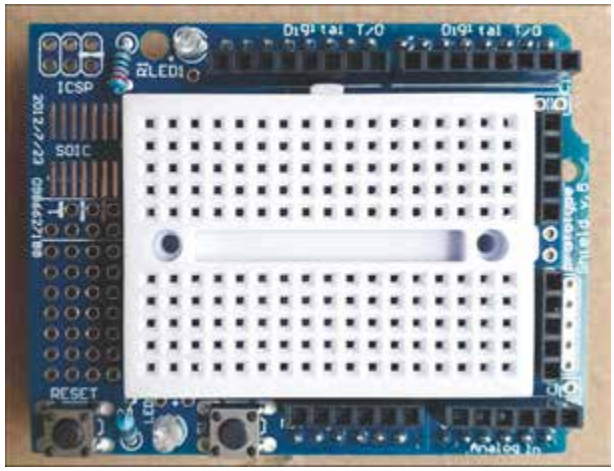

Abbildung 3.36 Prototype Shield v.5 mit aufgesetztem Breadboard

Benötigen Sie etwas mehr Platz auf dem Breadboard, dann können Sie den von SparkFun für 3,95 US$ angebotenen *Arduino and Breadboard Holder* (*https://www.sparkfun.com/products/11235*) einsetzen. Abbildung 3.37 zeigt diesen, bestückt mit einem Arduino Uno und einem passenden Breadboard.

Abbildung 3.37 Breadboard Holder (Steckbrett-Halter)

In eine ganz andere Liga steigen Sie auf, wenn Sie das von der italienischen Firma Gtronics angebotene und als *Arduino Protoshield* bezeichnete Board einsetzen (siehe Abbildung 3.38). Da das Board für rund 50 € angeboten wird, wird nicht jeder es sofort bestellen.

Dennoch sind die gebotenen Möglichkeiten vor allem für die Ausbildung sehr hilfreich, weshalb ich auch hier die URL des Gtronics-Shops (*https://www.gtronicsshop.com/en/12-protoshield-plus*) angebe. Dort finden Sie auch alle weiterführenden Informationen.

Abbildung 3.38 Arduino Protoshield (Quelle: Gtronics.NET)

3.3.3 Breadboard Power

Für funktionierende Schaltungsaufbauten auf dem Breadboard sind (bei den größeren Boards) an der Ober- und Unterkante Lochreihen angeordnet, die der Zuführung der Betriebsspannung (+) und des Massepotenzials (–) dienen. Damit diese Lochreihen auch die gewünschten Spannungen bereitstellen, verwenden Sie am einfachsten sogenannte *Breadboard Power Adapter*. Diese werden in unterschiedlichen Bauformen von vielen Distributoren zu Preisen von unter 5 € angeboten. Abbildung 3.39 zeigt einen auf ein Breadboard aufgesteckten Adapter. Die Spannungsversorgung kann über Micro-USB ❶ oder über einen Rundstecker ❷ erfolgen. Es lassen sich 3,3 V oder 5 V Betriebsspannung für die beiden Lochreihen unabhängig voneinander erzeugen.

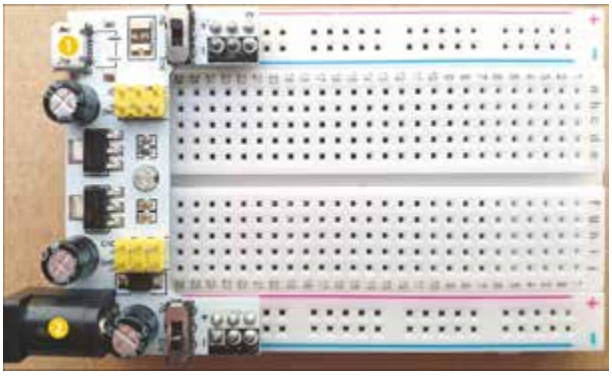

Abbildung 3.39 Breadboard Power Adapter

3.4 Qwiic, Grove und mikroBUS Connection

In diesem Abschnitt stelle ich Ihnen Hardware-Erweiterungen vor, mit denen sich das Prototyping auf Arduino-Basis wirkungsvoll vereinfachen lässt. Neben den in Abschnitt 2.3 vorgestellten Arduino-Shields gibt es die Prototyping-Systeme *Qwiic* von SparkFun, *Grove* von Seeed Studio und *mikroBUS* von Mikroelektronika.

In diesem Abschnitt lernen Sie, wie Sie mit diesen Systemen Ihre Prototypen schnell erstellen und neue Funktionen austesten können.

3.4.1 Qwiic Connection

SparkFun Qwiic Connection ist ein System aus über den I^2C-Bus verbundenen Sensoren, Aktoren, Shields, Boards und passenden Kabeln, die das Prototyping beschleunigen und weniger fehleranfällig machen. Alle Qwiic-kompatiblen Boards verwenden einen vierpoligen JST-Anschluss mit 1 mm Rastermaß. Dadurch wird weniger Platz auf dem jeweiligen Board benötigt. Durch die Polarisierung der Anschlüsse können Sie keine Anschlussfehler begehen (*https://www.sparkfun.com/qwiic*).

Um einen Überblick über das Qwiic-System zu bekommen, können Sie die Qwiic-Seite von SparkFun *https://www.sparkfun.com/qwiic#products* konsultieren. Ich zeige Ihnen in Abbildung 3.40 ein BME280-Breakout-Board. An der linken und an der rechten Seite können Sie die Qwiic-Konnektoren (JST) sehen.

Breakout Board

Als Breakout Boards werden Leiterplatten bezeichnet, die elektronische Bauteile tragen, deren Anschlüsse nach außen geführt sind. Das ist häufig die einzige Möglichkeit im Prototypenbereich, um Bauteile mit kompakten Gehäusen überhaupt zu kontaktieren. Außerdem wäre man selbst technisch kaum in der Lage, diese Bauteile auf einer Leiterplatte zu montieren bzw. zu löten. Das in Abbildung 3.40 gezeigte Breakout Board mit dem BME280-Sensor-IC ist hierfür ein sehr gutes Beispiel.

Abbildung 3.40 BME280 Qwiic Breakout (Quelle: SparkFun)

Durch die Gestaltung der Boards des Qwiic-Systems lassen sich komplexe, örtlich verteilte Schaltungen aufbauen. Abbildung 3.41 zeigt eine solche Zusammenschal-

tung von Mikrocontroller (RedBoard Qwiic) und verschiedenen Qwiic-Breakout-Boards (*GPS Breakout – XA1110*, *OpenLog*, *Transparent Graphical OLED Breakout*, *Environmental Combo Breakout*).

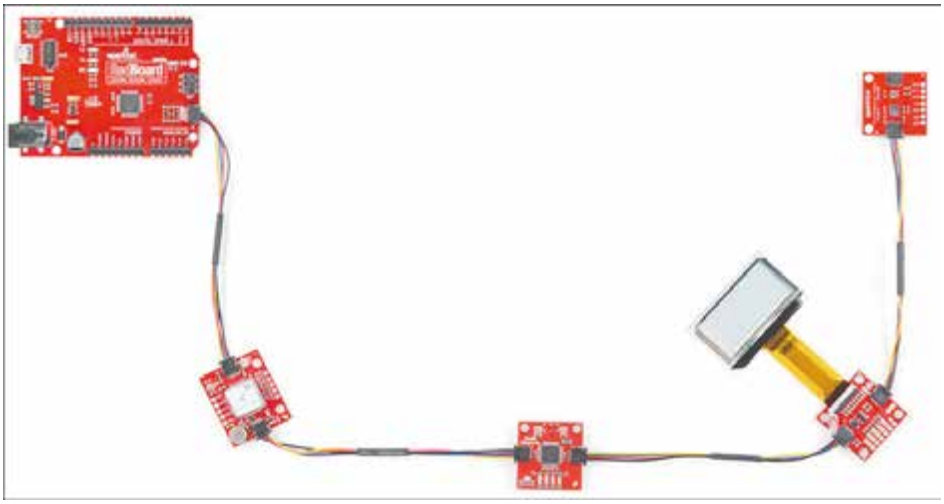

Abbildung 3.41 Zusammenschaltung von Qwiic-Boards (Quelle: Sparkfun)

Wie Sie schon lesen konnten, wurde der I^2C-Bus von Philips (heute NXP) für die Datenübertragung zwischen unterschiedlichen Bausteinen, wie EEPROMs, RAMs, AD- und DA-Umsetzern, RTCs und vielen mehr, und Mikrocontrollern auf einer Leiterplatte entwickelt. Der I^2C-Bus ist nicht für lange Strecken ausgelegt. Das Protokoll erlaubt die Verbindung von bis zu 128 unterschiedlichen Bausteinen (Devices) mithilfe einer Zwei-Draht-Leitung.

Eine sichere Übertragung über den I^2C-Bus bei größeren Distanzen ist nur mit differenziellen Signalen möglich. Die schnellste und einfachste Möglichkeit, um die Reichweite des I^2C-Busses zu erweitern, ist die Verwendung des *Differential I2C Breakout Board* (siehe Abbildung 3.42).

Abbildung 3.42 SparkFun Differential I2C Breakout – PCA9615 (Qwiic)

Das Board verwendet ein PCA9615-IC von NXP, das die beiden Standard-I²C-Signale SCL und SDA in vier Differenzsignale umwandelt: zwei für SCL und zwei für SDA. Die Differenzsignale werden über ein Ethernet-Kabel übertragen, das über die integrierten RJ-45-Anschlüsse an das Board angeschlossen wird. SparkFun gibt an, dass die differenzielle Signalübertragung Entfernungen von bis zu 300 m ermöglicht.

3.4.2 Grove-System

Das *Grove-System* ist ein modulares Open-Source-Prototyping-System mit standardisierten Steckverbindern.

Das Grove-System besteht aus einem *Base Shield* und verschiedenen Modulen. Ein Base Shield ermöglicht den einfachen Anschluss aller Mikrocontrollereingänge oder -ausgänge der Grove-Module. Jedes Grove-Modul erfüllt eine einzelne Funktion, z. B. eine einfache Taste oder einen komplexen Sensor. Eine Übersicht über die mehr als 300 Grove-Module finden Sie unter *https://www.seeedstudio.com/category/Grove-c-1003.html*.

Abbildung 3.43 zeigt das *Grove Base Shield* für den Arduino. Ein Schiebeschalter (1) ermöglicht die Wahl der Betriebsspannung in Abhängigkeit vom eingesetzten Arduino. Auf Grove-Konnektoren sind die Anschlüsse A0 bis A3, D2 bis D8 sowie UART und viermal I²C herausgeführt.

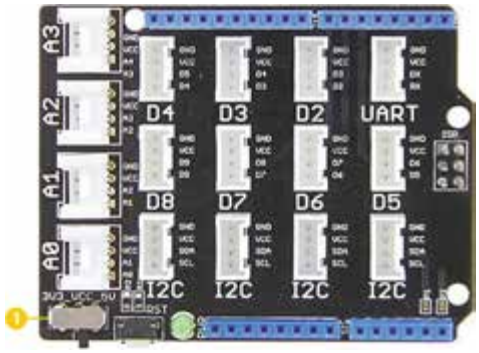

Abbildung 3.43 Grove Base Shield

Ich werde an mehreren Stellen dieses Buches auf den Einsatz von Grove-Modulen zurückkommen. Der Schaltungsaufbau wird dadurch wesentlich vereinfacht, und die Grove-Module selbst sind durch den Hersteller gut dokumentiert.

3.4.3 mikroBUS System

mikroBUS versteht sich als Standard für Zusatzkarten, der maximale Erweiterbarkeit mit der geringsten Anzahl von Pins bietet.

Kontaktiert über einen mikroBUS-Sockel, lassen sich sogenannte *Clickboards* als Schaltungserweiterung nutzen. Sensoren, drahtlose Transceiver, Audioverstärker, LED-Anzeigen und anderes mehr stehen als Clickboard zur Verfügung. Eine Übersicht über die Komponenten des mikroBUS-Systems der Belgrader Firma MikroElektronika (*mikroe*) finden Sie unter *https://www.mikroe.com/mikrobus*.

Um Clickboards für die Schaltungserweiterung einsetzen zu können, kann ein Arduino mit einem *Arduino Uno Click Shield* verbunden werden (siehe Abbildung 3.44). Damit stehen für die Erweiterung zwei mikroBUS-Steckplätze zur Verfügung.

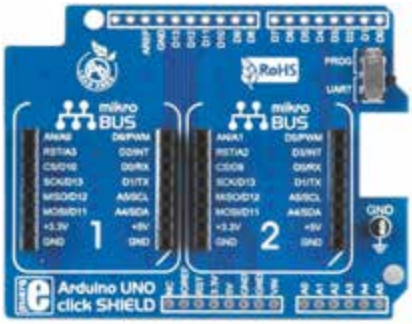

Abbildung 3.44 Arduino Uno Click Shield

Librarys und Demoprogramme, die mit MikroElektronika-Compilern entwickelt wurden, sind auf der mikroe-Website zu finden. Die MikroElektronika-Compiler sind allerdings nicht Arduino-kompatibel!

Für den Betrieb am Arduino hat die französische Firma *Lextronics* eine Reihe von Programmbeispielen für zahlreiche Clickboards erstellt. Unter der URL *https://www.mikroe.com/blog/-arduino-libraries-for-click-boards* finden Sie die Programmbeispiele sowie Schaltbilder im Fritzing-Format. Das CAD-Programm Fritzing stelle ich Ihnen in Abschnitt 3.7.1 noch im Detail vor. Den Hinweisen von Lextronics folgend, können Sie später auch selbst Programme für weitere Clickboards erstellen.

3.5 Spannungsversorgung

Für die Spannungsversorgung der Arduino-Boards gibt es in der Regel mehrere Möglichkeiten. Zum einen kann der eingesetzte Mikrocontroller direkt mit der betreffenden Betriebsspannung versorgt werden, was eine extern geregelte Versorgungsspannung von 3,3 V oder 5 V erfordert.

Hat das betreffende Arduino-Board eine Spannungsregelung on-board, dann kann über den Anschluss VIN oder eine separate Buchse der Arduino mit einer höheren Spannung versorgt werden. Die einfachste Variante ist aber, das Arduino-Board über den USB-Anschluss mit Spannung zu versorgen.

USB-Anschluss

Mit Ausnahme des Lilypad Arduino haben alle hier vorgestellten Arduinos einen USB-Anschluss, der zum Programm-Upload dient und zur Spannungsversorgung verwendet werden kann.

Die auf den Arduinos verbauten Stecker sind verschieden, weshalb Sie sich mit unterschiedlichen USB-Kabeln ausrüsten sollten. Meist hat man diese aber ohnehin von bereits gekauften Geräten im Haus.

Beim Arduino Uno finden Sie beispielsweise eine USB-Buchse vom Typ B, die ein Kabel mit einem Stecker gemäß Abbildung 3.45 verlangt. Diese Kabel werden oft für den Anschluss eines Druckers an einen PC verwendet.

Abbildung 3.45 USB-Stecker vom Typ B

Die kompakteren Arduino-Boards verlangen auch nach kleineren Steckverbindern, und so hat beispielsweise der Arduino Nano eine Mini-USB-Buchse, die ein Kabel mit dem in Abbildung 3.46 gezeigten Mini-USB-Stecker erfordert. Diese Steckverbinder sind häufig bei Digitalkameras zu finden.

Abbildung 3.46 Mini-USB-Stecker

Beim Arduino Micro und vielen anderen Arduinos wird eine Micro-USB-Buchse eingesetzt, die ein Kabel mit dem in Abbildung 3.47 gezeigten Micro-USB-Stecker erfordert. Diese Stecker werden oft bei älteren Smartphones und Tablets eingesetzt.

Abbildung 3.47 Micro-USB-Stecker

Gerade bei aktuellen Arduino-kompatiblen Boards kommt zunehmend eine USB-Buchse vom Typ C zum Einsatz. Diese Buchse erfordert ein Kabel, das mit einem USB-Stecker vom Typ C (siehe Abbildung 3.48) versehen ist.

Abbildung 3.48 USB-Stecker vom Typ C

USB Typ C ist ein 2015 eingeführtes Steckerformat. Diese Stecker sind kleiner und von beiden Seiten verwendbar. Sie müssen also nicht mehr darauf achten, wie herum der Stecker in die Buchse gesteckt werden muss. Laptops, Tablets und Smartphones weisen diesen Stecker auf.

Wenn man ein solches Kabel nicht zur Hand hat, dann kann man auch einen Adapter von Typ C auf Micro-USB verwenden. Abbildung 3.49 zeigt einen solchen Adapter, der von verschiedenen Herstellern angeboten wird.

Abbildung 3.49 Adapter »USB Typ C auf Micro-USB« von Xystec

So viel zu der Hardware, mit der Sie den Anschluss vornehmen können.

Zur Spannungsversorgung müssen Sie wissen, dass USB 1.x und 2.0 nur bis zu 500 mA und USB 3.0 bis zu 900 mA Strom bereitstellen. Tabelle 3.3 zeigt die Übertragungsgeschwindigkeit und Stromversorgung der verschiedenen USB-Standards im Vergleich.

Standard	USB 1.0	USB 2.0	USB 3.0	USB 3.1
max. Übertragungsgeschwindigkeit	12 Mbit/s	480 Mbit/s	5 Gbit/s	10 Gbit/s
max. Stromversorgung	500 mA	500 mA	900 mA	5 A

Tabelle 3.3 Übertragungsgeschwindigkeit und Stromversorgung via USB-Anschluss

Zur Inbetriebnahme eines Arduinos können Sie also ohne Bedenken die Spannungsversorgung über USB vornehmen. Erst wenn externe Lasten oder drahtlose Kommunikation hinzukommen, kann es zu einer Überlastung des USB-Ports durch zu hohe Stromaufnahme kommen.

In gleicher Weise wirken sich zu lange USB-Kabel mit geringem Drahtquerschnitt aus: Am Serienwiderstand des Kabels bewirkt ein großer Strom einen Spannungsabfall, der die Betriebsspannung am USB-Port des Arduinos herabsetzt.

Wenn in der Folge dann die Versorgungsspannung (durch einen Spike kurzzeitig) zusammenbricht bzw. vom USB-Port unterbrochen wird, stürzt das laufende Programm des angeschlossenen Arduinos ab oder weist ein sehr merkwürdiges Verhalten auf.

Solche Fehler sind nur mit erheblichem messtechnischem Aufwand lokalisierbar. Beziehen Sie den Stromverbrauch einzelner Komponenten deshalb von vornherein in die Betrachtungen mit ein, und legen Sie die Stromversorgung eher großzügig aus.

Zwei Beispiele mit Komponenten, die ich später noch verwenden werde, sollen das verdeutlichen:

- **Mikrocontroller ESP8266, ESP32** – Stromaufnahme 50 bis 70 mA, Stromspitzen im WiFi-Mode 400 mA und mehr (sehr lesenswert dazu: *https://arduino-hannover.de/2018/07/25/die-tuecken-der-esp32-stromversorgung/*)
- **GSM/GPRS Module SIM800L** – Stromaufnahme (Data Mode (1 Tx, 4 Rx), GSM850) = 453.57 mA (typ. Mittelwert), Spitzenströme bis 2 A! (*https://datasheet-pdf.com/PDF/SIM800L-Datasheet-SIMCom-989664*)

3.5.1 USB-Hub mit Schnellladeanschluss

Können Sie in stromhungrigen Anwendungen auf die Kommunikation über USB verzichten, dann können Sie die Verbindung zum PC durch ein USB-Ladegerät ersetzen. Ein USB-Ladegerät liefert bei richtiger Wahl dann den benötigten Strom.

Da die Kommunikation nicht nur für das Debugging wichtig ist, stellt der Vorschlag oben nur eine Möglichkeit für einige wenige Anwendungsfälle dar.

