edv-buchversand.de

Arduino
Das umfassende Handbuch

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

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.

So weit erst einmal eine verkürzte Erklärung dazu, wie der Hersteller selbst sein Arduino-System positioniert. Ich möchte diese Auflistung noch 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 eingesetzt wird – sowohl von Makern als auch von Ingenieuren und sogar von großen Unternehmen. 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. Darauf gehe ich später in Kapitel 2, »Die Arduino-Hardware«, noch detaillierter ein.

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 die 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 Symbol, das Sie in Abbildung 1.1 sehen.

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.sch* und *UNO-TH_Rev3e.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, 14-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 Clones, 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 dass 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/t/lgpl-and-arduino-in-commercial-products*).

Schwieriger wird es dann bei Bibliotheken (*Librarys*), die nicht von Arduino selbst in das Arduino-Gesamtpaket integriert wurden. So sind 2024 in der *Arduino Library List* (*https://www.arduinolibraries.info*) Arduino-Librarys mit den folgenden Lizenzen zu finden:

- OBSD (2)
- AGPL 3.0 (34)
- Apache 2.0 (275)
- Artistic 2.0 (6)
- BSD 2 Clause (34)
- BSD 3 Clause (213)
- BSD 3 Clause Clear (3)
- BSL 1.0 (2)
- CC BY 4.0 (2)
- CC BY SA 3.0 (1)
- CC BY SA 4.0 (6)
- CC0 1.0 (25)
- GPL 2.0 (88)
- GPL 3.0 (675)
- GPL 3.0 only (1)
- GPL 3.0 or later (2)
- ISC (4)
- LGPL 2.1 (193)
- LGPL 2.1 or later (1)
- LGPL 3.0 (215)
- MIT (2538)
- MPL 2.0 (12)

- MS PL (1)
- NOASSERTION (629)
- OSL 3.0 (1)
- Unlicense (34)
- 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 sowie 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« habe ich das Zusammenspiel der verschiedenen Beteiligten beschrieben. Sie finden den ersten und zweiten Teil des Beitrags unter folgenden Links:

- Teil 1: *http://www.elektroniknet.de/embedded/hardware/artikel/127588/*
- Teil 2: *http://www.elektroniknet.de/embedded/hardware/artikel/130181/*

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 und den zunehmend zum Einsatz kommenden KI-Chatbots 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, 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.

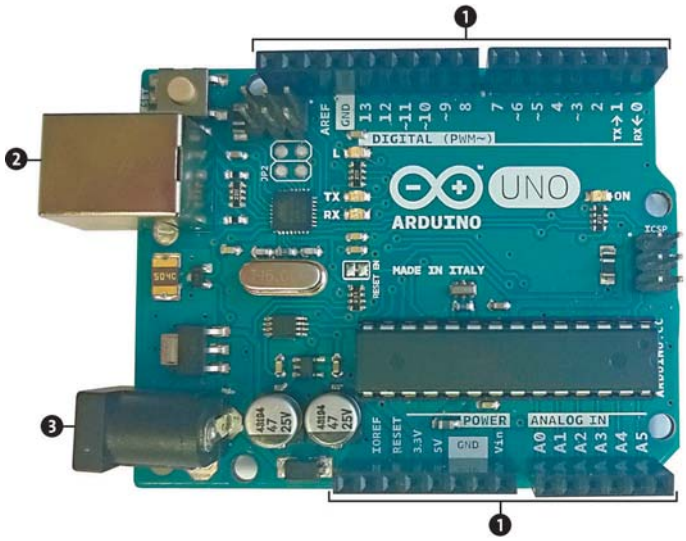

Abbildung 1.2 Arduino Uno Rev3

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 (Headern) 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.

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.

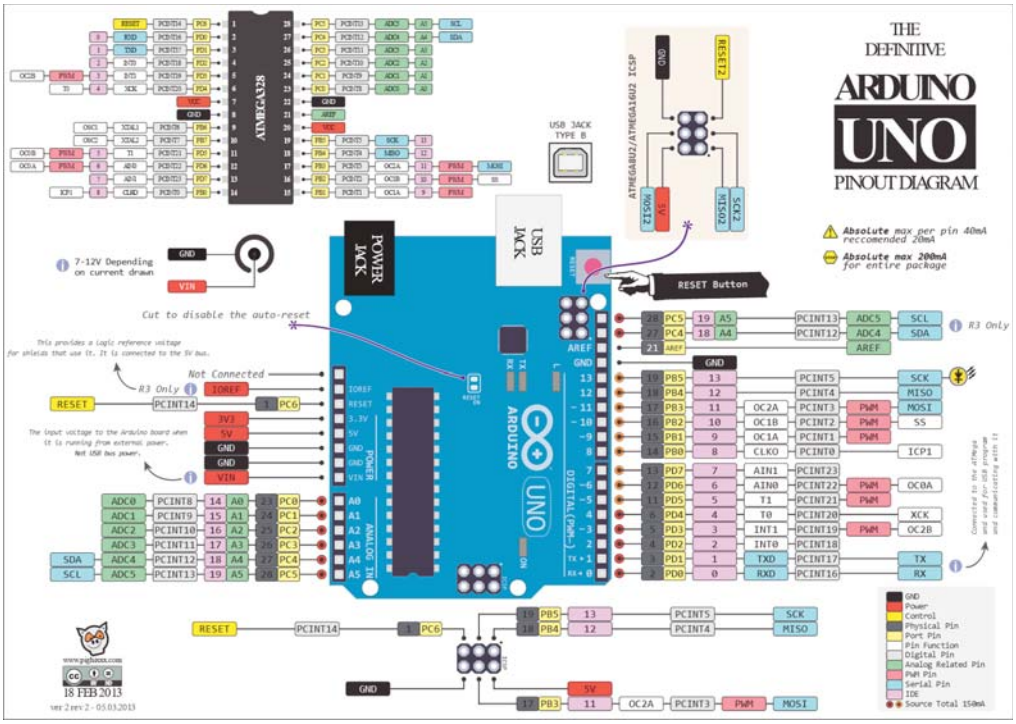

Abbildung 1.3 Pinbelegung beim Arduino Uno Rev3

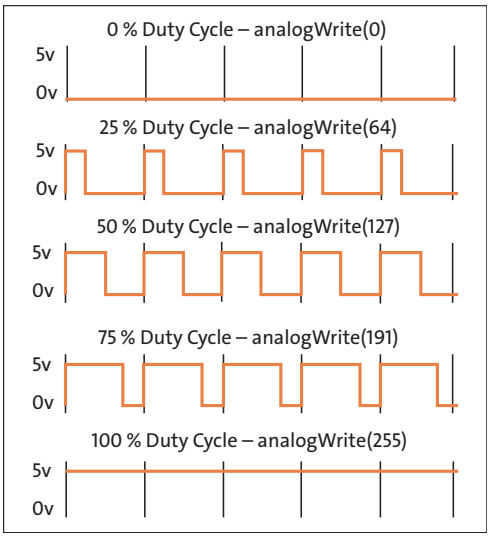

Abbildung 1.4 PWM

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 verwenden. Den bekommen Sie für 5,40 € bei Reichelt. Der ATmega328P ohne Bootloader kostet bei Reichelt nur 3,33 €:

- *https://www.reichelt.de/arduino-atmega328-mit-arduino-bootloader-ard-atmega-328-p230602.html*
- *https://www.reichelt.de/8-bit-atmega-avr-mikrocontroller-32-kb-20-mhz-pdip-28-atmega-328p-pu-p119685.html*

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 ich auch schon angedeutet habe, dass es weitere Arduinos gibt, möchte ich noch kurz etwas zur Namensgebung schreiben.

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 liegt in Version 1.8 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

Damit Sie den Arduino besser verstehen können, will ich Ihnen einige Grundlagen zum Mikrocontroller allgemein erläutern. Wenn Ihnen das Folgende 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. Gern können Sie später hierher zurückkehren, 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 Der Mikrocontroller-Kern

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

Das *ATmega328P Data Sheet* können Sie von *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.