Abhilfe kann die Verwendung eines speziellen USB-Hubs schaffen, der, durch ein eigenes Netzteil gespeist, einen sogenannten *Schnellladeanschluss* aufweist. Der *D-Link DUB-H4* ist ein solcher USB-Hub mit vier Downstream-USB-Anschlüssen vom Typ A (Buchse) inklusive eines Schnellladeanschlusses, der bei Anschluss des zugehörigen Netzteils bis zu 2 A liefern und den erforderlichen Strombedarf der meisten Anwendungen decken kann, ohne dass Sie auf die Kommunikation mit dem seriellen Port verzichten müssen (*https://eu.dlink.com/de/de/products/dub-h4-4-port-usb-2-hub*). Abbildung 3.50 zeigt den USB-Hub mit seinem Schnellladeanschluss.

Abbildung 3.50 D-Link DUB-H4 mit Schnelladeanschluss (markiert)

3.5.2 Steckernetzteil

Wollen Sie den Arduino und eine möglicherweise stromhungrige Zusatzschaltung über ein Steckernetzteil mit Spannung versorgen, dann sollten Sie die Spannung des Steckernetzteils nicht zu hoch wählen.

Die Spannungsdifferenz zwischen der angelegten Spannung VIN und der Betriebsspannung von 5 V (beim Arduino Uno) wird, multipliziert mit dem aufgenommenen Strom am Längsregler, in Wärme umgesetzt. Außerdem ist der Strom beim Arduino Uno durch den Spannungsregler MC33269D-5.0 auf 800 mA begrenzt. Abbildung 3.51 zeigt in einem Schaltungsausschnitt den Längsregler beim Arduino Uno.

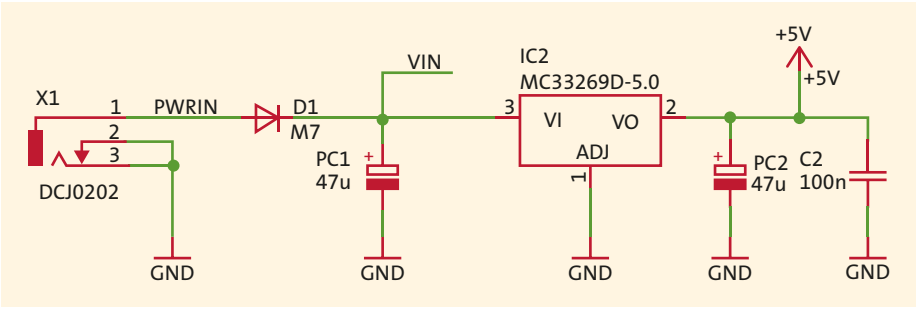

Abbildung 3.51 Längsregler beim Arduino Uno

Kapitel 7
Anzeigeelemente

In diesem Kapitel stelle ich Ihnen gebräuchliche Bauteile zur visuellen Anzeige von Informationen und deren Ansteuerung vor. Schaltungs- und Programmbeispiele verdeutlichen den jeweiligen Einsatz.

Nachdem Sie gesehen haben, wie Daten eingegeben und verarbeitet werden, wollen Sie bestimmt auch wissen, wie sie dargestellt werden können. Dazu stelle ich Ihnen in diesem Kapitel verschiedene Anzeigeelemente vor.

7.1 LEDs und RGB-LEDs

LEDs, also Licht emittierende Dioden, begleiten uns als Anzeigeelemente im täglichen Leben. In Abschnitt 3.1.3 konnten Sie sich über die physikalischen Eigenschaften der Dioden allgemein, aber auch über die der LEDs informieren. Für die Anwendung, auf die wir uns hier konzentrieren, bleiben nur wenige Aspekte zu beachten. Um eine LED zum Leuchten zu bringen, ist diese im *Durchlassbereich* zu betreiben.

Um die Farben der LEDs zu realisieren, werden verschiedene Halbleiter verwendet. Dadurch ändern sich nicht nur die elektrischen Parameter, sondern auch die Lichtstärke der LED. In Tabelle 7.1, die einen Auszug aus einer unter *https://www.reichelt.de/reicheltpedia/index.php/LED* veröffentlichten Tabelle darstellt, sind die wichtigsten Farben und elektrischen Parameter dargestellt:

LED-Farbe	Farbe	LED-Strom	LED-Flussspannung
Amber		40 mA	2 V
Blau		30 mA	3,3 V
Gelb		20 mA	2,1 V
Grün		20 mA	2,1 V

Tabelle 7.1 Farben und elektrische Parameter von LEDs

LED-Farbe	Farbe	LED-Strom	LED-Flussspannung
RGB	rot grün blau	20 mA	R = 2 V G = 2,2 V B = 4 V
Rot		20 mA	2,3 V
Pink		20 mA	3,6 V
Türkis		20 mA	3,8 V
Ultraviolett	nicht sichtbar	20 mA	3,8 V
Violett		20 mA	3,6 V
Weiß		20 mA	3,6 V

Tabelle 7.1 Farben und elektrische Parameter von LEDs (Forts.)

An Tabelle 7.1 erkennen Sie, dass bei vergleichbarem Strom die *Flussspannungen* recht unterschiedlich sind.

Tabelle 7.2 zeigt die unterschiedlichen Flussspannungen anhand eines Auszugs aus dem Datenblatt für eine RGB-LED von *Knightbright*.

Parameter	Symbol	Emitting Color	Value		Unit
			Typ.	Max.	
Forward Voltage @ IF = 20 mA	VF	Hyper Red	1,9	2,5	V
		Blue	3,3	4	
		Green	3,3	4,1	

Tabelle 7.2 Flussspannungen für Knightbright Full Color LED Lamp WP154A4SUREQBFZGC (Quelle: Auszug aus dem Datenblatt)

Bei der Berechnung des jeweiligen LED-*Vorwiderstands* sind diese Unterschiede und die Art der Anschaltung an den Arduino zu berücksichtigen. Abbildung 7.1 zeigt die beiden Möglichkeiten, wie Sie eine LED mit Vorwiderstand mit einem Arduino Uno verbinden.

Ist der Ausgang D2 Hi, dann fließt Strom durch LED1 und R1, und LED1 wird leuchten. LED2 leuchtet, wenn der Ausgang D3 Lo-Pegel aufweist. Welche Variante Sie zur Steu-

erung von LEDs einsetzen, ist Geschmackssache. In den Anwendungen finden Sie beide.

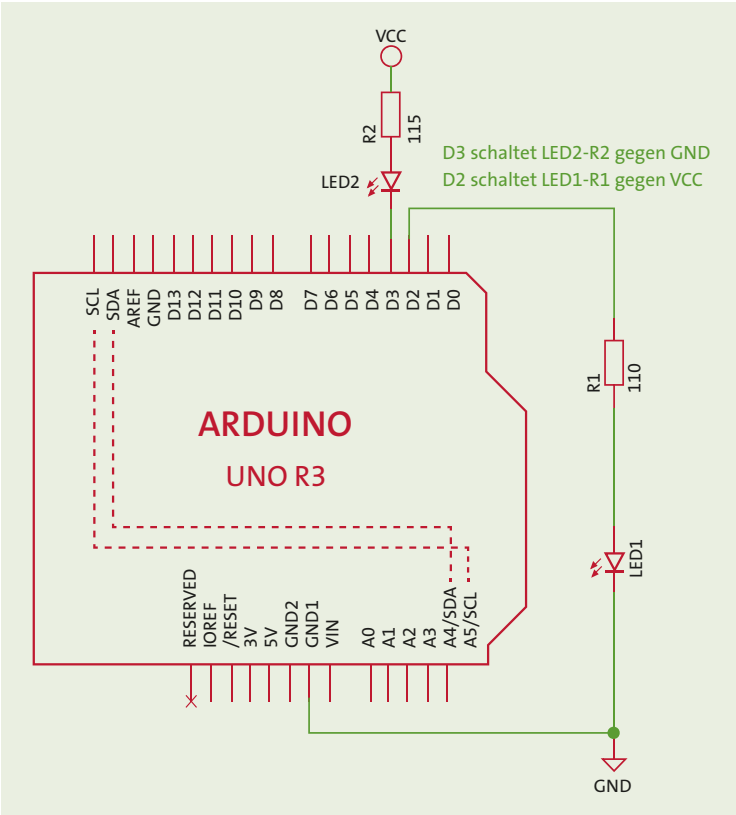

Abbildung 7.1 LEDs am Arduino Uno

Für die Berechnung des Vorwiderstands gelten die folgenden Beziehungen:

$$R2 = \frac{V_{CC} - V_F - V_{OL}}{I_F}$$

$$R1 = \frac{V_{OH} - V_F}{I_F}$$

V_{CC} steht für die Betriebsspannung, V_F für die Flussspannung der jeweiligen LED, V_{OL} für die Restspannung (Lo) bzw. V_{OH} für die maximale Ausgangsspannung (Hi) am Ausgangspin des Arduino sowie I_F für den Strom durch die LED.

Wenn wir mit den typischen Flussspannungen für die verschiedenen emittierten Farben aus Tabelle 7.2 rechnen, erhalten wir die Werte für die Vorwiderstände der LEDs aus Tabelle 7.3.

Farbe	Vorwiderstand		Bedingung
	R1	R2	
Rot	115 Ω	110 Ω	V_{CC} = 5 V, I_F = 20 mA, V_{OL} = 0,8 V, V_{OH} = 4,1 V
Grün	45 Ω	40 Ω	
Blau	45 Ω	40 Ω	

Tabelle 7.3 Vorwiderstände für RGB-LED

Die konkreten Widerstandswerte erscheinen sehr gering. Sie müssen aber bedenken, dass bei einem Strom von 20 mA durch die LED bei den einzelnen LEDs des betrachteten Bauteils eine Lichtstärke von typischerweise 1200 mcd (Millicandela) für Rot, 6500 mcd für Grün und 2000 mcd für Blau erzeugt wird. Zu Vergleichszwecken: Eine Kerzenflamme weist 1000 mcd auf, daher auch der Name der Maßeinheit *Candela* (lat. für *Kerze*).

Aus den gegebenen Zahlen resultiert, dass die Helligkeit dieser RGB-LED bei einem Strom von 20 mA recht hoch und sehr unterschiedlich für die einzelnen Farben ist. Um eine abgestimmte Helligkeit der einzelnen Farben zu erreichen, müssen Sie also die Vorwiderstände noch experimentell anpassen oder aber die Helligkeit über die Software beeinflussen.

Für die folgenden Programmbeispiele benutze ich die in Abbildung 7.2 gezeigte Schaltung mit einer blauen und einer roten LED sowie einer RGB-LED.

Wenn Sie den Schaltungsaufbau vermeiden wollen und im Besitz eines *YwRobot EasyModule Shields* (vorgestellt in Abschnitt 2.3.1) sind, dann können Sie dieses einfach auf den Arduino Uno aufstecken, und schon sind Sie bereit.

Das Programm *Uno_RGB.ino* zeigt die Ansteuerung der einzelnen LEDs. Die Zuordnung der LEDs zu den einzelnen Pins wird durch eine Enumeration vorgenommen:

```
enum{Red = 9, Green, Blue, SingleRed, SingleBlue};
```

In der Funktion `setup()` können Sie diese Pins dann als Ausgang konfigurieren und auf Low setzen. Für die rote der RGB-LEDs sieht das folgendermaßen aus:

```
pinMode(Red, OUTPUT); digitalWrite(Red,LOW);
```

In der Funktion `loop()` werden dann Blink- und Fade-Funktionen aufgerufen. *Fade* ist das langsame Hell- und anschließend langsame Dunkelwerden einer LED. Für die rote der RGB-LEDs geschieht das durch die folgenden Funktionen:

```
blinkLED(Red);
fadeLED(Red);
```

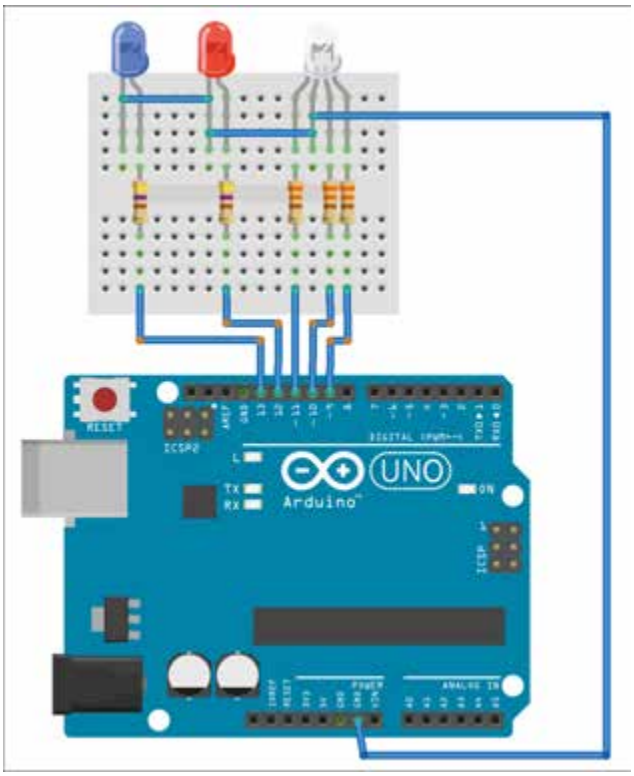

Abbildung 7.2 LEDs an Arduino Uno

Die Funktion `blinkLED(int RGB)` schaltet die betreffende LED für eine Sekunde ein und anschließend für eine Sekunde aus. Hier wird das durch `digitalWrite()` und `delay()` vorgenommen:

```
void blinkLED(int RGB)
{
  digitalWrite(RGB, HIGH);
  delay(1000);
  digitalWrite(RGB, LOW);
  delay(1000);
}
```

Das Fading erfolgt über die Ansteuerung der LED mit einem PWM-Signal. Dabei bedeuten `analogWrite(pin, 0)` »keinen Stromfluss durch die LED«, `analogWrite(pin, 127)` »halber Strom bzw. halbe Helligkeit« und `analogWrite(pin, 255)` »maximaler Stromfluss und höchstmögliche Helligkeit (nur begrenzt durch den Vorwiderstand)«.

```
void fadeLED(int RGB)
{
  int i = 0;
  while (i <= 255)
  {
    analogWrite(RGB, i);
    delay(10);
    i += 5;
  }
  i = 255;
  while (i >=0)
  {
    analogWrite(RGB, i);
    delay(10);
    i -= 5;
  }
  delay(1000);
}
```

Im Repository finden Sie unter der URL *https://github.com/ckuehnel/Arduino2023/tree/main/Arduino_Uno/Uno_RGB* ein Video, das die sichtbaren Unterschiede zwischen der Blink- und der Fade-Funktion zeigt.

Des Weiteren finden Sie im Repository das Programm *Uno_RGB_adjust.ino*, mit dem Sie die Helligkeit der RGB-LEDs abgleichen können. Das Programm ist recht schlicht und besteht im Wesentlichen aus verschiedenen Aufrufen von `analogWrite()` und der seriellen Eingabe von Helligkeitswerten. Im Screenshot aus Abbildung 7.3 sehen Sie die für den Abgleich erforderlichen Schritte.

Abbildung 7.3 Abgleich der Helligkeiten einer RGB-LED

Die blaue LED hat unter den gegebenen Voraussetzungen gleicher LED-Vorwiderstände die geringste Helligkeit. Deshalb wird sie auf das Maximum gesetzt (`analogWrite(Blue, 255)`),

und die beiden anderen LED werden durch einen kleineren PWM-Wert daran angepasst. Der Wert wird vom Serial Monitor aus eingegeben und über die serielle Schnittstelle an den Arduino gesendet.

Bei der roten LED habe ich nach visueller Einschätzung bei einem Wert von 170 und bei der grünen LED bei einem Wert von 40 einen vergleichbaren Helligkeitswert erreicht. Durch die Eingabe eines Wertes von über 255 wird die jeweilige Helligkeitseinstellung dann verlassen. Nach Beendigung der beiden Anpassungen blinken die einzelnen LEDs nacheinander. So kann die Anpassung der Helligkeit visuell verifiziert werden.

Sind Sie mit den Einstellungen noch nicht zufrieden, dann starten Sie das Programm erneut und wiederholen die beiden Helligkeitsanpassungen.

7.2 Sieben-Segment-Anzeige

Sieben-Segment-Anzeigen dienen zur Anzeige von zumeist numerischen Daten, weil die Anzeige von Text nur sehr beschränkt möglich ist.

Eine recht ausführliche Einführung in diesen Anzeigentyp finden Sie unter *https://www.mikrocontroller.net/articles/AVR-Tutorial:_7-Segment-Anzeige*. Ich möchte Sie deshalb hier stärker auf die Anwendungsaspekte lenken.

Mehrstellige Anzeigen werden in der Regel aus mehreren Einzelelementen zusammengesetzt. Abbildung 7.4 zeigt links einen zweistelligen Sieben-Segment-Anzeigebaustein und rechts ein Breakout Board mit einer vierstelligen Sieben-Segment-Anzeige.

Abbildung 7.4 Sieben-Segment-LED-Displays

Um die Zahl der Anschlüsse zu reduzieren, wird auf einem solchen Breakout Board ein Treiberschaltkreis vorgesehen, der eine serielle Datenverbindung zum steuernden Mikrocontroller und eine Kaskadierung von Anzeigen ermöglicht. Beim hier dargestellten Sieben-Segment-Display sorgt ein *TM1637* für die serielle Ansteuerung des Displays, wodurch neben VCC und GND nur noch zwei serielle Leitungen CLK und DIO für die Übergabe der Daten erforderlich sind. Abbildung 7.5 zeigt die erforderlichen Verbindungen von Sieben-Segment-Display (mit TM1637) und Arduino Uno.

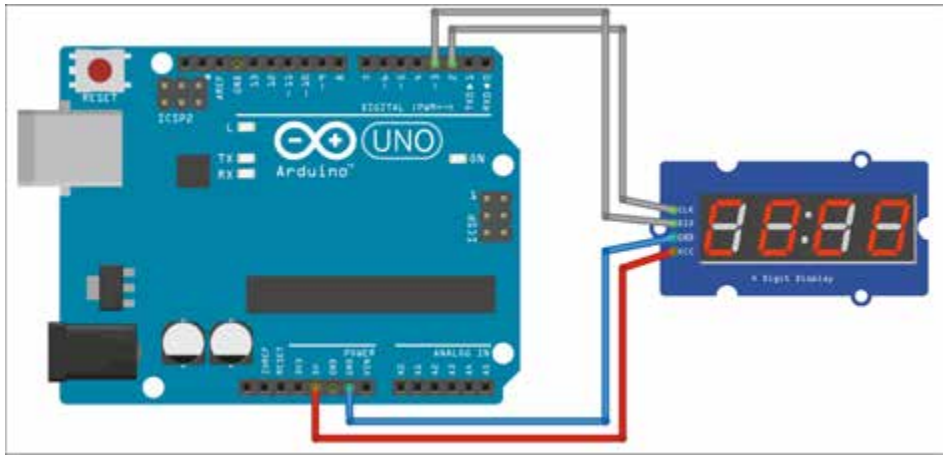

Abbildung 7.5 Sieben-Segment-Anzeige mit TM1637 am Arduino Uno

Außer numerischen Daten lassen sich mit einer Sieben-Segment-Anzeige bei etwas Fantasie auch alphanumerische Zeichen ausgeben. Einen Auszug aus den Möglichkeiten der Darstellung zeigt Abbildung 7.6 (*https://github.com/dmadison/LED-Segment-ASCII*).

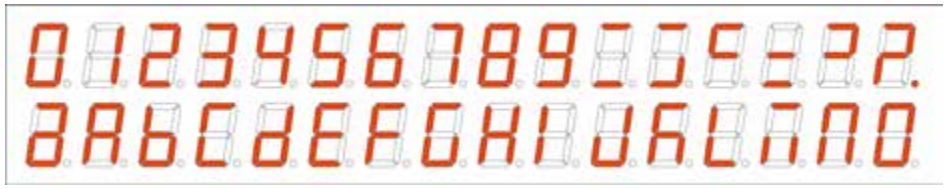

Abbildung 7.6 Alphanumerische Anzeige mit einem Sieben-Segment-Display

Es gibt zahlreiche Librarys, die den Treiberbaustein TM1637 unterstützen. Gefallen hat mir die Library *TM1637Display*, die Sie auf GitHub unter *https://github.com/avishorp/TM1637* finden: Sie ermöglicht eine komfortable Programmierung. Diese Library muss als Zip-Archiv eingebunden werden, da sie zum aktuellen Zeitpunkt nicht im Library Manager verfügbar war.

Als Programmbeispiel ist in der Library die Datei *TM1637Test.ino* enthalten, die verschiedene Anzeigeformate sehr gut erklärt. Mit dem Programm *Uno_TM1637.ino* zeige ich Ihnen die Anzeige von dezimalen und hexadezimalen Zahlen.

Nachdem Sie die Library *TM1637Display* eingebunden haben, müssen Sie den seriellen Leitungen CLK und DIO die zu verwendenden I/O-Pins zuordnen, bevor das Display-Objekt erstellt werden kann:

```
#include <TM1637Display.h>

// Module connection pins (Digital Pins)
#define CLK 2
#define DIO 3

TM1637Display display(CLK, DIO);
```

Der Schriftzug dEC_ kann als konstantes Array SEG_DEC[] definiert werden, indem in ihm die leuchtenden Segmente angegeben werden. Auf diese Weise lassen sich die verschiedensten Muster definieren. Grenzen werden nur durch die geringe Anzahl von Segmenten gesetzt.

```
const uint8_t SEG_DEC[] =
{
  SEG_B | SEG_C | SEG_D | SEG_E | SEG_G,    // d
  SEG_A | SEG_D | SEG_E | SEG_F | SEG_G,    // E
  SEG_A | SEG_D | SEG_E | SEG_F,            // C
  SEG_D                                     // _
};
```

Die Helligkeit des Displays wird mit der Methode setBrightness() eingestellt. Die Werte für die Helligkeit dürfen zwischen 0 (dunkel) und 7 (maximale Helligkeit) variieren.

```
display.setBrightness(2);
```

Die Methode setSegments() schaltet die durch den Übergabeparameter definierten Segmente des Displays ein. Im Array SEG_DEC[] hatte ich die Zeichen »dEC_« definiert, die so ausgegeben werden:

```
display.setSegments(SEG_DEC);
```

Mit den Methoden showNumberDec() und showNumberHexEx() werden dezimale bzw. hexadezimale Zahlen angezeigt. Der logische Parameter entscheidet über die Anzeige von führenden Nullen.

```
display.showNumberDec(i, false);
display.showNumberHexEx(i, 0, true);
```

Abbildung 7.7 und Abbildung 7.8 zeigen die Anzeige von negativen und positiven Zahlen mit den ersten Anweisungen.

Abbildung 7.7 Anzeige einer negativen Zahl

Abbildung 7.8 Anzeige einer positiven Zahl

Die verwendete Library unterstützt auch die Ansteuerung der Dezimalpunkte, was aber hardwareseitig nicht bei jeder Anzeige vorgesehen ist.

Neben den Sieben-Segment-Anzeigen mit dem Treiber TM1736 gibt es auch nahezu baugleiche, die einen MAX7219/MAX7221 als Treiber einsetzen.

7.3 LED-Dot-Matrix-Anzeige

Im vorigen Abschnitt konnten Sie sehen, dass bei Sieben-Segment-Displays die Anzeigemöglichkeiten weitgehend auf numerische Anzeigen beschränkt bleiben.

Sogenannte LED-Dot-Matrix-Anzeigen (Punkt-Matrix) bieten da wesentlich mehr Möglichkeiten. Mehrstellige Anzeigen werden in der Regel aus mehreren Einzelelementen zusammengesetzt. Abbildung 7.9 zeigt links eine 8×8-LED-Matrix-Anzeige und rechts ein Breakout Board oder Panel mit einer 32 × 8 LEDs umfassenden Matrix-Anzeige.

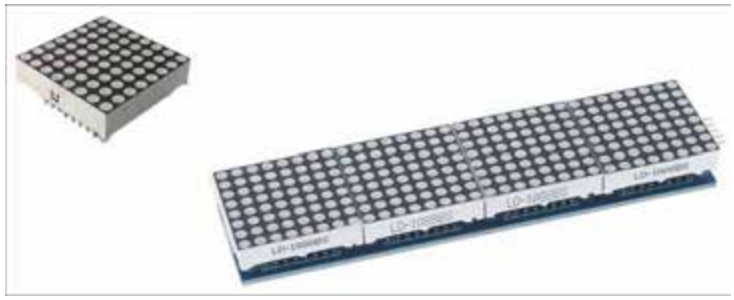

Abbildung 7.9 LED-Matrix-Anzeigen

Wie aus Abbildung 7.9 ersichtlich ist, besteht das LED-Panel aus vier intern verbundenen 8×8-LED-Matrix-Elementen. Jedes Element wird durch einen eigenen, unter dem Display angeordneten Treiber-IC MAX7219 oder MAX7221 angesteuert. Die MAX72xx sind kompakte Display-Treiber für die serielle Ein-/Ausgabe, die kaskadiert werden können. Sie können den Ausgang eines Displays mit dem Eingang eines weiteren Displays verbinden und erhalten so beispielsweise eine 32×8-Dot-Matrix-Anzeige. Mithilfe eines MAX72xx-Display-Treibers können numerische Sieben-Segment-LED-Anzeigen mit bis zu acht Ziffern, Balkendiagramm-Anzeigen oder 64 einzelne LEDs (also auch die 8×8-Matrix) angesteuert werden.

Durch die Kaskadierung wird die Ansteuerung mithilfe eines Mikrocontrollers sehr einfach, da es aus Sicht der Ansteuerung unerheblich ist, wie viele Einzelelemente kaskadiert werden. Abbildung 7.10 zeigt den Anschluss eines 8×8-Dot-Matrix-Displays an den Arduino Uno.

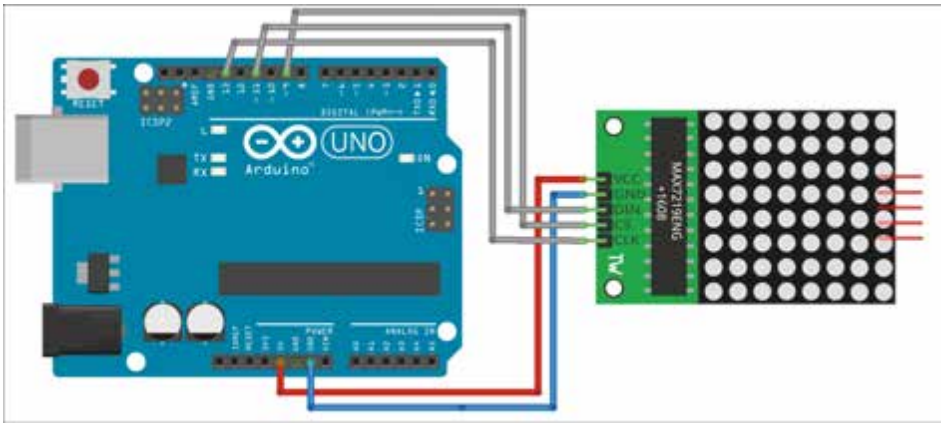

Abbildung 7.10 Dot-Matrix-Anzeige am Arduino Uno

Da es sich hier um ein SPI-Interface zum Display handelt, müssen Sie auch die SPI-Anschlüsse des Arduino verwenden. Beim Arduino Uno liegen die Hardware-SPI-Anschlüsse auf den in Tabelle 7.4 genannten I/O-Pins. Verwendet werden hier aber nur SCK und MOSI.

Arduino Uno-I/O-Pin	SPI
13	SCK
12	MISO
11	MOSI
10	SS

Tabelle 7.4 I/O-Pins für Hardware-SPI

Für den MAX72xx-Display-Treiber gibt es eine Reihe von Librarys unterschiedlicher Qualität. Über den Library Manager habe ich die *LEDMatrixDriver*-Library von Bartosz Bielawski installiert.

Als Programmbeispiel ist in der Library die Datei *MarqueeText.ino* enthalten, mit der eine Laufschrift erzeugt wird, die den gesamten Zeichenvorrat darstellt. Im Repository finden Sie dieses mit dem Arduino Uno getestete Programm unter dem Namen *Uno_LEDMatrix_marque.ino*. Außerdem ist hierzu auch ein Video abgelegt.

Mit dem Programm *Uno_LEDMatrix.ino* zeige ich Ihnen hier ausschnittsweise die Anzeige von Dezimalzahlen.

Nach der Einbindung der Library *LEDMatrixDriver* und der ausgelagerten Font-Definition müssen Sie die Leitung für den Chip-Select mit `LEDMATRIX_CS_PIN` dem zu verwendenden I/O-Pin zuordnen, bevor das Display-Objekt `lmd` erstellt werden kann. Die SPI-Leitungen SCK und MOSI werden per Default als verbunden angenommen. Im verwendeten Display werden vier 8×8-Segmente verwendet, was durch die Konstante `LEDMATRIX_SEGMENTS = 4` festgehalten wird. Arbeiten Sie mit einer anderen Anzahl von Segmenten, dann müssen Sie diese Konstante anpassen.

```
#include <LEDMatrixDriver.hpp>
#include "Font.h"

const uint8_t LEDMATRIX_CS_PIN = 9;

// Number of 8x8 segments you are connecting
const int LEDMATRIX_SEGMENTS = 4;
const int LEDMATRIX_WIDTH    = LEDMATRIX_SEGMENTS * 8;

// The LEDMatrixDriver class instance
LEDMatrixDriver lmd(LEDMATRIX_SEGMENTS, LEDMATRIX_CS_PIN);
```

In der Funktion `setup()` kann nun das Display initialisiert und die Helligkeit eingestellt werden. Die Helligkeit wird durch Werte zwischen 0 und 10 gesteuert:

```
void setup()
{
  // init the display
  lmd.setEnabled(true);
  lmd.setIntensity(2);   // 0 = low, 10 = high
}
```

In der Funktion `loop()` wird nun der in Charakter-Array `txt` befindliche Text durch die Funktion `drawString()` zwischengespeichert und mit der Methode `lmd.display()` in den Framebuffer geschrieben. Als Framebuffer wird ein Speicherbereich bezeichnet, der den auszugebenden Inhalt zwischenspeichert. Die Zeilen zur Unterdrückung der führenden Nullen habe ich hier ausgeblendet.

```
void loop()
{
…
  len = strlen(txt);
  drawString(txt, len, x, 0);
```

```
  // Toggle display of the new framebuffer
  lmd.display();
  delay(1000);
}
```

Das Programm *Uno_LEDMatrix.ino* zählt nun im Sekundentakt den Display-Inhalt hoch, bis der Wert 9999 erreicht ist (siehe Abbildung 7.11).

Abbildung 7.11 Anzeige des Programms »Uno_LEDMatrix.ino«

Den Font, der angezeigt werden soll, habe ich im Programm in den Tab *font.h* ausgelagert. Hier ist ein Ausschnitt:

```
byte font[95][8] = { {0,0,0,0,0,0,0,0}, // SPACE
…
                     {0x00,0x04,0x0c,0x14,0x04,0x04,0x04,0x04}, // 1
                     {0x00,0x30,0x48,0x04,0x04,0x38,0x40,0x7c}, // 2
                     {0x00,0x38,0x04,0x04,0x18,0x04,0x44,0x38}, // 3
                     {0x00,0x04,0x0c,0x14,0x24,0x7e,0x04,0x04}, // 4
…
};
```

Wollen Sie den Font anpassen oder Zeichen darin verändern, dann können Sie das mit dem *DotMatrixTool* von Stefan Gordon (*https://github.com/stefangordon/dotmatrixtool*) auf sehr komfortable Weise tun. Abbildung 7.12 zeigt die Definition des Zeichens »1« und den erzeugten Code als Beispiel.

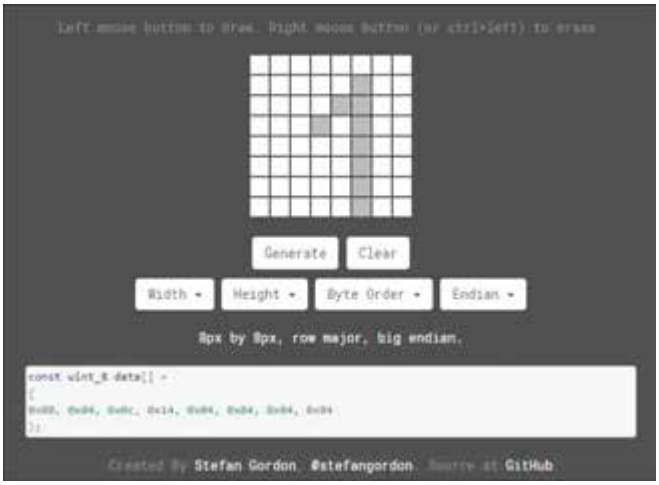

Abbildung 7.12 DotMatrixTool

7.4 Seriell gesteuerte RGB-LEDs

Wie die letzten Abschnitte gezeigt haben, ist die Ansteuerung einer größeren Anzahl von farbigen LEDs schnell mit erheblichem Schaltungsaufwand verbunden. Deshalb ist es fast naheliegend, RGB-LEDs und Steuerschaltung in einem Chip zu vereinen. Mittlerweile sind dazu mehrere Varianten auf dem Markt, von denen ich Ihnen hier NeoPixel und DotStar vorstellen möchte.

7.4.1 NeoPixel

Von der chinesischen Firma *WorldSemi* (*http://www.world-semi.com/*) wird mit dem *WS2812* eine Kombination aus einer Steuerschaltung WS2811 und einem RGB-LED-Chip angeboten, die eine seriell gesteuerte RGB-LED bildet. Diese Bauelemente gibt es in verschiedenen Gehäuseformen.

Abbildung 7.13 zeigt links eine WS2812-RGB-LED in einem herkömmlichen 8-mm-LED-Gehäuse, während rechts ein 5050-Gehäuse ($5 \times 5\ mm^2$) dargestellt ist. Adafruit bezeichnet diese Bauelemente als *NeoPixel*, und dieser Begriff hat sich eingebürgert. Das 5050-Gehäuse eignet sich sehr gut zum Aufbau von LED-Ketten oder -Matrizen. Aufgrund des seriellen Interface können WS2812 nahezu endlos kaskadiert werden. Das Datensignal wird von NeoPixel zu NeoPixel wieder aufgefrischt (siehe Abbildung 7.14).

Abbildung 7.13 NeoPixel

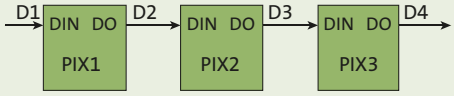

Abbildung 7.14 NeoPixel-Kaskadierung

Achtung: Strombedarf

Den nicht unerheblichen Strombedarf der NeoPixel müssen Sie bei der Kaskadierung unbedingt beachten. Werden alle drei LEDs im WS2812 mit je 20 mA betrieben, dann sind das 60 mA Verbrauch pro NeoPixel.

Setzen Sie beispielsweise einen ein Meter langen *Adafruit NeoPixel Digital RGB LED Strip* mit 144 LEDs ein, dann kommen immerhin reichliche 8,6 A zusammen. Sie sollten dann ein 5 V/10 A-Netzteil vorsehen.

Nach dem Power-On-Reset empfängt der Eingang DIN Daten vom steuernden Controller. Das erste NeoPixel verwendet die ersten 24 Bit und speichert diese im internen Daten-Latch. Die weiteren Daten werden über den Ausgang DO zum Eingang DIN des nächsten NeoPixels in der Kaskade weitergegeben. Nach der Datenübernahme eines NeoPixels werden die weiterzugebenden Daten um 24 Bit reduziert.

Die Übertragungszeit für ein Bit beträgt ungefähr 1,25 µs (TH +TL = 1,25 µs ± 600 ns), und die Rücksetzzeit zwischen zwei Aktualisierungen beträgt ungefähr 50 µs. Abbildung 7.15 zeigt das Timing für den WS2812.

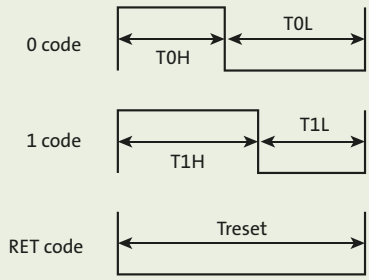

Abbildung 7.15 Timing des WS2812

Unter diesen Bedingungen ist die Refresh-Rate wie folgt definiert:

$$f_R = \frac{1}{(24 \times T_{\text{bit}} \times n_{\text{PIXEL}}) + T_{\text{reset}}}$$

Das ergibt einen theoretischen Wert von 1887 Hz.

Aufgrund der seriellen Verbindung von NeoPixels ist ihr Anschluss an einen Arduino unabhängig von deren Anzahl. Es spielt also für die Schaltung keine Rolle, ob wir eine *WS2812 RGB-LED 8 mm* oder einen NeoPixel-Ring oder -Strip mit einer Vielzahl von WS2812-5050-RGB-LEDs anschließen.

Abbildung 7.16 zeigt beispielhaft einen WS2812-LED-Strip am Arduino Uno. Der LED-Strip wird separat mit Spannung versorgt, weil maximal 8 × 60 mA Stromaufnahme erwartet werden müssen. Zur Pufferung von Schaltspitzen wird ein 1000-µF-Elektrolytkondensator eingesetzt.

Bei Verwendung der *Adafruit-NeoPixel*-Library müssen Sie sich nicht um das Timing des Übertragungsprotokolls kümmern. Die Library kann über den Library Manager installiert werden.

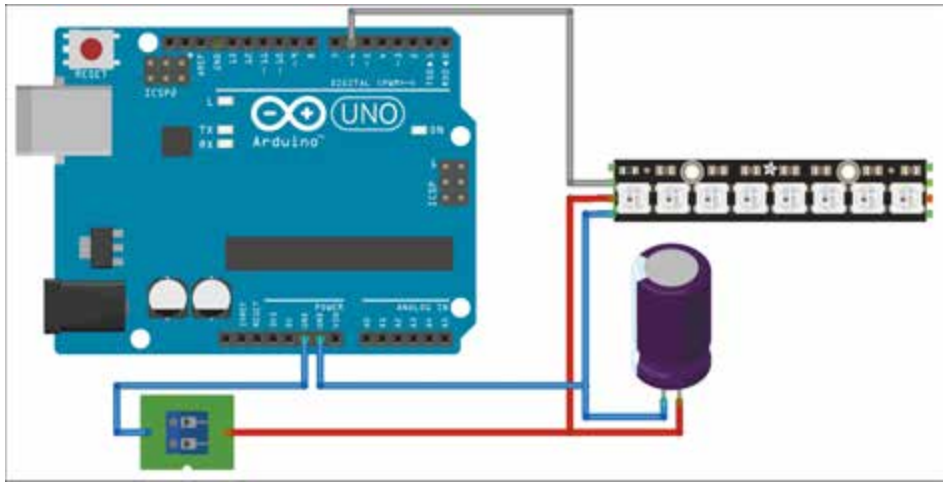

Abbildung 7.16 WS2812-LED-Strip am Arduino Uno

Bestandteil der Library sind einige Programmbeispiele, die zu erzeugende Effekte sehr gut dokumentieren.