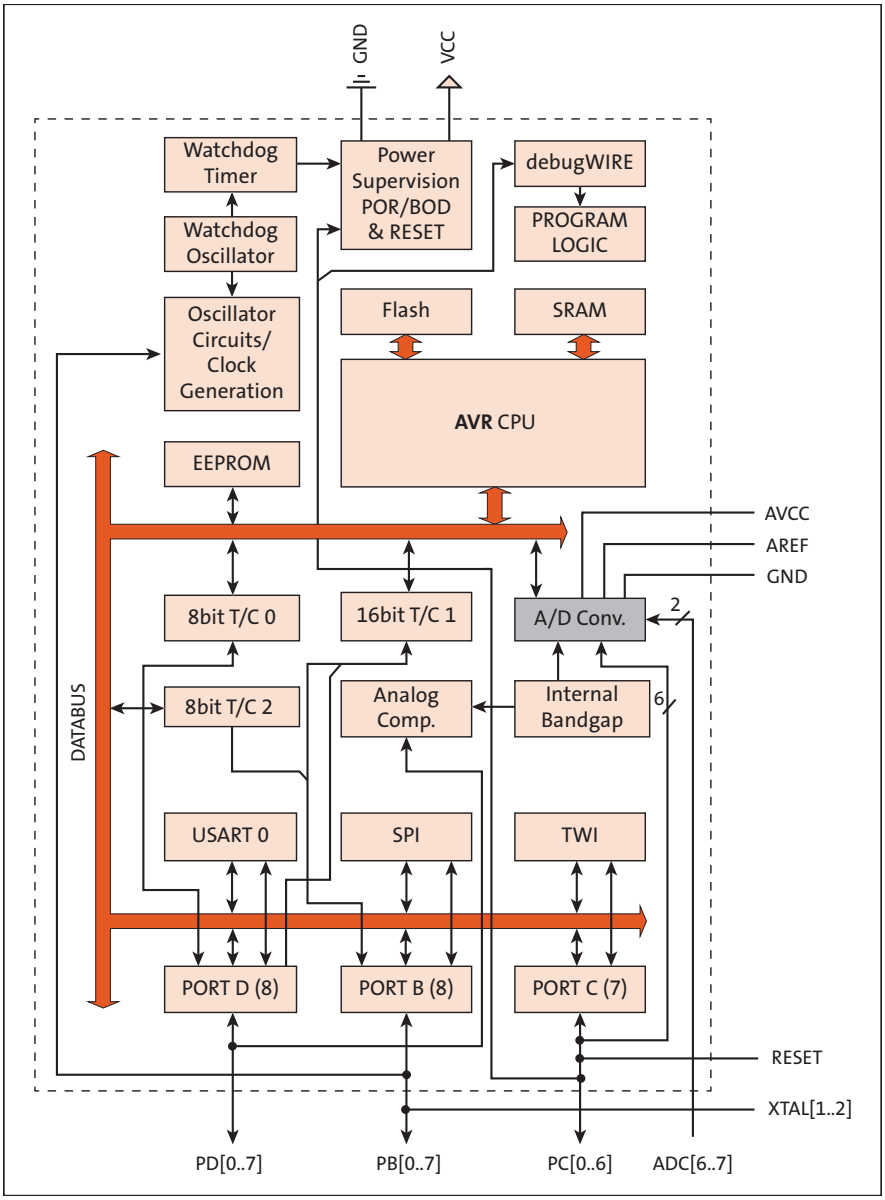

Abbildung 1.5 ATmega328P-Blockdiagramm (Quelle: ATmega328 Data Sheet)

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

Tabelle 1.2 Speicheraufteilung des SRAM beim ATmega328P

Speicher	Adressbereich
160 Extended I/O-Register	0x0060–0x00FF
Internes SRAM (2048 × 8 Bit)	0x0100–0x08FF

Tabelle 1.2 Speicheraufteilung des SRAM beim ATmega328P (Forts.)