Im Programm *Uno_NeoPixels.ino* zeige ich Ihnen nur die einfachsten Möglichkeiten mit einem (Verlaufs-)Blinker, den Sie bestimmt schon an einem Fahrzeug vor Ihnen gesehen haben.

Durch die Library *Adafruit NeoPixel* sind alle komplexen Operationen gekapselt. Die Ansteuerung des hier verwendeten NeoPixel-Streifens mit acht NeoPixels ist dadurch recht einfach.

Nach der Einbindung der erforderlichen Librarys

```
#include <Adafruit_NeoPixel.h>
#ifdef __AVR__
  #include <avr/power.h>
#endif
```

ist der Anschluss an dem Arduino zu definieren, der die Daten an das erste NeoPixel sendet. Gemäß Abbildung 7.16 ist das der I/O-Pin 6. Die Anzahl der im Streifen (Strip) oder Ring kaskadierten NeoPixel muss wegen des aufzubauenden Telegramms mitgeteilt werden (`NUMPIXELS`). Die Zeit zwischen dem Beschreiben der einzelnen NeoPixels (`DELAYVAL`) und die Helligkeit (`BRIGHTNESS`) werden ebenfalls zunächst definiert:

```
#define PIN        6   // Pin driving DIN of first NeoPixel
#define NUMPIXELS 8   // Number of NeoPixels in strip or ring
#define DELAYVAL 100  // Time (in milliseconds) to pause between pixels
#define BRIGHTNESS 50 // Brightness of NeoPixel
```

Damit sind alle Vorkehrungen getroffen, um eine Instanz der Klasse `Adafruit_NeoPixel` zu erzeugen. Die ersten beiden zu übergebenden Parameter habe ich eben erläutert. Beim dritten Parameter handelt es sich um eine Angabe zum NeoPixel-Typ, der hier mit `NEO_GRB + NEO_KHZ800` gesetzt ist. Dieser Wert entspricht auch dem Default-Wert und könnte sogar weggelassen werden.

```
// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUMPIXELS, PIN, NEO_GRB +
  NEO_KHZ800);
```

In der Funktion `setup()` wird nun mit `strip.begin()` der Ausgangspin konfiguriert und mit `strip.setBrightness()` die Helligkeit der NeoPixel eingestellt, bevor mit `strip.show()` die Daten (Bitstream) an die NeoPixels geschickt werden:

```
void setup()
{
  strip.begin();
  strip.setBrightness(BRIGHTNESS);
  strip.show(); // Initialize all strip to 'off'
}
```

In der Funktion `loop()` werden als Erstes alle NeoPixel gelöscht (`strip.clear()`, `strip.show()`), um dann anschließend die NeoPixel nacheinander mit einem Wert für Rot zu beschreiben (`strip.setPixelColor(i, strip.Color(40, 0, 0))`). Der Wert `40` für Rot bedeutet eine reduzierte Helligkeit, die mit dem Wert für `BRIGHTNESS` skaliert wird. Das bedeutet, dass in der Methode `strip.Color(R, G, B)` die Helligkeit der einzelnen Farben beeinflusst werden kann, während `strip.setBrightness()` die Gesamthelligkeit beeinflusst.

```
void loop()
{
  strip.clear();   // Set all pixel colors to 'off'
  strip.show();    // Send the updated pixel colors to the hardware.
  delay(DELAYVAL); // Pause before next pass through loop

  for(int i=0; i<NUMPIXELS; i++)
  {
    strip.setPixelColor(i, strip.Color(40, 0, 0)); //red
```

```
    strip.show();    // Send the updated pixel colors to the hardware.
    delay(DELAYVAL); // Pause before next pass through loop
  }
}
```

Wenn Sie sich den Zusammenhang zwischen der Wellenlänge des Lichts und der abgestrahlten Farbe eines NeoPixels verdeutlichen wollen, dann sollten Sie sich das Programm *Uno_Neopixels_Spektrum.ino* im Repository ansehen. Die Basis für dieses Programm wurde von Michael Stal erstellt (*https://www.heise.de/developer/artikel/Von-Erleuchtungen-und-Lichterketten-3277261.html*).

Im ersten Schritt wird auf einem NeoPixel-RGB-Strip das Spektrum des sichtbaren Lichts ausgegeben. Vergleichen Sie hierzu die Abbildung 7.17 mit Abbildung 7.18.

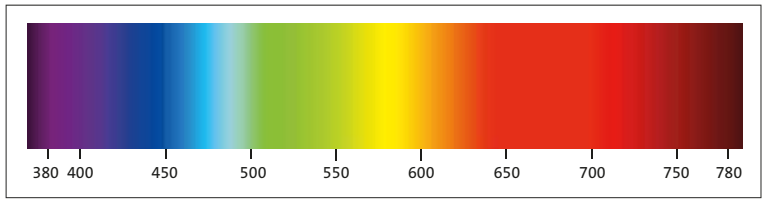

Abbildung 7.17 Spektrum des sichtbaren Lichts

Abbildung 7.18 Spektrum des abgestrahlten Lichts der NeoPixel

Sie können außerdem über den Serial Monitor die gewünschte Wellenlänge eingeben und erhalten dann die dazu passenden RGB-Werte (siehe Abbildung 7.19). Zum Vergleich werden diese Werte zusätzlich an die NeoPixel ausgegeben, die dann in der betreffenden Farbe leuchten.

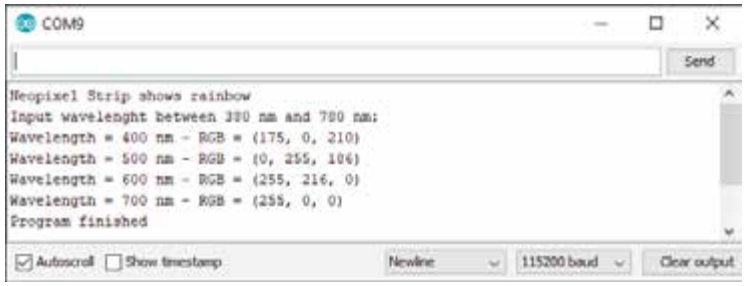

Abbildung 7.19 Abruf der RGB-Werte für verschiedene Wellenlängen

7.4.2 DotStar

Die WS2818-NeoPixel haben sich praktisch als Standard etabliert und es gibt einige Clones, wie z. B. PD9823/PL9823.

Nachteilig an den NeoPixels ist, dass vom steuernden Mikrocontroller ein striktes Timing gefordert wird. Für unser Beispiel im letzten Abschnitt war das kein Problem. Bei anderen Controllern muss das nicht so sein.

Das taiwanesische Unternehmen *APA Electronic Co.* hat einen neuen Controller-IC zur Steuerung der RGB-LED entwickelt und die Kombination von Controller und LED in einem 5050-Gehäuse als *APA102* auf den Markt gebracht. Adafruit bietet eine Vielzahl unterschiedlicher Formate von LED-Streifen auf APA102-Basis an und bezeichnet diese als *DotStar*. Abbildung 7.20 zeigt eine APA102-LED mit sechs Anschlüssen.

Abbildung 7.20 APA102

Das Interface zum steuernden Mikrocontroller benutzt eine Leitung mehr und ist nun klassisches SPI. Zum SPI-Interface werde ich Ihnen in Abschnitt 10.2 noch Grundlagen vermitteln.

Jede LED verfügt über zwei Eingänge und zwei Ausgänge, die in Reihe geschaltet werden können (siehe Abbildung 7.21). Darüber hinaus können die Daten mit einer nahezu beliebigen Taktrate übertragen werden.

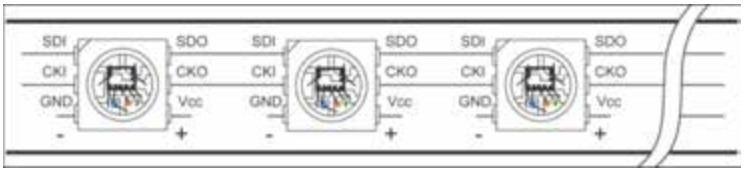

Abbildung 7.21 Kaskadierung mehrerer APA102

Das Datenformat der seriellen Daten ist in Abbildung 7.22 dargestellt. In *Tim's Blog* (*https://cpldcpu.wordpress.com/2014/11/30/understanding-the-apa102-superled/*) finden Sie weiterführende Erläuterungen.

Über die Bits *Global* im *LED Frame* kann die Helligkeit der LED insgesamt in 32 Schritten (0 bis 31) gesteuert werden. Bei maximaler Helligkeit (31) ist kein PWM-Flackern sichtbar. Das unterscheidet sich erheblich vom WS2812.

Zur Helligkeitssteuerung sind zwei PWM-Modulationsschemata überlagert worden: Die globale Helligkeit wird mit ~ 582 Hz moduliert, während der RGB-Wert mit einer ungefähr 32-mal höheren Frequenz (19,2 kHz) moduliert wird. Diese Frequenz liegt deutlich über der Wahrnehmungsschwelle (*Flimmerfusionsfrequenz*). Die sehr hohe PWM-Frequenz ist ein markantes Merkmal dieser LEDs. In Kombination mit der höheren Aktualisierungsrate aufgrund der SPI-Schnittstelle sollten sie für Anwendungen mit schnell wechselnden (Bild-)Inhalten besser geeignet sein als die WS2812.

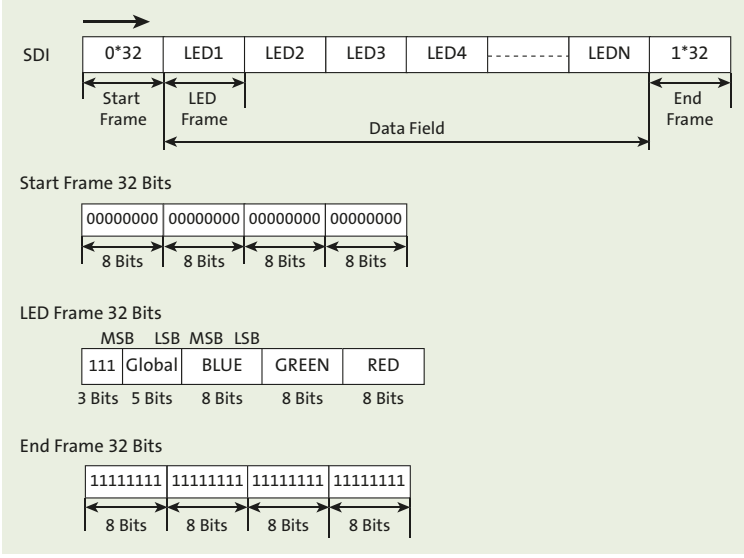

Abbildung 7.22 Das Datenformat der seriellen Daten für APA102

Zusammenfassend lässt sich sagen, dass die Eigenschaften von APA102-LEDs im Vergleich zum De-facto-Standard WS2812 vielversprechend sind (*https://cpldcpu.wordpress.com/2014/08/27/apa102/*):

- Ansteuerung über Standard-SPI-Schnittstelle: Dadurch ist kein kritisches Timing erforderlich und der Datenaustausch erfolgt viel schneller.
- Die sehr hohe PWM-Frequenz ermöglicht flimmerfreie Darstellungen.
- Ein spezielles Gehäuse mit besserer Wärmeableitung ist verfügbar.

Die Ansteuerung durch die Hardware-SPI-Schnittstelle erfordert festgelegte I/O-Pins am Arduino. In Abbildung 7.10 hatte ich diese bereits gezeigt. In der Gesamtschaltung nach Abbildung 7.23 wird der Eingang CI am DotStar-Strip vom SPI-Clock SCK (Pin 13) getrieben. Der Dateneingang DI ist mit dem SPI-Ausgang MOSI (Pin 11) verbunden.

Der komplexe Aufbau des seriellen Datenstroms muss Sie nicht weiter beunruhigen, denn es gibt auch hierfür wieder eine Library, die die Ansteuerung vergleichbar zu den NeoPixels gestaltet.

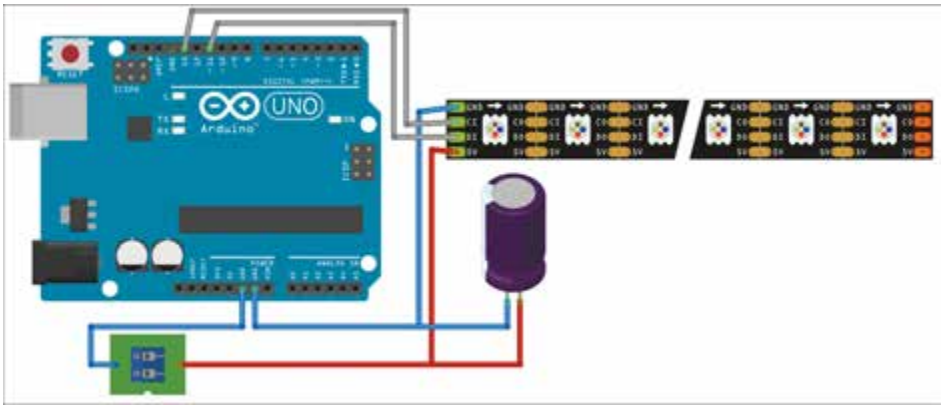

Abbildung 7.23 DotStar-Strip am Arduino Uno

Die *DotStar*-Library unterstützt die Ansteuerung von APA102-LEDs und ich verwende diese im folgenden Programmbeispiel.

Das Programm *Uno_DotStar.ino* arbeitet mit drei APA102-LEDs, die auf Breakout Boards installiert und kaskadiert sind.

Zu Beginn werden wieder die erforderlichen Librarys eingebunden. Die anzuzeigenden Daten werden über SPI an die APA102-LEDs übertragen, weshalb die *SPI*-Library erforderlich ist. Die *Adafruit-DotStar*-Library stellt die Programmierschnittstelle zur Verfügung:

```
#include <Adafruit_DotStar.h>
#include <SPI.h>
```

Die Anzahl der anzusteuernden LEDs wird in `NUMPIXELS` festgehalten, und `DELAYVAL` definiert eine Wartezeit von 100 ms:

```
#define NUMPIXELS 3   // Number of LEDs in strip
#define DELAYVAL 100  // Time (in milliseconds) to pause between pixels
```

Bei der Verwendung der Hardware-SPI sind die zu verwendenden I/O-Pins festgelegt und müssen deshalb bei der Initialisierung des Objekts `strip` nicht mehr angegeben werden:

```
// Hardware SPI is a little faster, but must be wired to specific pins
// (Arduino Uno = pin 11 for data, 13 for clock, other boards are different).
Adafruit_DotStar strip(NUMPIXELS, DOTSTAR_BGR);
```

In der Funktion `setup()` werden die I/O-Pins initialisiert und alle LEDs ausgeschaltet:

```
strip.begin(); // Initialize pins for output
strip.show();  // Turn all LEDs off ASAP
```

Nun können in der Funktion `loop()` nacheinander die drei LEDs eingeschaltet werden. Die LEDs leuchten rot, allerdings mit reduzierter Helligkeit (`40, 0, 0`):

```
strip.clear();   // Set all pixel colors to 'off'
strip.show();    // Send the updated pixel colors to the hardware.
delay(DELAYVAL); // Pause before next pass through loop

for(int i=0; i<NUMPIXELS; i++)
{
  strip.setPixelColor(i, 40, 0, 0); //red
  strip.show();    // Send the updated pixel colors to the hardware.
  delay(DELAYVAL); // Pause before next pass through loop
}
```

Durch den Einsatz der *Adafruit-DotStar*-Library zeigt das Programm *Uno_DotStar.ino* nur geringe Abweichungen vom Programm *Uno_NeoPixels.ino*, was durchaus den Sinn einer Library ausmacht.

7.4.3 NeoPixel vs. DotStar

Zu besseren Orientierung möchte ich in Tabelle 7.5 NeoPixel und DotStar vergleichend gegenüberstellen (*https://learn.adafruit.com/adafruit-dotstar-leds*).

DotStar	NeoPixel
Vorteile: sehr schnelle Daten- und PWM-Raten einfaches SPI-Interface ohne strikte Anforderungen an das Timing keine speziellen Anforderungen an I/O-Pins, DMA, Interrupt-Management	Vorteile: kostengünstiger als APA102 zahlreiche Formfaktoren (Pixel, Ring, Matrix etc.) Eine Steuerleitung RGRW-(RGB+white-)Varianten verfügbar DMA-Unterstützung für verschiedene Plattformen, z. B. SAMD21, SAMD51, ESP8266/ESP32 FadeCandy-kompatibel. *FadeCandy* ist ein über USB-gesteuerter NeoPixel-Treiber für effektvolle Beleuchtungssteuerung. Ein schönes Beispiel finden Sie unter *https://www.youtube.com/watch?v=Gvz3PiechTE*

Tabelle 7.5 Die Vor- und Nachteile von DotStar und NeoPixel

DotStar	NeoPixel
Nachteile: ▶ teurer als NeoPixel ▶ wenige Formfaktoren verfügbar ▶ zwei Steuerleitungen erforderlich	Nachteile: ▶ 800 kHz Datenrate – nicht jede Plattform ist dazu in der Lage. ▶ 400 Hz Refresh-/PWM-Rate ist nur bedingt für flackerfreie Anzeigen geeignet. ▶ nicht kompatibel mit einigen Plattformen, die keine DMA haben und/oder Interrupts benötigen (z. B. die *Arduino Servo*-Library oder die `tone()`-Funktion für ATmega) ▶ benötigt spezielle I/O-Pins bei verschiedenen Plattformen, z. B. *ESP8266 DMA Support* u. a.

Tabelle 7.5 Die Vor- und Nachteile von DotStar und NeoPixel (Forts.)

7.4.4 FastLED

DotStar und NeoPixel sind beide seriell gesteuerte und kaskadierbare RGB-LEDs. Die *FastLED*-Library von Daniel Garcia unterstützt beide Interfaces.

FastLED ermöglicht die Steuerung einer Vielzahl von LED-Chips – wie der von Adafruit (NeoPixel, DotStar, LPD8806), SparkFun (WS2801) und AliExpress. Außer Funktionen zum Schreiben von Farbwerten zur LED enthält die Library auch Funktionen für 8-Bit-Berechnungen zur Manipulation von RGB-Werten sowie Klassen für den Pin- und SPI-Zugriff.

Mit der Entwicklung dieser Library wurden die folgenden Ziele verfolgt:

- Schneller Start für neue Entwickler: Schließen Sie die betreffende LED oder den betreffenden LED-Strip etc. an, und konfigurieren Sie diese, ohne sich mit den Chips selbst oder den zu verwendenden Datenformaten auseinandersetzen zu müssen.
- Wechseln Sie die verwendeten LEDs, und passen Sie die Konfiguration im Quelltext an. Nach der Kompilation läuft der gleiche Code mit den neuen LEDs wieder.
- Sie erreichen eine hohe Performance durch einen effizienten Code.

Setzen Sie die Library ein, und bilden Sie sich anhand Ihrer eigenen Anwendung eine eigene Meinung. Die aktuelle Version der FastLED-Library installieren Sie über den Library Manager.