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

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, finden Sie unter *https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/arduino-bootloader-brennen*. Eine gewisse Erfahrung beim Umgang mit einem Arduino ist hierzu aber Voraussetzung, weshalb ich Ihnen rate, zumindest anfangs die wenigen Euro für den bereits programmierten Bootloader auszugeben.

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 zu bearbeiten. (Interrupts sind Unterbrechungen des laufenden Programms.)

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

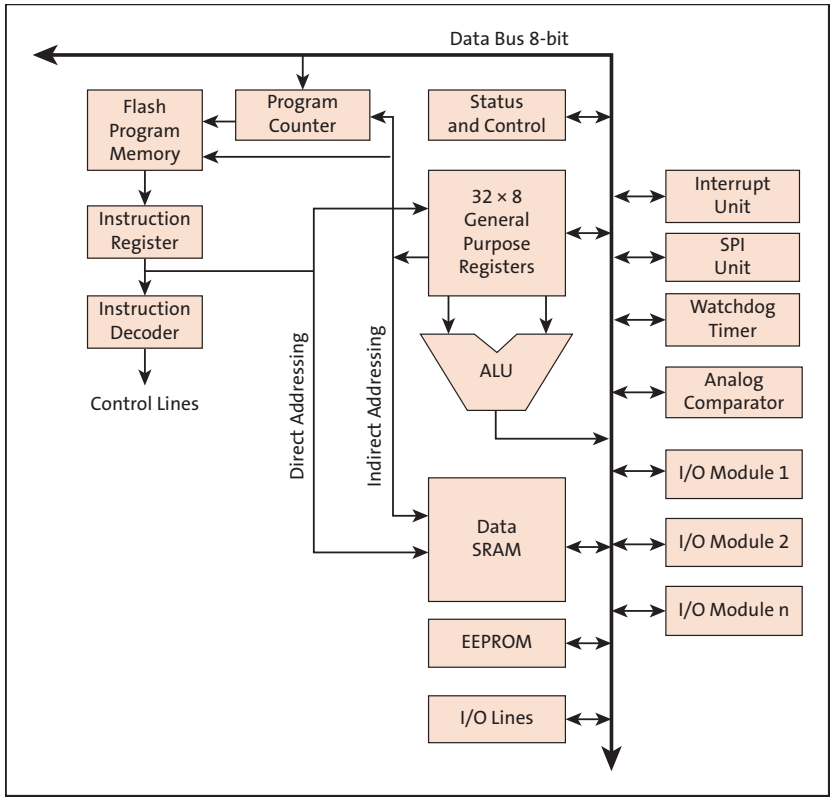

Abbildung 1.6 AVR-Architektur (Quelle: ATmega328P Data Sheet)

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

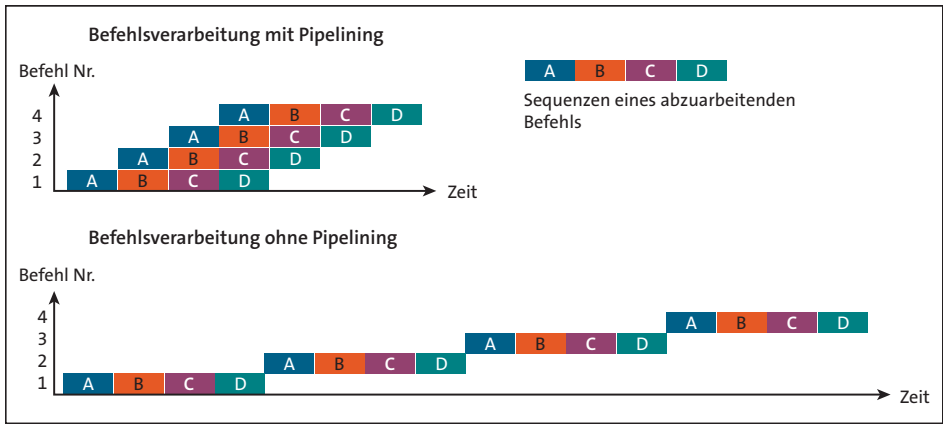

Abbildung 1.7 Das Prinzip des Pipelinings
(Quelle: https://de.wikipedia.org/wiki/Pipeline_(Prozessor))

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 habe ich die Adressierung der Register bereits gezeigt.

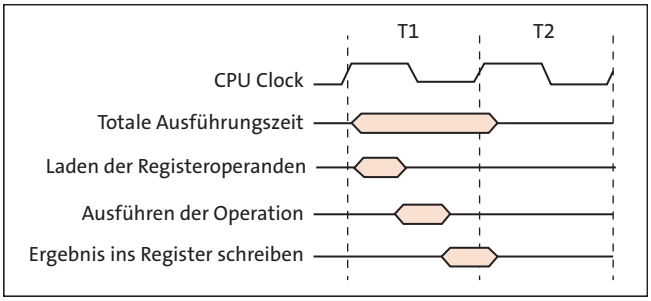

Abbildung 1.8 Single-Cycle-ALU-Operation

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 rührt 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 Die Mikrocontroller-Peripherie

Wie aus Abbildung 1.5 und Abbildung 1.6 ersichtlich ist, 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.

Das Interrupt-System

Der ATmega328P kann eine Vielzahl unterschiedlicher Interrupts verarbeiten. *Interrupts* sind Unterbrechungen des laufenden Programms und dienen zur 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, Watchdog System Reset
2	0x0002	INT0	External Interrupt 0
3	0x0004	INT1	External Interrupt 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 Interrupt
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. Der Begriff 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.
- **I^2C-Bus Interrupt** – ausgelöst durch verschiedene Zustände der I^2C-Kommunikation.
- **Store Program Memory Ready Interrupt** – ausgelöst durch das Ende eines Schreibvorgangs in den Programmspeicher.

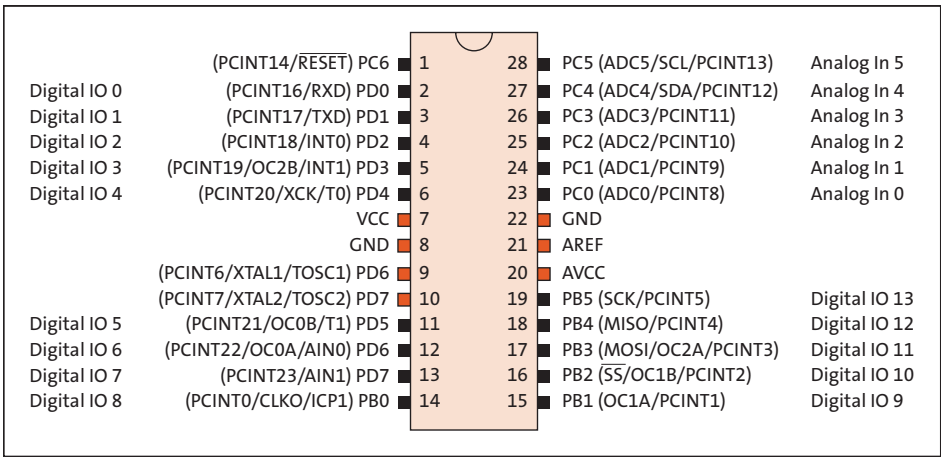

Abbildung 1.9 Anschlussbezeichnungen beim Arduino und ATmega328P

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 ausgeben 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.