Das Programm *Uno_FastLED.ino* zeigt an einem einfachen Beispiel die Ansteuerung von drei NeoPixel- bzw. drei DotStar-LEDs. Zu Beginn erfolgt die Einbindung der FastLED-Library, gefolgt von einer Reihe von Vereinbarungen. Die I/O-Pins für `CLOCK` und

DATA werden einheitlich für beide LED-Varianten auf die SPI-Pins gelegt. Für die NeoPixel sind die Daten von I/O-Pin 11 kein Problem, und für die DotStar-LED sind bei Verwendung der Hardware-SPI der I/O-Pin 13 für CLOCK und der I/O-Pin 11 für DATA zwingend.

```
#include <FastLED.h>

#define NUM_LEDS 3
#define DATA_PIN 11
#define CLOCK_PIN 13
#define DELAYVAL 250  // Time (in milliseconds) to pause between pixels
#define BRIGHTNESS 40 // Brightness
```

Mit der folgenden Anweisung wird nun ein Array für die verwendete Anzahl von LEDs erzeugt:

```
CRGB leds[NUM_LEDS];  // Define the array of leds
```

In der Funktion setup() werden nun die Objekte für die NeoPixel- bzw. die DotStar-LEDs erzeugt. Ich zeige hier nur die beiden verwendeten Varianten im Quelltext. Die Library bietet Unterstützung für eine Vielzahl von Clones und Varianten und hat damit recht universellen Charakter.

```
// Uncomment/edit one of the following lines for your leds arrangement.
// ## Clockless types ##
FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUM_LEDS);
// ## Clocked (SPI) types ##
// FastLED.addLeds<APA102, DATA_PIN, CLOCK_PIN, BGR>(leds, NUM_LEDS);
```

Nachdem das gewünschte FastLED-Objekt erstellt ist, können Sie die Helligkeit vorgeben und alle LEDs ausschalten:

```
FastLED.setBrightness(BRIGHTNESS);
FastLED.show(); // Initialize all strip to 'off'
```

Nun können in der Funktion loop() die wiederkehrenden Aktionen festgelegt werden. Hier ist das zuerst das Löschen aller LEDs, bevor die LEDs nacheinander mit wechselnder Farbe angesteuert werden:

```
FastLED.clear();   // Set all pixel colors to 'off'
FastLED.show();    // Send the updated pixel colors to the hardware.
FastLED.delay(DELAYVAL); // Pause before next pass through loop

for(int i=0; i<NUM_LEDS; i++)
{
```

```
  switch (i)
  {
    case 0: leds[i] = CRGB::Red;   break;
    case 1: leds[i] = CRGB::Green; break;
    case 2: leds[i] = CRGB::Blue;  break;
  }
  FastLED.show();
  FastLED.delay(DELAYVAL); // Pause before next pass through loop
}
```

Wollen Sie nun von den NeoPixel- auf DotStar-LEDs wechseln, dann reicht es, die NeoPixel-Zeile im `setup()` auszukommentieren und die DotStar-Zeile zu aktivieren. Nach erfolgter Kompilierung und dem Upload läuft alles mit der anderen LED-Variante genauso.

7.5 LCDs

LCDs bestehen aus Segmenten, die unabhängig voneinander ihre Transparenz ändern können. Dazu wird mit elektrischer Spannung in jedem Segment die Ausrichtung der Flüssigkristalle gesteuert. Damit ändert sich die Durchlässigkeit für polarisiertes Licht.

Polarisiertes Licht wird mit Polarisationsfiltern erzeugt, die entweder einfallendes Umgebungslicht (bei reflektierenden Anzeigen) oder Licht einer Hintergrundbeleuchtung (bei Anzeigen im Transmissionsmodus) filtern.

Soll ein Display beliebige Inhalte darstellen können, sind die Segmente in einem gleichmäßigen Raster angeordnet (Pixel-Array). Bei Geräten, die nur bestimmte Zeichen darstellen sollen, haben die Segmente oft eine speziell darauf abgestimmte Form, so bei der Sieben-Segment-Anzeige zur Darstellung von Zahlen (Matrix-Anzeige). Abbildung 7.24 zeigt ein kundenspezifisches LCD, das beide Möglichkeiten aufzeigt.

Abbildung 7.24 Beispiel für ein kundenspezifisches LCD

7.5.1 Display mit HD44780

Das Angebot an preiswerten LCDs ist nahezu unüberschaubar. Dennoch zeigt sich, dass gerade bei alphanumerischen LCDs in den meisten Fällen LCD-Controller vom Typ *HD44780* von *Hitachi*, die heute als Industriestandard gelten, eingesetzt werden oder Controller, die zu diesem kompatibel sind.

Grundsätzlich sind zwei Formen der Ansteuerung eines LCDs zu unterscheiden. Bei der *direkten Ansteuerung* treiben die I/O-Pins des angeschlossenen Mikrocontrollers direkt die Leitungen des LCDs. Bei der indirekten Ansteuerung wird ein I^2C- oder SPI-Bus-Schaltkreis zwischen Mikrocontroller und LCD geschaltet, um den Bedarf an digitalen I/O-Pins zu reduzieren.

Abbildung 7.25 zeigt den direkten Anschluss eines Text-LCDs mit zwei Zeilen zu je 16 Zeichen. Die Schnittstelle zum ansteuernden Arduino ist parallel, d. h., die Datenleitungen des LCDs sind direkt mit den digitalen Ausgängen des Arduino verbunden.

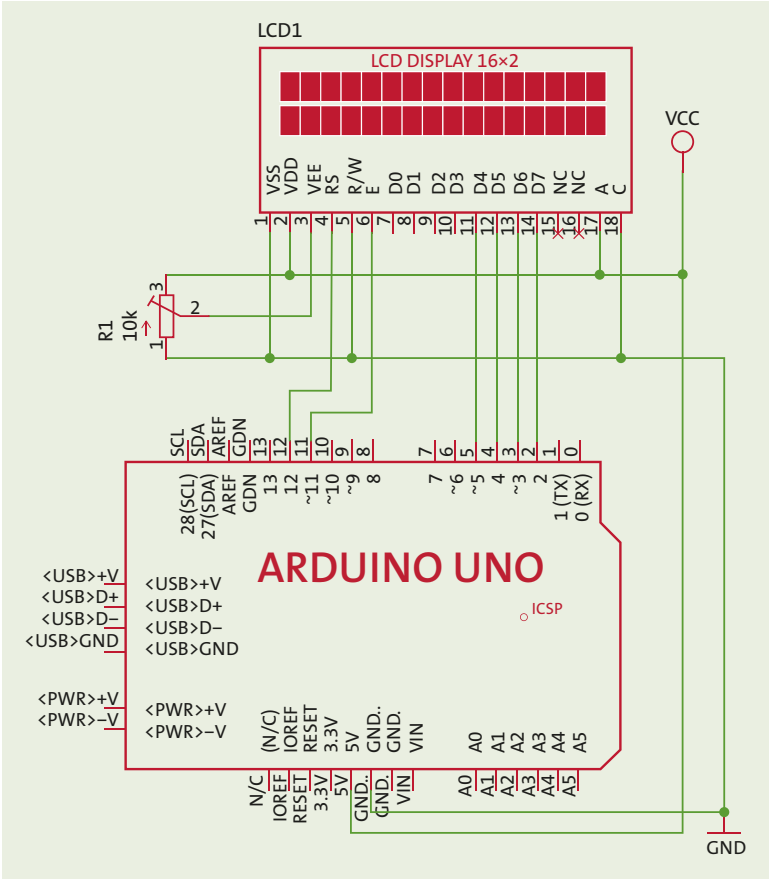

Abbildung 7.25 HD44780-LCD am Arduino Uno

Die Ansteuerung kann mit allen acht Datenleitungen oder aber mit nur vier Datenleitungen erfolgen. Damit nicht unnötig I/O-Ressourcen durch die Ansteuerung des LCDs gebunden sind, wird das LCD gewöhnlich im 4-Bit-Mode betrieben.

Das Potenziometer dient zum Einstellen des Kontrasts des LCDs, die in der Regel einmal vorgenommen wird. Über die Anschlüsse A und C erfolgt die Spannungsversorgung der Hintergrundbeleuchtung.

Text-LCDs sind in unterschiedlicher Ausstattung (einzeilig, zweizeilig, vierzeilig) mit acht, 16 oder 20 Zeichen nahezu standardisiert von zahlreichen Anbietern für einen Preis von unter 10 € erhältlich.

Die Programmierung der Ausgaben auf dem Text-LCD wird durch zahlreiche Librarys unterstützt. Ich verwende hier die *LiquidCrystal*-Library v1.0.7.

In den Beispielen zur LiquidCrystal-Library ist das Programmbeispiel *Blink.ino* (File • Examples • LiquidCrystal • Blink) enthalten, das die notwendige Initialisierung zeigt.

Die LiquidCrystal-Library wird durch die Anweisung

```
#include <LiquidCrystal.h>
```

eingebunden. Die Zuordnung der Arduino-I/O-Pins zu den Anschlüssen des LCDs erfolgt durch die Konstantendefinition und nachfolgende Initialisierung gemäß der Anschlussbelegung in Abbildung 7.25.

```
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
```

Bevor Text über das LCD angezeigt werden kann, müssen Sie das LCD noch initialisieren und ihm die Zahl der Zeichen pro Zeile sowie die Anzahl der Zeilen mitteilen. Dann können Sie Texte über das LCD anzeigen lassen:

```
lcd.begin(16, 2);
lcd.print("Hello World!");
```

Das ist bereits alles, denn die Library »versteckt« vor Ihnen, wie komplex die Aufbereitung der anzuzeigenden Informationen ist.

7.5.2 Grove-LCDs mit I²C

Verschiedene LCDs sind mit einem standardisierten RS232- oder I²C-Interface ausgerüstet, sodass die Zahl der belegten I/O-Pins am Arduino erheblich reduziert werden kann. In den meisten Fällen wird das serielle Interface durch ein Zusatzmodul gebildet, das entweder huckepack auf dem Standard-LCD angeordnet wird (siehe Abbildung 7.26) oder wie beim *Grove-LCD RGB Backlight* bereits auf dem Board implementiert ist (siehe Abbildung 7.27).

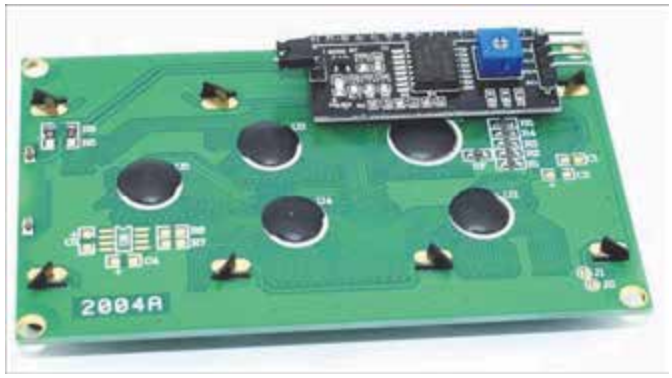

Abbildung 7.26 I2C-Interface am Standard-LCD

Abbildung 7.27 Grove-LCD RGB Backlight

Seeed Studio bietet das in Abbildung 7.27 gezeigte *Grove-LCD RGB Backlight* zum Preis von 11,90 US$ an. Die günstigeren Varianten mit einfarbiger Hintergrundbeleuchtung werden als

- Grove – 16 × 2 LCD (Black on Yellow) – Schwarz auf Gelb
- Grove – 16 × 2 LCD (Black on Red) – Schwarz auf Rot
- Grove – 16 × 2 LCD (White on Blue) – Weiß auf Blau

bereits für 5.95 US$ angeboten (*https://www.seeedstudio.com/category/Grove-c-1003.html?cat=932*).

Die Kontaktierung zum Arduino Uno kann sehr einfach über einen Grove-I²C-Konnektor auf einem *Grove Base Shield v2* nach Abbildung 2.59 erfolgen oder aber auch konventionell über den I²C-Port am Arduino Uno.

Zur Vereinfachung der Software verwende ich die *Grove-LCD-RGB-Backlight*-Library, die alle hier genannten Grove-LCDs unterstützt.

Das Programm *Uno_Grove_LCD.ino* zeigt die einfache Ansteuerung des LCDs.

Sie müssen zwei Librarys einrichten. Die *Wire*-Library wird für die Ansteuerung über den I²C-Bus benötigt, während die Library *rgb_lcd* das eigentliche API (*Application Programming Interface*) zum Display darstellt.

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

Mit der Anweisung `rgb_lcd lcd;` wird eine Instanz der Klasse `rgb_lcd` erzeugt, über deren Methoden nun das Display angesteuert werden kann. Verwenden Sie ein LCD mit farbiger Hintergrundbeleuchtung, dann können die Farben durch die folgenden Konstanten festgelegt werden:

```
const int colorR = 255;
const int colorG = 0;
const int colorB = 0;
```

In der Funktion `setup()` müssen Sie das Display konfigurieren. Wir haben hier ein zweizeiliges Display mit 16 Zeichen pro Zeile. Ist eine farbige Hintergrundbeleuchtung vorhanden, dann kann diese gesetzt werden, bevor ein Text in die obere Zeile geschrieben wird:

```
lcd.begin(16, 2);
lcd.setRGB(colorR, colorG, colorB);
lcd.print("Grove - 16x2 LCD");
```

In der Funktion `loop()` kann nun der Cursor an den Anfang der unteren Zeile positioniert werden. Dann folgen das Zusammensetzen des auszugebenden Strings und dessen Ausgabe auf das LCD:

```
lcd.setCursor(0, 1);
String str = "Running ";
str += (millis()/1000);
str += " s";
lcd.print(str);
```

Wie Sie sehen können, ist die Ausgabe von Daten auf das LCD sehr einfach. Abbildung 7.28 zeigt die Ausgabe des Programms *Uno_Grove_LCD.ino* auf einem *Grove – 16x2 LCD (White on Blue)*.

Abbildung 7.28 Ausgabe des Programms »Uno_Grove_LCD.ino«

7.5.3 LCD Keypad Shield

Das *LCD Keypad Shield* von *DFRobot* vereint die Eingabe über Taster und die Ausgabe über ein 16-×-2-Text-LCD. Über AZ-Delivery.de (*https://www.az-delivery.de/products/azdelivery-hd44780-1602-lcd-module-display-2x16-zeichen-fur-arduino-lcd1602-keypad*) kann das gesamte Board bereits für 7,99 € bezogen werden.

Abbildung 7.29 Das »LCD Keypad Shield« von DFRobot

Der Schaltungsauszug in Abbildung 7.30 zeigt die Schaltungsteile für LCD-Ansteuerung und Tastenabfrage. Der LCD-Teil zeigt bis auf eine leicht unterschiedliche Pinbelegung keine Besonderheiten.

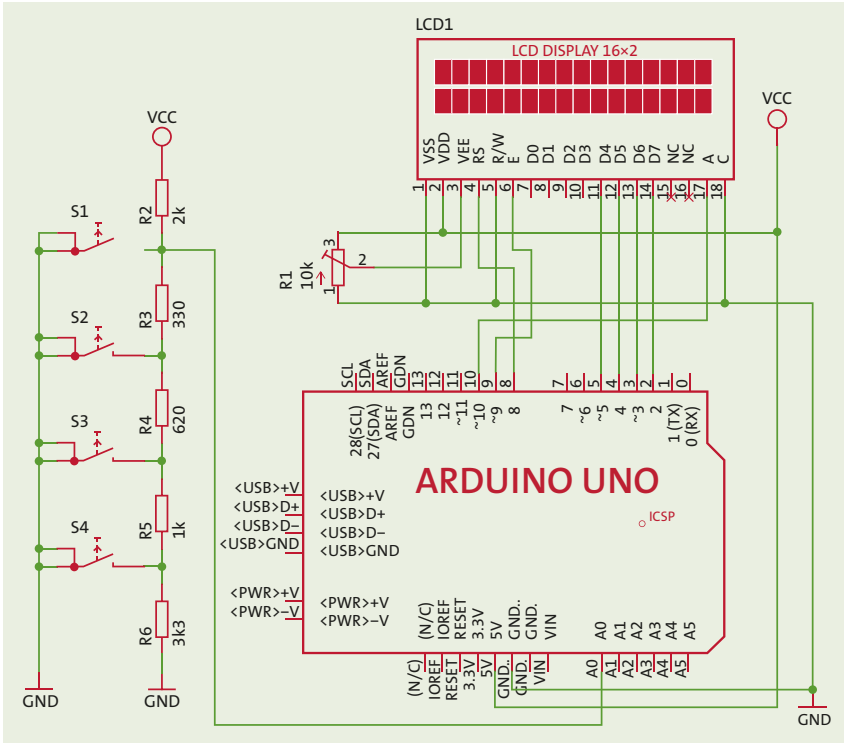

Abbildung 7.30 Das »LCD Keypad Shield« von DFRobot (Schaltungsauszug)

Interessant ist die Lösung der Tastenabfrage, die bei konventioneller Lösung mehrere I/O-Pins nutzen würde. Hier wird aber ein Spannungsteiler benutzt, dessen Spannungsabgriff eine von der betätigten Taste abhängige Spannung aufweist. Diese Spannung wird über Pin A0 dem AD-Umsetzer zugeführt, und dessen Ausgangsdaten repräsentieren dann den gedrückten Schalter.

Durch die *LiquidCrystal*-Library wird die Ansteuerung eines LCDs, das auf dem Hitachi-HD44780-Chipsatz (oder einem kompatiblen Chipsatz) basiert, stark vereinfacht. Die Library unterstützt den 4-Bit- und den 8-Bit-Mode. In Abbildung 7.30 sehen Sie den 4-Bit-Mode. Hinzu kommen ohnehin noch die Steuerleitungen (RS, E und optional R/W sowie A für das Schalten der Hintergrundbeleuchtung).

Den Quelltext des Programmbeispiels *Uno_LCDKeyPadShield.ino* erläutere ich wieder abschnittsweise. Zu Beginn wird die verwendete Library eingebunden und der `DEBUG`-Identifier definiert:

```
#include <LiquidCrystal.h>
#define DEBUG 1
```

Die I/O-Pins werden als Konstanten vereinbart und werden so zur Initialisierung des Objekts `lcd` verwendet:

```
const int rs = 8, en = 9, d4 = 5, d5 = 4, d6 = 3, d7 = 2
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
const int backlight = 10; // Backlight connected to D10
```

Nach der Definition von Variablen und Identifiern für die Taster sind die Vorkehrungen abgeschlossen:

```
int lcd_key     = 0;
int adc_key_in  = 0;

#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5
```

Das Auslesen der Tastatur erfolgt über den Analogeingang A0 und wird im Debug-Mode dann auch im Serial Monitor sichtbar. Je nach ausgelesenem Analogwert wird diesem ein Tastaturstatus (`btnRIGHT`, `btnUP` usw.) zugeordnet und zurückgegeben:

```
int read_LCD_buttons()
{
  adc_key_in = analogRead(0);      // read the value from the sensor
```

```
  if (DEBUG)
  {
    Serial.print("Button value = ");
    Serial.println(adc_key_in);
  }
  // set DEBUG to 1 to read the adc_key_in values via serial monitor
  // my buttons when read are centered at these values: 0, 99, 255, 407, 639
  // I add approx 50 to those values and check to see if we are close
  if (adc_key_in > 1000) return btnNONE; // We make this the 1st option
                                         // for speed reasons since it
                                         // will be the most likely result
  // For V1.1 use this threshold
  if (adc_key_in < 50)   return btnRIGHT;
  if (adc_key_in < 150)  return btnUP;
  if (adc_key_in < 300)  return btnDOWN;
  if (adc_key_in < 450)  return btnLEFT;
  if (adc_key_in < 700)  return btnSELECT;

  return btnNONE;  // when all others fail, return this...
}
```

In der Funktion `setup()` (Listing 7.1) können nun das serielle Interface und das LCD konfiguriert werden. Ebenfalls wird die Eingabeaufforderung PUSH THE BUTTONS ausgegeben:

```
void setup()
{
  Serial.begin(115200);
  pinMode(backlight, OUTPUT);
  digitalWrite(backlight, HIGH);
  lcd.begin(16, 2);              // start the library
  lcd.setCursor(0,0);
  lcd.print("Push the buttons"); // print a simple message
}
```

Listing 7.1 Quelltext Uno_LCDKeyPadShield.ino – setup()

In der Funktion `loop()` (Listing 7.2) wird nun ab Position 9 in der unteren Zeile des LCDs die Programmlaufzeit in Sekunden angezeigt. Am Beginn der unteren Zeile erfolgt bei einer Tastaturbetätigung die Anzeige des Tastaturstatus.

```
void loop()
{
  lcd.setCursor(9,1);        // move cursor to second line 9>th position
  lcd.print(millis()/1000);  // display seconds elapsed since power-up
```

```
  lcd.setCursor(0,1);        // move cursor begin of second line
  lcd_key = read_LCD_buttons();  // read the buttons

  switch (lcd_key) // switch depending on button pressed
  {
    case btnRIGHT: lcd.print("RIGHT "); break;
    case btnLEFT: {…}
    case btnUP: {…}
    case btnDOWN: {…}
    case btnSELECT:
    {
      lcd.print("SELECT");
      digitalWrite(backlight, LOW);
      break;
    }
    case btnNONE:
    {
      lcd.print("NONE  ");
      digitalWrite(backlight, HIGH);
      break;
    }
  }
  delay(250);
}
```

Listing 7.2 Quelltext Uno_LCDKeyPadShield.ino – loop()

Beide Funktionalitäten – Tastenfeld und LCD-Ansteuerung – können als Software-Baustein aufgefasst und in andere Anwendungen eingebaut werden.