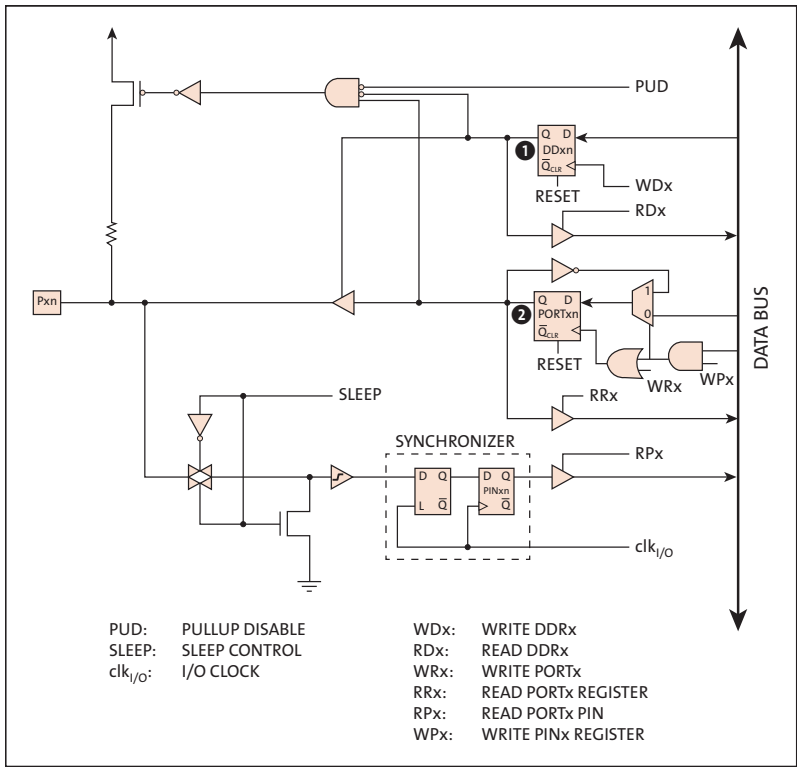

Abbildung 1.10 Digitaler I/O-Port (Quelle: ATmega328P Data Sheet)

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.

Tabelle 1.4 Logische Zustände am I/O-Port

DDxn	PORTxn	I/O	Pull-up	Kommentar
1	0	Ausgang	Nein	Ausgang Low
1	1	Ausgang	Nein	Ausgang High

Tabelle 1.4 Logische Zustände am I/O-Port (Forts.)

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
- Write Collision Flag Protection
- Wakeup vom Idle Mode
- Double Speed (CK/2) Master SPI Mode

Die beiden 8-Bit-Schieberegister im Master und im Slave können als gemeinsames 16-Bit-Schieberegister aufgefasst werden. Wenn Daten vom Master zum Slave geschoben werden, dann werden gleichzeitig Daten vom Slave zum Master transportiert. Abbildung 1.11 zeigt den Datenfluss beim seriellen Datentransfer.

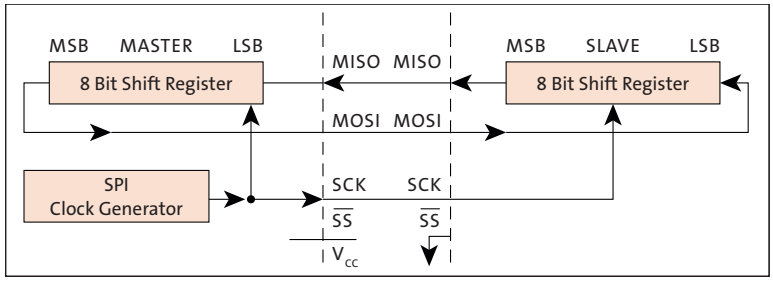

Abbildung 1.11 Datenfluss beim seriellen Datentransfer (Quelle: ATmega328P Data Sheet)

Um einen Datenaustausch über SPI vornehmen zu können, werden typischerweise vier Leitungen benötigt:

- MOSI – Master Out, Slave In
- MISO – Master In, Slave Out
- SCK – Serial Clock
- /SS – Slave Select

Um einen SPI-Baustein zu adressieren, müssen Sie dessen /SS-Pin auf Low setzen. Synchron zum Takt SCK werden die Bits vom Master zum Slave sowie gleichermaßen vom Slave zum Master verschoben. Clock-Polarität und Clock-Phase können sehr flexibel festgelegt werden. Ebenso kann unterschieden werden, ob das MSB oder das LSB zuerst verschoben wird. Durch diese Flexibilität ist es möglich, dass Sie das SPI an ganz unterschiedliche Anforderungen anpassen können.

Die Hardware-SPI ist beim ATmega328P verschiedenen Pins von Port B zugeordnet. Ein SPI kann aber auch vollständig in Software implementiert werden, wobei Sie dann die freie Wahl der verwendeten I/O-Pins haben.

Auf die Konfiguration und den Datenaustausch über SPI komme ich in Abschnitt 10.2 zurück. An dieser Stelle ist es wichtig, dass Sie den Datenaustausch über SPI als Schieberegister-Operation verstanden haben.

I^2C-Bus (TWI-Bus)

Der I^2C-Bus wurde von Philips entwickelt, um in einer vernetzten Umgebung Daten zwischen Mikrocontrollern und unterschiedlichen Bausteinen übertragen zu können, wie EEPROMs, RAMs, AD- und DA-Umsetzern, RTCs (*Real Time Clocks*, dt. Echtzeit-Uhrenbausteinen) und anderen mehr.

Das Protokoll erlaubt die Verbindung von bis zu 128 unterschiedlichen Bausteinen (*Devices*) mithilfe einer Zwei-Draht-Leitung. Die Adressierung der einzelnen Teilnehmer im Netzwerk erfolgt über das Protokoll. An externen Komponenten werden nur zwei Pull-up-Widerstände benötigt.

Abbildung 1.12 zeigt die erforderlichen Verbindungen in einem typischen I²C-Bus-Netzwerk. Die Leitungen SDA und SCL, über Pull-up-Widerstände an die Betriebsspannung VCC geführt, verbinden alle Mitglieder des Netzwerkes. In einem I²C-Bus-Netzwerk können verschiedene Master mit verschiedenen Slaves verbunden werden (Multi-Master-System).

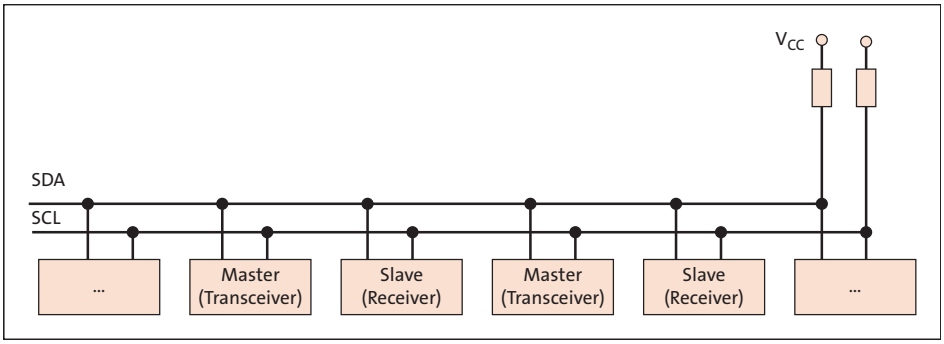

Abbildung 1.12 I²C-Bus-Netzwerk

Die zu realisierenden Peripheriefunktionen sind wiederum bausteinspezifisch. Neben den von zahlreichen Herstellern angebotenen EEPROMs und RAMs gibt es vielfältige weitere I²C-Bus-Bausteine. Eine sehr nützliche Übersicht zu I²C-Bus-Bausteinen finden Sie unter der URL *https://i2cdevices.org/devices*.

Die AVR-Mikrocontroller unterstützen den I²C-Bus durch eine von Atmel als *Two-Wire Interface* (*TWI*) bezeichnete Funktionsgruppe auf dem Chip.

Die Hauptmerkmale dieses Hardware-TWI sind:

- einfaches, leistungsfähiges und flexibles Kommunikationsinterface
- Master- oder Slave-Betrieb
- Betrieb als Sender oder Empfänger
- 128 verschiedene Slave-Adressen sind möglich.
- *Multi-Master Arbitration* wird unterstützt.
- bis zu 400 kHz Transfergeschwindigkeit
- Ausgangstreiber mit limitierter Slew-Rate
- Rauschunterdrückung; unterdrückt Spikes auf den Busleitungen.
- frei programmierbare Slave-Adressen
- Wakeup nach Adressdecodierung
- kompatibel zum Philips-I²C-Bus-Protokoll

Abbildung 1.13 zeigt ein vereinfachtes Blockdiagramm dieser komplexen Funktionsgruppe mit den zugehörigen Registern.