7.5.4 Arduino-Shield mit EA DOGS102W-6 und EA PCBARDDOG1701

Von *Electronic Assembly* (EA) werden Text- und Grafik-LCDs angeboten, die eine starke Verbreitung gefunden haben. Speziell für den Arduino bietet EA Shields zur Aufnahme seiner LCDs an (siehe Tabelle 7.6):

Arduino Shield PCB	LCD/ePaper	Typ
PCBARDDOG1701	DOGS102-6	102 × 64 dots, 1,7"
PCBARDDOG7036	DOGM081 DOGM162 DOGM163	1 × 8 Zeichen, 11,97 mm Schrift 2 × 16 Zeichen, 5,57 mm 3 × 16 Zeichen, 3,65 mm

Tabelle 7.6 Arduino-Shields für Displays von Electronic Assembly

Arduino Shield PCB	LCD/ePaper	Typ
PCBARDDOG7565	DOGM132-5 DOGM128-6 DOGL128-6	132 × 32 dots, 2,1" 128 × 64 dots, 2,3" 128 × 64 dots, 2,8"
PCBARDEPA1606	EPA20-A	172 × 72 dots, 2"

Tabelle 7.6 Arduino-Shields für Displays von Electronic Assembly (Forts.)

Ich möchte Ihnen im Folgenden das LCD *DOGS102-6* vorstellen, montiert auf einem *PCBARDDOG1701*.

Bei diesem Shield handelt es sich um ein Grafikdisplay der DOG-Serie mit 102 × 64 Pixel. Als LCD-Controller kommt ein *UC1701* von *UltraChip* aus Taiwan zum Einsatz, der über ein SPI-Interface mit dem steuernden Mikrocontroller verbunden ist. Es ist eine farbige Hintergrundbeleuchtung vorgesehen. Das Arduino-Shield kann als unbestückte Platine PCBARDDOG1701 im Arduino-Uno-Formfaktor bei EA bestellt werden. Alle technischen Informationen zum LCD DOGS102-6 finden Sie im deutschsprachigen Datenblatt des Herstellers (*https://www.lcd-module.de/deu/pdf/grafik/dogs102-6.pdf*). Das komplettierte Arduino-Shield – auch mit unterschiedlich farbiger Hintergrundbeleuchtung – haben Sie dann in der Form zur Verfügung, die in Abbildung 7.31 zu sehen ist.

Zur Ansteuerung des Arduino-Shields müssen Sie sich nicht mit einem weiteren LCD-Controller auseinandersetzen. Die Library wird ebenfalls vom Hersteller bereitgestellt und ist im Zip-Archiv unter *https://www.lcd-module.de/fileadmin/downloads/development%20service/Arduino/Arduino%20meets%20EA%20DOGS102.zip* enthalten. Erzeugen Sie im Library-Verzeichnis ein neues Verzeichnis mit dem Namen *../libraries/DOG_1701*, und extrahieren Sie den Inhalt der Library dorthin. Dann können Sie diese Library auch in Ihrem Quelltext verwenden.

Abbildung 7.31 LCD-Shield mit farbiger Hintergrundbeleuchtung

Das Programm *Uno_DOGS102_Icon.ino* zeigt Ihnen, wie Sie Daten mit dem DOGS102-6 zur Anzeige bringen. Den anzuzeigenden Inhalt konnten Sie schon in Abbildung 7.31 sehen.

Das Programm beginnt durch das Einbinden der SPI-Library wie gehabt, nur dass hier noch verschiedene Fonts mit eingebunden werden, die ebenfalls Bestandteil der Library sind. Zum Schluss wird die Headerdatei *logobmp.h* eingebunden, die die Grafik links oben im Bild darstellt. Darauf komme ich aber noch separat zu sprechen, wenn der Code aus Listing 7.3 vollständig abgearbeitet wurde.

```
#include <SPI.h>
#include <dog_1701.h>
#include <font_16x32nums.h>
#include <font_6x8.h>
#include <font_8x16.h>
#include <font_8x8.h>
#include "logo_BMP.h"
```

Mit der Anweisung `dog_1701 DOG;` wird die Instanz `DOG` der Klasse `dog_1701` erzeugt, die die zur Anzeige der Daten erforderlichen Methoden bereitstellt. Zuvor werden aber noch die I/O-Pins für die Hintergrundbeleuchtung als Konstanten festgelegt:

```
//the following port definitions are used by the demo board "EA PCBARDDOG1701"
const int led_green = 5;
const int led_red   = 3;
int led = led_red;
```

In der Funktion `setup()` können nun die Hintergrundbeleuchtung und das Objekt `DOG` initialisiert werden. Die hier verwendeten Angaben entsprechen der verwendeten Hardware und sind durch diese festgelegt.

```
void setup()
{
  init_backlight(false); //use RGB backlight
  //SS = 10, 0,0=use Hard-SPI, 9=A0, 4=RESET, EA DOGS102-6 (=102x64 dots)
  DOG.initialize(10,0,0,9,4,DOGS102);
}
```

In der Funktion `sample_screen()` wird der in Abbildung 7.31 gezeigte Bildinhalt erzeugt. Das können Sie sehr gut zeilenweise verifizieren:

```
void sample_screen(void)
{
  DOG.clear();  //clear whole display
  DOG.picture(0,0,Image_logo_BMP);
```

```
  DOG.string(30,0,font_8x16,"DOGS102-6");  //font size 8x16 first page
  DOG.rectangle(30,2,102,2,0x03); //show small line (filled pattern =
                                  //0x03) to the left and right
  DOG.string(0,4,font_6x8,"Further info:");
  DOG.string(0,5,font_6x8,"Arduino meets ");
  DOG.string(0,6,font_6x8,"EA DOGS102");
  DOG.string(0,7,font_6x8,"@ lcd-module.com");
}
```

In der Funktion `loop()` wird nun im Takt von 2 Sekunden eine zufällige Farbe für die Hintergrundbeleuchtung erzeugt und der erzeugte Bildinhalt ausgegeben:

```
void loop()
{
  rgb_backlight(random(256), random(256)); //BL random color (green, red);
  DOG.view(VIEW_BOTTOM);  //default viewing direction
  sample_screen();        //show content
  delay(2000);
}

void init_backlight(boolean mono)
{
  if(mono) //EA LED39X41-W, EA LED39X41-A
  {
    pinMode(led,  OUTPUT);
    mono_backlight(255);
  }
  else  //EA LED39X41-GR
  {
    pinMode(led_green, OUTPUT);
    pinMode(led_red,   OUTPUT);
    rgb_backlight(255,0);
  }
}
```

Listing 7.3 Quelltext Uno_DOGS102_Icon.ino – Auszug

Abschließend möchte ich noch erklären, wie das Logo links oben erstellt wird. Vom Hersteller EA können Sie hierzu unter der URL *https://www.lcd-module.de/fileadmin/html-seiten/deu/disk/Setup%20LCD-Tools%20Portable%204.8.exe* verschiedene Tools herunterladen, unter denen sich das Programm *BitmapEdit* befindet (siehe Abbildung 7.32).

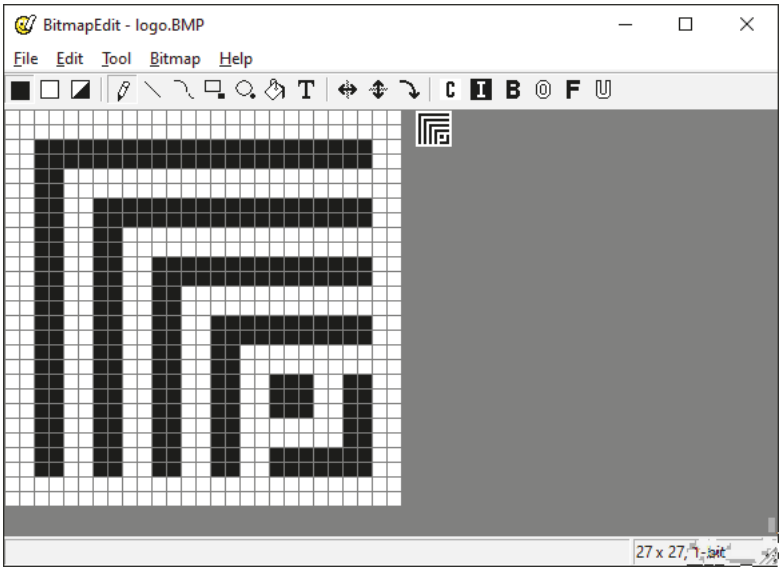

Abbildung 7.32 Der EA-Bitmap-Editor »BitmapEdit«

Mit diesem Programm kann ein Bitfeld (hier 27 × 27 dots) erstellt, beschrieben und als Headerdatei mit BMP-Daten exportiert werden. Abbildung 7.33 zeigt die erzeugte Headerdatei *logo_BMP.h* im Editor Notepad++.

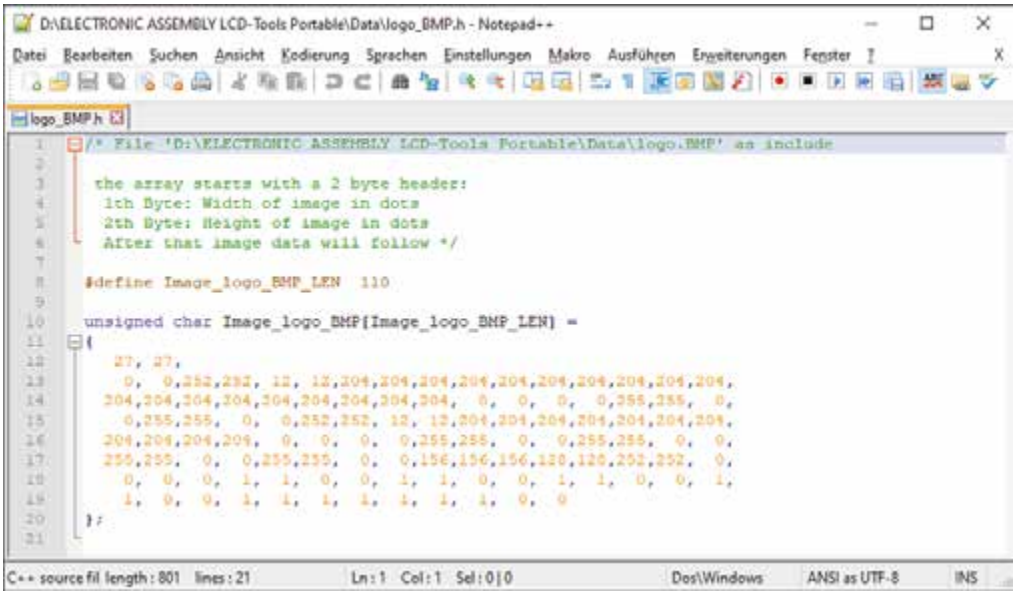

Abbildung 7.33 Das erzeugte Logo »logo_BMP.h«

Nach einem Klick auf das markierte Feld können Sie in der Arduino IDE über NEW TAB einen neuen Tab (Quelltextfenster) erstellen und die erzeugte Headerdatei hineinkopieren. Damit das Character-Array im Flash-Speicher abgelegt wird, müssen Sie das Attribut `__attribute__((section(".progmem.data")))` ergänzen. Abbildung 7.34 zeigt die so angepasste Headerdatei *logo_BMP.h* in der Arduino IDE.

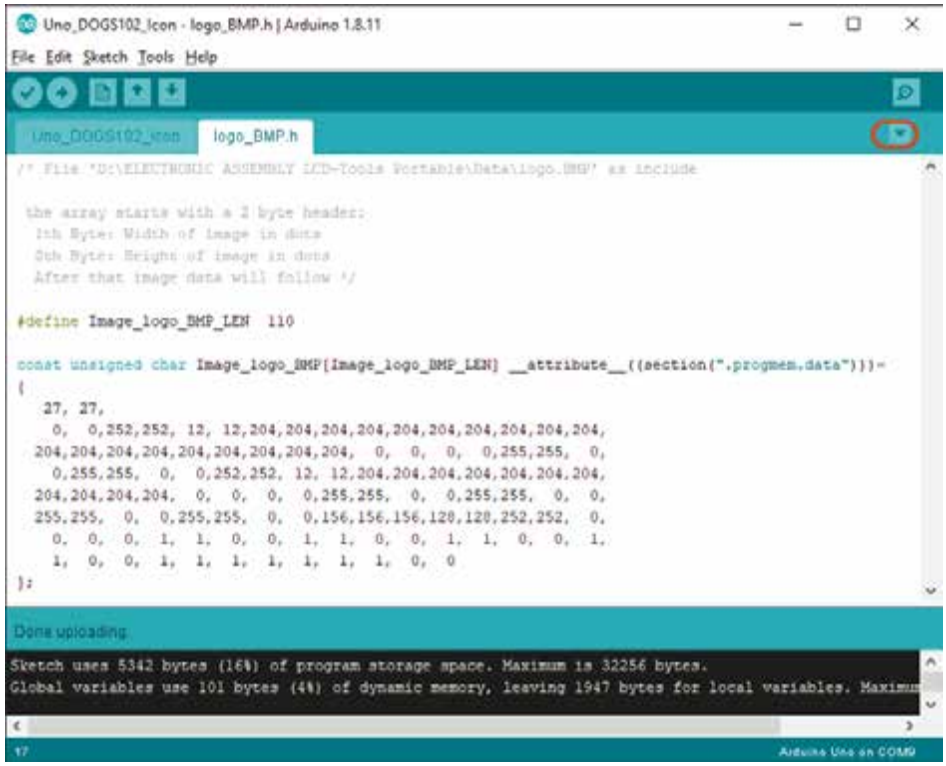

Abbildung 7.34 Die Datei »logo_BMP.h« in einem separaten Tab

7.5.5 Nokia-5110-Grafik-LCD

Mit einer Auflösung von 84 × 48 dots gehört das *Nokia 5110* zur Klasse der einfachen Grafik-LCDs. Es ist gerade in Maker-Kreisen stark verbreitet und für viele Anwendungen geeignet. Es stammt aus den Nokia-Mobiltelefonen älteren Datums und wird heute in der Regel auf einem Basisboard montiert angeboten (siehe Abbildung 7.35). Wenn Sie Glück haben, dann bekommen Sie dieses Display für unter 5 €.

Das Nokia 5110 verwendet einen *PCD8544*-Controller. Das ist ein Low-Power-CMOS-LCD-Controller/Treiber, der für eine grafische Anzeige von 48 Zeilen und 84 Spalten ausgelegt ist. Der PCD8544 kann über eine serielle Busschnittstelle mit einem Arduino verbunden werden.

Abbildung 7.35 Nokia5110 – Mobiltelefon und LCD-Modul

Es gibt zahlreiche Librarys für dieses LCD. Ich möchte Ihnen hier wegen der Einfachheit die *PCD8544*-Library von Carlos Rodrigues empfehlen, die Sie als Zip-Archiv von GitHub herunterladen und installieren können (*https://github.com/carlosefr/pcd8544*). Die Library erfordert die folgenden Verbindungen vom Nokia-5110-LCD zum Arduino Uno:

Display-Pin	Arduino-Uno-Pin
RST	I/O-Pin 6
CE	I/O-Pin 7
DC	I/O-Pin 5
DIN	I/O-Pin 4
CLK	I/O-Pin 3
VCC	+3,3 V
LIGHT	I/O-Pin 13
GND	GND

Tabelle 7.7 Verbindungen zwischen dem Nokia-5110-LCD und dem Arduino

Wenn Sie andere I/O-Pins am Arduino verwenden wollen, dann müssen Sie nur die Einstellungen in der Headerdatei *PCD8544.h* der Library anpassen. Wenn Sie aber den oben angegebenen Anschlussbedingungen folgen, erhalten Sie das Anschlussschema aus Abbildung 7.36.

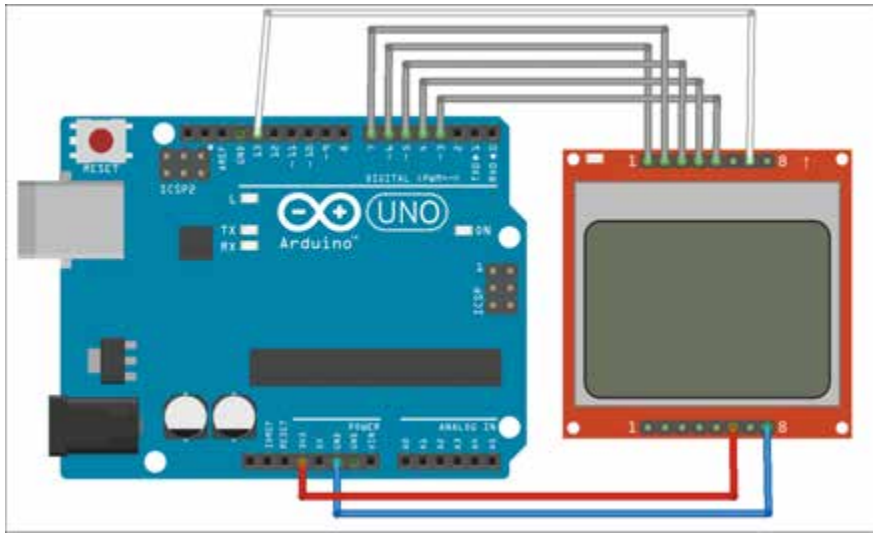

Abbildung 7.36 Nokia5110 am Arduino Uno

Das Programm *Uno_Nokia5110.ino* erzeugt die in Abbildung 7.37 gezeigte Darstellung einer Temperaturanzeige. Der Temperaturwert selbst wird hier durch eine Zufallszahl simuliert. Aus diesem Bild können Sie bereits die Möglichkeiten der eingesetzten Library im Zusammenspiel mit der Hardware schließen.

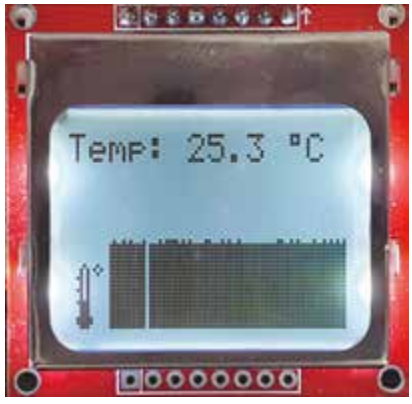

Abbildung 7.37 Ausgabe des Programms »Uno_Nokia5110.ino«

Neben der normalen Textanzeige werden ein Symbol (°) und eine Bitmap (Thermometer-Icon) erzeugt, außerdem wird der Temperaturverlauf als Chart grafisch dargestellt.

Ich erläutere Ihnen das Programm *Uno_Nokia5110.ino* wieder ausschnittsweise.

Zu Beginn erfolgt das Einbinden der verwendeten *PCD8544*-Library, gefolgt von einer Reihe von Anweisungen, die weitgehend selbsterklärend sein sollten. Mit dem Define `STARTSCREEN` wird die Möglichkeit gegeben, dem eigentlichen Programm einen Startbildschirm voranzustellen. Ich komme darauf gesondert zurück.

Dann folgen einige Konstantendefinitionen, bevor das LCD-Objekt erstellt wird. Den vollständigen Code finden Sie in Listing 7.4:

```
#include <PCD8544.h>

#define STARTSCREEN 1 // set to display a start screen

// Controlling backlight
static const byte ledPin = 13;
// The dimensions of the LCD (in pixels)...
static const byte LCD_WIDTH = 84;
static const byte LCD_HEIGHT = 48;
// The number of lines for the temperature chart...
static const byte CHART_HEIGHT = 5;
// A custom "degrees" symbol...
static const byte DEGREES_CHAR = 1;
static const byte degrees_glyph[] = { 0x00, 0x07, 0x05, 0x07, 0x00 };
// A bitmap graphic (10x2) of a thermometer...
static const byte THERMO_WIDTH = 10;
static const byte THERMO_HEIGHT = 2;
static const byte thermometer[] =
{ 0x00, 0x00, 0x48, 0xfe, 0x01, 0xfe, 0x00, 0x02, 0x05, 0x02,
  0x00, 0x00, 0x62, 0xff, 0xfe, 0xff, 0x60, 0x00, 0x00, 0x00};

static PCD8544 lcd;
```

Listing 7.4 Quelltex Uno_Nokia5110.ino – Auszug

Zu erläutern sind sicherlich die Konstanten `degrees_glyph[]` und die Bitmap `thermometer[]`.

Bei einer *Glyphe* handelt es sich um ein grafisch definiertes Zeichen, das eine bestimmte Position im Zeichensatz ersetzt. Hier wird das im ASCII-Zeichensatz nicht enthaltene Zeichen ° für die Ausgabe des Temperaturwertes in °C durch eine solche Glyphe als Sonderzeichen definiert.

Abbildung 7.38 zeigt die ASCII-Zeichentabelle, bei der ich nicht zu verändernde Zeichen markiert habe. Das betrifft in erster Linie die sichtbaren Zeichen des Zeichensatzes und einige Steuerzeichen.

HEX	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SOH	RS	.	>	N	^	n	~
F	SI	US	/	?	O		o	DEL

Abbildung 7.38 ASCII-Zeichen-Tabelle

Sie können aber durchaus dem Zeichen SOH (0x01) eine andere Bedeutung zuweisen. Genau das erfolgt durch diese Anweisung:

```
static const byte degrees_glyph[] = { 0x00, 0x07, 0x05, 0x07,  0x00 };
```

Die PCD8544-Library ermöglicht die Verwendung von benutzerdefinierten Bitmap-Symbolen (5 × 8), die durch ein Array von 5 Bytes definiert sind. Um das Erstellen benutzerdefinierter Symbole zu vereinfachen, können Sie einen grafischen *Glyph Editor* unter *http://carlosefr.github.io/pcd8544/* online verwenden. Abbildung 7.39 zeigt die Definition des Zeichens ° in diesem Editor. Das so definierte Byte-Array kann dann in den Quelltext übernommen werden.

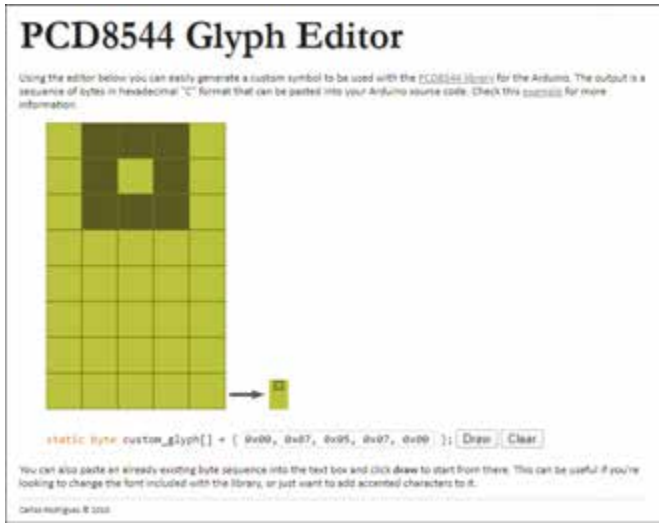

Abbildung 7.39 Der »PCD8544 Glyph Editor«

Beim Erzeugen einer Bitmap können Sie wieder so vorgehen, wie Sie es bereits in Abschnitt 7.5.4 gelernt haben.

Wenn Sie beachten, dass sich die Bitmap immer zeilenweise aus 8 Bit zusammensetzt, können Sie Ihre Bitmap im Programm *BitmapEdit* definieren. Für das Thermometer sieht die Bitmap dann so wie in Abbildung 7.40 aus. Abbildung 7.41 zeigt die vom Programm BitmapEdit exportierte BMP-Datei.

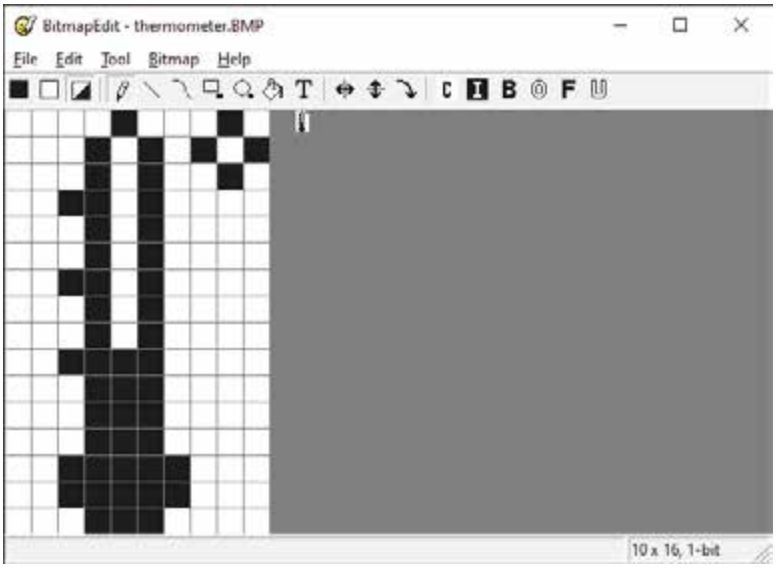

Abbildung 7.40 Die Bitmap für das Thermometer

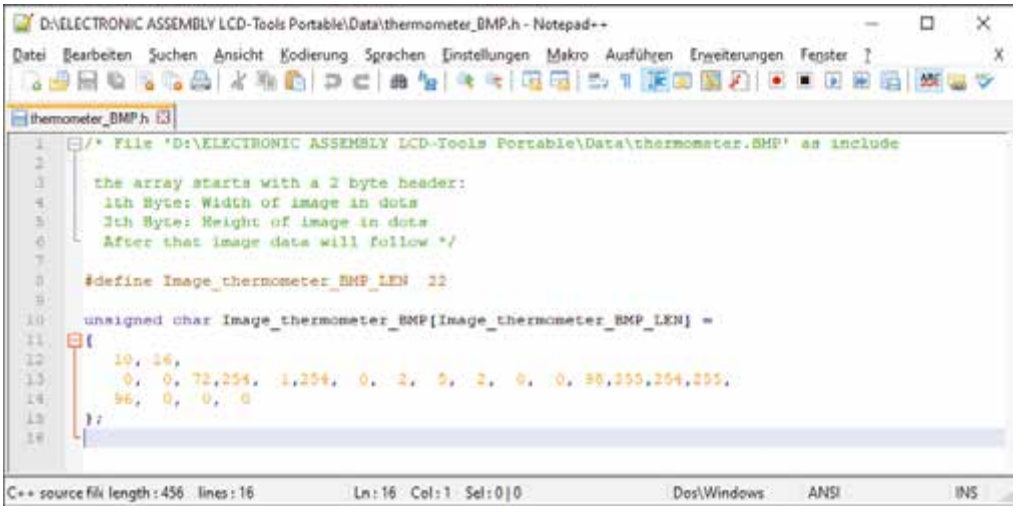

Abbildung 7.41 Die Bitmap-Datei im Editor

Das erzeugte Character-Array in der Headerdatei *thermometer_BMP.h* muss noch von den Headerbytes befreit werden und kann dann in den Quelltext des Programms *Uno_Nokia5110.ino* kopiert werden. In der Anweisung

```
static const byte thermometer[] =
{ 0x00, 0x00, 0x48, 0xfe, 0x01, 0xfe, 0x00, 0x02, 0x05, 0x02,
  0x00, 0x00, 0x62, 0xff, 0xfe, 0xff, 0x60, 0x00, 0x00, 0x00};
```

sind genau diese Zahlen (allerdings in hexadezimaler Darstellung) zu sehen.

In der Funktion `setup()` (Listing 7.5) kann nun das LCD initialisiert werden. Das grafisch erzeugte Zeichen ° wird der Position 1 im Zeichensatz zugewiesen, und über das I/O-Pin 13 wird die Hintergrundbeleuchtung des LCDs eingeschaltet. Bevor möglicherweise der Startbildschirm aufgerufen wird, erfolgt noch die Initialisierung des (Pseudo-)Zufallszahlen-Generators.

```
void setup()
{
  lcd.begin(LCD_WIDTH, LCD_HEIGHT);

  // Register the custom symbol...
  lcd.createChar(DEGREES_CHAR, degrees_glyph);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  randomSeed(analogRead(0));

  if (STARTSCREEN) displayStartScreen();
}
```

Listing 7.5 Quelltext Uno_Nokia5110.ino – setup()

In der Funktion `loop()` (Listing 7.6) wird nun der simulierte Temperaturwert als Text ausgegeben und das Thermometersymbol links unten ausgegeben. Die Anweisung `lcd.print(" \001C ")` ruft das dem ASCII-Code `0x01` zugeordnete Zeichen ° auf, wodurch die gesamte Anweisung die Zeichen °C am LCD zur Anzeige bringt. Zum Schluss wird noch der Temperatur-Chart aktualisiert. Dieser wird durch die Anweisung `lcd.drawColumn(CHART_HEIGHT, value)` erzeugt, wodurch ein auf die `CHART-HEIGHT` normierter Wert als Balken dargestellt wird.

Kapitel 11
Datenformate und Kommunikationsprotokolle

Daten sind eigentlich nur dann interessant, wenn sie ausgetauscht und ausgewertet werden können. Je besser diese Vorgänge standardisiert und strukturiert sind, desto einfacher ist die Verarbeitung.

In diesem Kapitel stelle ich Ihnen beispielhaft das JSON-Datenformat und das MQTT-Kommunikationsprotokoll vor. Beide spielen vor allem beim Zugriff auf bzw. über das Internet eine wichtige Rolle. Ich werde beide in Programmen noch einsetzen, sodass Sie hier erste Grundlagen dazu finden.

11.1 JSON

JSON steht für *JavaScript Object Notation* und ist ein Textformat für den Austausch von Daten zwischen Anwendungen. JSON ist das beliebteste Austauschformat, denn es ist gut lesbar, einfach, schnell und erzeugt nur wenig Overhead.

Ein JSON-Objekt ist eine Sammlung von Schlüssel-Wert-Paaren (*Key Value Pairs*). JSON-Schlüssel sind Strings in doppelten Hochkommas. Links steht der Schlüssel, rechts der Wert.

Wie sich JSON zur Formatierung von Sensorausgaben einsetzen lässt, zeigt die folgende JSON-Ausgabe:

```
{
  "sensor1": { "temperature" : 22.2 , "humdity" : 66 , "pressure" : 999 },
  "sensor2": { "temperature" : 22.2 , "humdity" : 66  },
  "sensor3": { "position" : [8.8172433, 47.1979687]}
}
```

Mit *ArduinoJson* steht eine Arduino-Library von Benoît Blanchon zur Verfügung, die die Arbeit mit JSON sehr effektiv unterstützt (*https://github.com/bblanchon/ArduinoJson*).

Das in Listing 11.1 gezeigte Programmbeispiel *Uno_JsonOutputv6.ino* erzeugt den oben angegebenen JSON-Output für die (hier fiktiven) Sensoren BME280, DHT11

und NEO6M GPS. Die *ArduinoJson*-Library ist das eigentliche Tool. Im Programm wird die Nutzung der Library gezeigt.

Der *ArduinoJson Assistant* (*https://arduinojson.org/v6/assistant/*) hilft, die Größe des JSON-Buffers festzulegen, und bietet Unterstützung bei der Anwendung von ArduinoJson. Aktuell liegt ArduinoJson in der Version v6 vor. Achten Sie unbedingt auf die Version, denn es sind noch viele Programme im Umlauf, die die veraltete Version v5 nutzen. Kompatibilität ist dann nicht gegeben!

Nach dem Einbinden der ArduinoJson-Library werden hier noch einige Ausgaben zu Programmnamen, zum Kompilierzeitpunkt und zu den eingesetzten Versionen der Arduino IDE und der ArduinoJson-Library gemacht. Die Ausgabe dieser Informationen kann (in jedem Programm) sehr hilfreich sein, ist aber Geschmacksache.

In der Funktion `setup()`, die hier alle Programmaktivitäten beherbergt, wird ein dynamisches JSON-Objekt `doc` erzeugt. Hilfe bei der Dimensionierung bietet der *ArduinoJson Assistant*.

In diese Struktur können nun die *Nested Objects* mit ihren Key-Value-Paaren eingebaut werden. Am Ende des Programms erfolgt die JSON-Ausgabe zum Serial Monitor:

```
#include <ArduinoJson.h>

// helper macro
#define LINE(name,val) Serial.print(name); Serial.print(" ");
Serial.println(val);

#define SENSOR1 "BME280"
#define SENSOR2 "DHT11"
#define SENSOR3 "NEO6M_GPS"

void setup()
{
  // Initialize Serial port
  Serial.begin(115200);
  while (!Serial) continue;

  LINE("File", __FILE__);
  String s1 = __DATE__;
  String s2 = __TIME__;

  // Date of compilation
  LINE("Compilation @", s1 +" " +s2);
```

```
  // Arduino IDE SW version
  LINE("ARDUINO IDE", ARDUINO);

  Serial.print("Test JSON using ArduinoJson v.");
  Serial.println(ARDUINOJSON_VERSION);

  //https://arduinojson.org/v6/assistant/ used for configuration
  const size_t capacity = JSON_ARRAY_SIZE(2) +JSON_OBJECT_SIZE(1) +
       JSON_OBJECT_SIZE(2) +2*JSON_OBJECT_SIZE(3);
  DynamicJsonDocument doc(capacity);

  Serial.println("Serialization of data...");

  JsonObject sensor1 = doc.createNestedObject("sensor1");
  sensor1["temperature"] = 22.2;
  sensor1["humdity"] = 66;
  sensor1["pressure"] = 999;

  JsonObject sensor2 = doc.createNestedObject("sensor2");
  sensor2["temperature"] = 22.2;
  sensor2["humdity"] = 66;

  JsonObject sensor3 = doc.createNestedObject("sensor3");

  JsonArray sensor3_position = sensor3.createNestedArray("position");
  sensor3_position.add(8.8172433);
  sensor3_position.add(47.1979687);

  Serial.println("\nOutput minified JSON...");
  serializeJson(doc, Serial);

  Serial.println("\n\nOutput prettified JSON...");
  serializeJsonPretty(doc, Serial);
}

void loop() {};
```

Listing 11.1 Quelltext Uno_JsonOutputv6.ino

Wie die Terminalausgabe des Programms *Uno_JsonOutputv6.ino* in Abbildung 11.1 zeigt, gibt es einmal den *minified JSON Output* (minimale Ausgabe) und zum anderen den *prettified JSON Output* (»schöne« Ausgabe), der sicher etwas einfacher lesbar ist. Entscheiden Sie, welche Variante Sie bevorzugen.

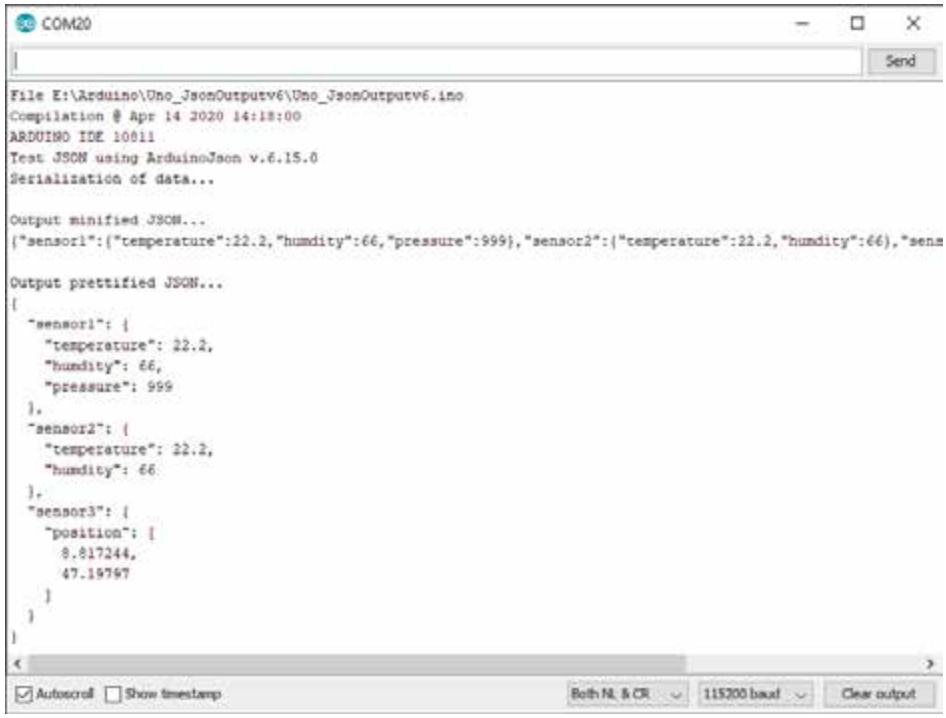

Abbildung 11.1 Terminalausgaben im JSON-Format

11.2 MQTT

MQTT (*Message Queuing Telemetry Transport*) ist ein äußerst simpel aufgebautes Publish/Subscribe-Protokoll für den Nachrichtenaustausch zwischen Geräten geringer Funktionalität.

Das robuste MQTT-Protokoll wurde für unzuverlässige Netze mit geringer Bandbreite und hoher Latenzzeit entwickelt.

11.2.1 Grundlagen

MQTT minimiert die genutzte Netzwerk-Bandbreite und die Anforderungen an Geräte – gleichzeitig wird für die Datenübermittlung eine hohe Zuverlässigkeit erreicht.

Diese Anforderungen bestehen insbesondere bei Sensornetzwerken, bei der Machine-to-Machine-Communication (*M2M*), in der Telemedizin, der Patientenüberwachung und allgemein beim IoT. Bei dieser Anwendung sind die angeschlossenen Geräte immer verbunden und kommunizieren ständig miteinander.

Seit Januar 2016 ist MQTT ISO/IEC-standardisiert als *ISO/IEC PRF 20922*.

Die Struktur, in der das MQTT-Protokoll arbeitet, besteht aus der Datenquelle, die als *Publisher* bezeichnet wird, der Datensenke, dem *Subscriber*, und dem zwischengeschalteten *Message Broker*, der für die Kommunikationssteuerung sorgt. Abbildung 11.2 zeigt das betreffende Modell.

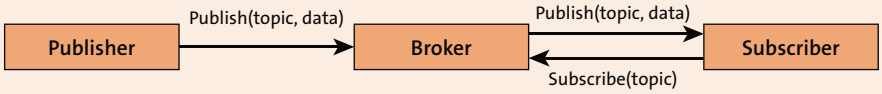

Abbildung 11.2 Publish/Subscribe Message Queueing Model

Wie Abbildung 11.2 zeigt, kommunizieren der Publisher und der Subscriber immer über den Broker. Die Nachrichten, die über diesen Kanal laufen, sind sogenannten *Topics* zugeordnet.

Nachdem sich ein Subscriber erfolgreich am Broker angemeldet hat, teilt er diesem mit, welche Topics er abonnieren möchte, um in Zukunft mit den jeweiligen Nachrichten versorgt zu werden (`Subscribe(topic)`).

Beim Publisher funktioniert das umgekehrt: Beim Versenden einer Nachricht teilt er dem Broker mit, zu welchem Topic diese gehört (`Publish(topic, data)`).

Die Hauptaufgabe des Brokers besteht also darin, Nachrichten anzunehmen und diese entsprechend an die Subscriber weiterzuleiten.

Topics sind einfache Zeichenketten, die eine Nachrichten-Hierarchie abbilden und mit einer einfachen URL vergleichbar sind. So kann ein Topic für einen Sensor, der Temperatur und Luftfeuchtigkeit ermittelt, beispielsweise folgendermaßen aufgebaut sein:

```
sensor/ID/temperature
```

```
sensor/ID/humidity
```

Der so ausgestattete Sensorknoten kann nach Verbindung mit dem MQTT-Broker diese Topics an den MQTT-Broker senden (publish). Der MQTT-Broker stellt diese Information dann allen IoT-Knoten bereit, die das betreffende Topic abonniert haben.

Haben wir nun mehrere gleichartige Sensorknoten in unserem Netz, dann werden an den MQTT-Broker Messages versendet, die sich nur in den IDs unterscheiden. Mit Wildcards können diese Topics gefiltert werden.

Durch Einführung der Wildcards `+` und `#` lassen sich die Messages entsprechend filtern. Wenn ein Subscriber den Topic `DHT11/+/temperature` abonniert, dann wird er die Temperaturwerte von allen vorhandenen DHT11-Sensorknoten erhalten. Sollen alle

im Netz vorhandenen Werte von allen vorhandenen DHT11-Sensoren erfasst werden, dann ist der Topic `DHT11/+/#` zu abonnieren.

MQTT wird oft in unsicheren Netzwerken eingesetzt, und es kann geschehen, dass IoT-Knoten nicht mehr mit dem Netzwerk verbunden sind. Die *Last-Will-and-Testament*-(LWT-)Mitteilung wird in MQTT verwendet, um andere Netzwerkknoten über den Verbindungsverlust zu informieren. Jeder MQTT-Client kann bei der Verbindung mit einem Broker seine LWT-Mitteilung spezifizieren. Der Broker speichert diese Mitteilung und versendet sie erst, wenn der betreffende Client unerwartet nicht mehr erreichbar ist.

Wenn Sie diese Grundlagen kennen, werden Sie die folgenden Ausführungen sicher schon verstehen. Falls Sie mehr in die Tiefe gehen wollen, kann ich Ihnen die umfangreichen Ausführungen in den *MQTT Essentials* (*http://www.hivemq.com/blog/mqtt-essentials/*) empfehlen.

Um sich praktisch mit dem MQTT-Protokoll bekannt zu machen, muss ein MQTT-Broker zur Verfügung stehen. Im nächsten Abschnitt finden Sie verschiedene Möglichkeiten dazu.

11.2.2 MQTT-Broker

Jede MQTT-Kommunikation im IoT läuft über einen MQTT-Broker. Setzen wir auf ein abgeschlossenes Netzwerk, dann kann der MQTT-Broker auf einem PC oder Linux-Device im eigenen Netz installiert werden (*Self Hosted MQTT Broker*). Für den Aufbau eines solchen MQTT-Brokers existieren verschiedene Softwarelösungen.

Mosquitto ist einer der verbreitetsten MQTT-Broker. Aktuell ist die Version 2.0.15 verfügbar und kann von *http://mosquitto.org/download/* heruntergeladen werden.

Der Raspberry Pi ist eine ausgezeichnete Mosquitto-Plattform und kann so den Sensoren einen MQTT-Broker zur Verfügung stellen. Eine Anleitung zur schrittweisen Installation von Mosquitto auf dem Raspberry Pi ist unter *http://blog.thingstud.io/recipes/how-to-make-your-raspberry-pi-the-ultimate-iot-hub/* zu finden. Ich werde hier nicht diesen Weg gehen.

Eine andere Variante ist, einen MQTT-Broker zu verwenden, der als Cloud-Applikation zur Verfügung steht (*Hosted MQTT Brokers*). Diese Variante hat den Vorteil, dass man die Grenzen seines abgeschlossenen Netzwerkes verlässt und die Daten über das Internet bereitstellt.

Auf der Website *mqtt.org* gibt es eine Liste öffentlich zugänglicher MQTT-Broker (*https://github.com/mqtt/mqtt.github.io/wiki/public_brokers*). Um sich an den Datenaustausch über das MQTT-Protokoll heranzutasten, bietet sich die Verwendung eines freien Broker-Dienstes als Spielwiese an.

Der *CloudMQTT*-Broker der schwedischen Firma *84codes AB* ist eine solche Möglichkeit. CloudMQTT sind Mosquitto-Server in der Cloud.

Zum Erstellen einer CloudMQTT-Instanz ist es erforderlich, unter *http://www.cloudmqtt.com/* ein Konto einzurichten und sich für einen Kunden-Plan zu entscheiden. Als Testfeld nutze ich den freien Plan *Cute Cat*. Die Anmeldung eines Kundenkontos erfolgt über eine E-Mail-Adresse, an die ein Link zur Freischaltung verschickt wird.

Nach der Erzeugung einer CloudMQTT-Instanz werden die Informationen zur erzeugten Instanz angezeigt. Alle in Abbildung 11.3 gezeigten Daten werden vom System zugewiesen. Das trifft auch für den Usernamen und das Passwort zu.

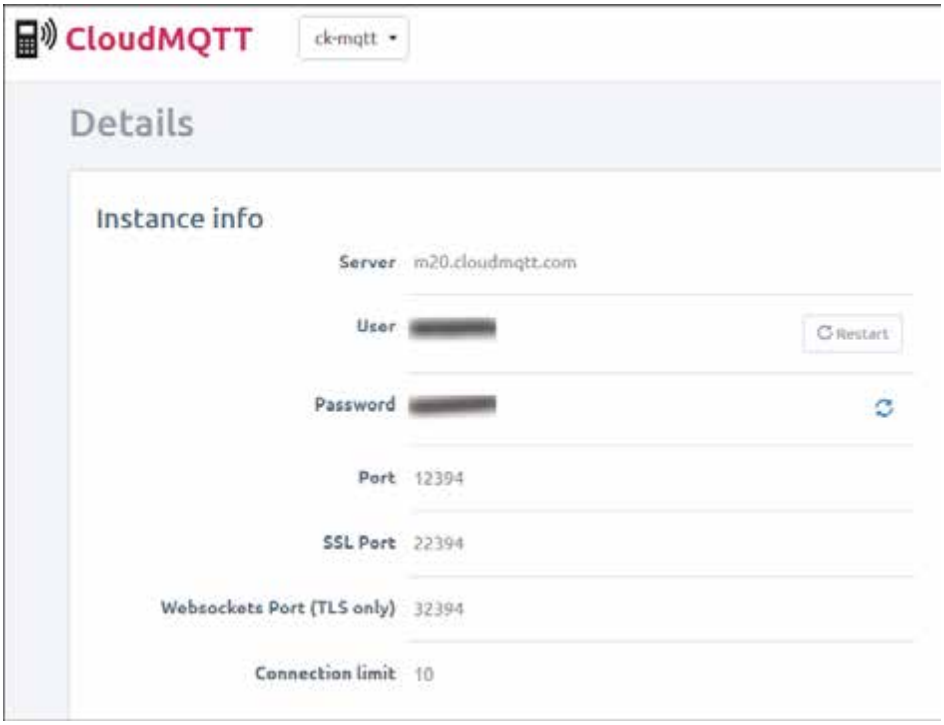

Abbildung 11.3 CloudMQTT-»Instance Info«

Bevor über das MQTT-Protokoll Daten ausgetauscht werden können, müssen sogenannte *ACL Rules* definiert werden. ACL steht für *Access Control List*.

Damit sind serverseitig alle Vorbereitungen getroffen, und der Datenaustausch kann nach Einrichtung des Sensorknotens (MQTT-Client, Publisher) im *Websocket*-UI (User Interface) getestet werden.

Abbildung 11.4 zeigt den Zugriff auf den CloudMQTT-Broker über das Websocket-UI. Abonniert wurde der Topic *UnoWiFiR2/#*, was durch die empfangenen Messages auch repräsentiert wird.

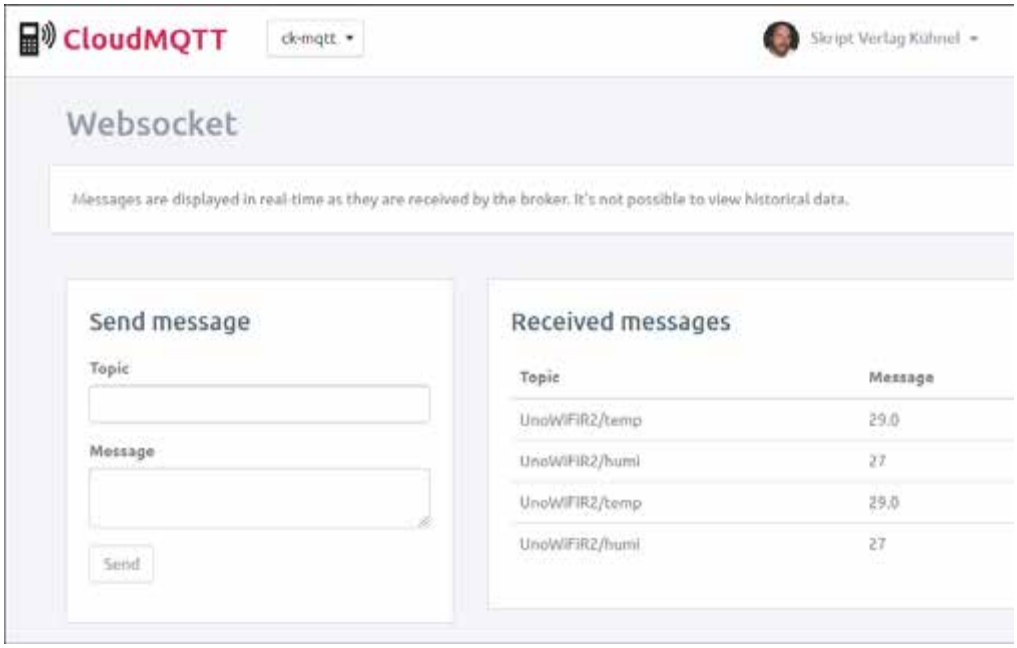

Abbildung 11.4 Empfangene MQTT-Messages von einem Sensor

11.2.3 MQTT-Client

Damit der MQTT-Broker auch die in Abbildung 11.4 gezeigten Nachrichten empfangen kann, stelle ich Ihnen hier einen einfachen MQTT-Client vor. Ich habe einfach einen *Arduino Uno WiFi Rev2* und ein *YwRobot Easy Module Shield v1* zusammengesteckt, um den DHT11-Sensor nutzen zu können. Hardwareseitig sind damit alle Vorkehrungen für einen Sensorknoten getroffen, der MQTT-Messages verschicken kann.

Das Programm *UnoWiFiR2_DHT11_MQTT.ino* fragt den DHT11-Sensor ab (oder auch einen beliebigen anderen), verpackt die Resultate in die zu versendende Nachricht und verschickt diese. Mit den ersten Zeilen des Quelltextes sind auch alle Bestandteile für die einzubindenden Librarys benannt. Die ersten beiden Librarys unterstützen den WiFi-Zugriff auf den Router. Die *PubSubClient*-Library ist für MQTT verantwortlich. Die *DHT*-Library unterstützt den Zugriff auf einen Sensor der DHT11-Familie, und in *arduino_secrets.h* sind die zu behütenden Geheimnisse, wie Zugangsdaten etc., abgelegt.

```
#include <SPI.h>
#include <WiFiNINA.h>
#include <PubSubClient.h>
#include "DHT.h"
#include "arduino_secrets.h"
```

Mit den Defines werden wieder Info- und Debug-Ausgaben gesteuert:

```
#define INFO 0
#define DEBUG 1
```

Mit dem Erzeugen des PubSubClients sind wir dann schon nahe am eigentlichen Kern:

```
/* create an instance of PubSubClient client */
WiFiClient UnoWiFiClient;
PubSubClient client(UnoWiFiClient);
```

Der Sensor DHT11 liefert zwei Messwerte, denen je ein Topic zugeordnet wird:

```
/* topics */
#define TEMP_TOPIC  "UnoWiFiR2/temp"
#define HUMI_TOPIC  "UnoWiFiR2/humi"
```

Bleibt noch das Konfigurieren und Initialisieren des Sensors vor dem eigentlichen `setup()`:

```
/* define DHT pins */
#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
```

In der Funktion `setup()` (Listing 11.2) werden die WiFi- und die MQTT-Verbindung aufgebaut und der Sensor initialisiert. Die Funktionen sind mehr oder weniger vorgegeben und werden in der Anwendung in der Regel so belassen:

```
void setup()
{
  …
  connectWiFi();
  configureMQTT();
  initSensor();

  if (DEBUG) Serial.println(F("Running..."));     //last line in setup()
}
```

Listing 11.2 Quelltext UnoWiFiR2_DHT11_MQTT.ino – setup()

In der Funktion `loop()` (Listing 11.3) spielt sich nach dem Verbindungsaufbau alles Weitere ab. Alle 30 Sekunden wir der DHT11-Sensor nach seinen Messwerten abge-

fragt. Handelt es sich um die erwarteten numerischen Resultate, dann werden diese in einen String umgewandelt und als MQTT-Message versendet:

```
void loop()
{
  /* if client was disconnected then try to reconnect again */
  if (!client.connected()) mqttconnect();
  /* this function will listen for incomming subscribed topic-process-
  invoke receivedCallback */
  client.loop();
  /* we measure temperature every 10 secs
  we count until 10 secs reached to avoid blocking program if using delay()*/
  long now = millis();
  if (now - lastMsg > 30000)
  {
    lastMsg = now;
    /* read DHT11/DHT22 sensor and convert to string */
    temp = dht.readTemperature();
    hum  = dht.readHumidity();
    if (!isnan(temp))
    {
      dtostrf(temp, 3, 1, msg);
      /* publish the message */
      client.publish(TEMP_TOPIC, msg);
      if (DEBUG) Serial.print(msg);
    }
    if (!isnan(hum))
    {
      dtostrf(hum, 3, 0, msg);
      /* publish the message */
      client.publish(HUMI_TOPIC, msg);
      if (DEBUG) Serial.println(msg);
    }
  }
}
```

Listing 11.3 Quelltext UnoWiFiR2_DHT11_MQTT.ino – loop()

Die schrittweise Abarbeitung des Programms können Sie leicht anhand der Ausgaben in Abbildung 11.5 nachverfolgen:

Die vom MQTT-Broker empfangenen Meldungen können nun von einem weiteren MQTT-Client abonniert (*subscribe*) werden.

Auf meinem Windows-PC verwende ich gern die Software *MQTTlens* und auf dem Smartphone *MQTT Dashboard*. Beide Programme können gratis heruntergeladen werden.

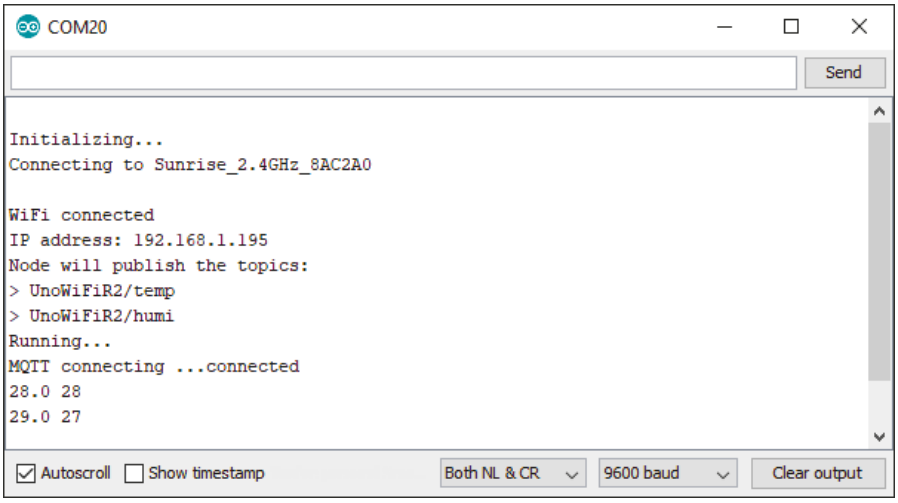

Abbildung 11.5 Ausgaben des Programms »UnoWiFiR2_DHT11_MQTT.ino«

Den Mittschnitt von MQTTlens zeigt Abbildung 11.6. Die Subscription lautet hier `UnoWiFiR2/#`. Das heißt, alle Topics, die mit `UnoWiFiR2/` beginnen, sind abonniert und werden angezeigt.

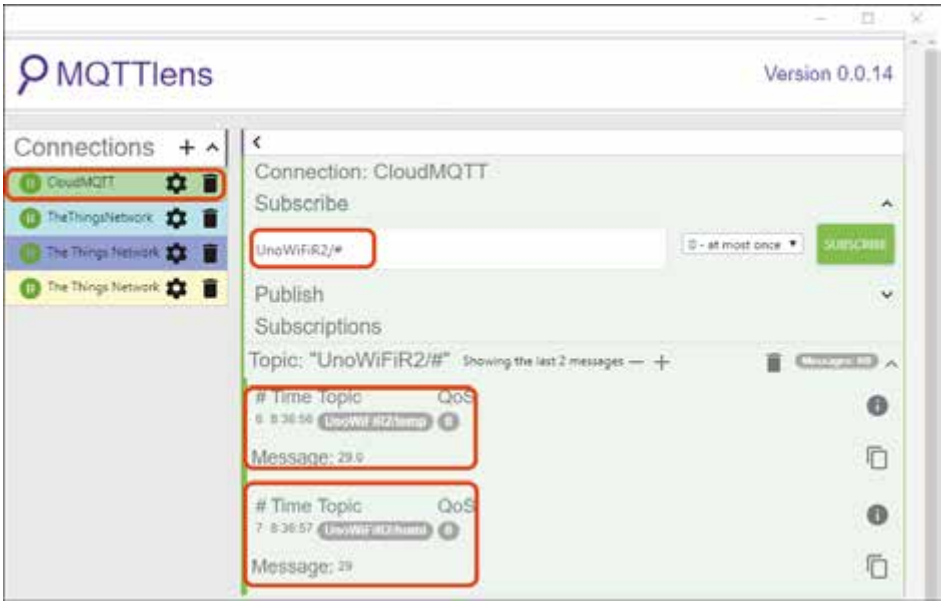

Abbildung 11.6 Mitschnitt von MQTT-Messages mit MQTTlens

Einen MQTT-Client auf Basis eines ESP32 habe ich in meinem Blog beschrieben: *HiGrow-Sensor: Daten erfassen und versenden, https://ckblog2016.net/2018/03/19/higrow-sensor-daten-erfassen-und-versenden/*. Wenn Sie also die Umgebungsbedingungen für Ihre Pflanzen überwachen wollen, dann ist das eine Möglichkeit.

Achten Sie aber darauf, dass die Bodenfeuchte kapazitiv gemessen wird: Bei resistiver Messung haben Sie nicht lange Freude, denn der Sensor zerstört sich durch Korrosion langsam selbst.