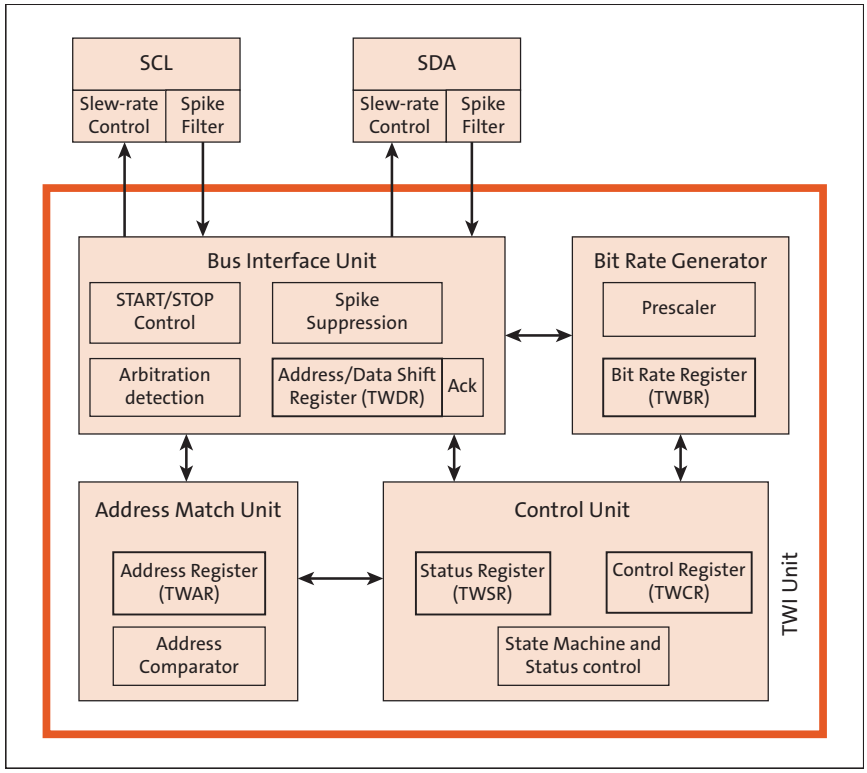

Abbildung 1.13 Blockdiagramm der »TWI Unit« (Quelle: ATmega328P Data Sheet)

Das TWI kann in vier unterschiedlichen Modi arbeiten:

- Master Transmitter Mode (MT)
- Master Receiver Mode (MR)
- Slave Transmitter Mode (ST)
- Slave Receiver Mode (SR)

Einige dieser Modi können in der gleichen Anwendung zum Einsatz kommen. Das in Abschnitt 9.2 beschriebene Beispiel zum Datenaustausch mit einem I^2C-Bus-EEPROM illustriert dies. Das TWI schreibt in diesem Programmbeispiel im Master Transmitter Mode Daten in das EEPROM, und im Master Receiver Mode liest es Daten vom EEPROM zurück.

USART

Der Abkürzung USART steht für die etwas sperrige Bezeichnung *Universal Synchronous and Asynchronous serial Receiver and Transmitter*. Die in den AVR-Mikrocontrollern implementierten USARTs sind zu den bekannten UARTs der Standard-Controller kompatibel, weisen aber auch weitergehende Funktionen auf.

Die Hauptmerkmale eines solchen USART sind:

- Full-Duplex-Betriebsweise
- synchroner oder asynchroner Betrieb
- Master- oder Slave-Synchronbetrieb
- interner hochauflösender Baudratengenerator
- hohe Baudraten bei niedrigen Taktfrequenzen
- 5 bis 9 Datenbits, 1 oder 2 Stoppbits
- ungerade oder gerade Parität, Paritätscheck durch Hardware
- Noise-Filter
- Overrun und Frame Error Detection
- Interrupts für das Ende der Übertragung bzw. das Empfangsende und Transmit-Register leer
- Multi-Prozessor-Modus
- Asynchronmodus mit doppelter Geschwindigkeit

Einen Eindruck von der Komplexität eines USART vermittelt ein vereinfachtes Blockdiagramm (siehe Abbildung 1.14).

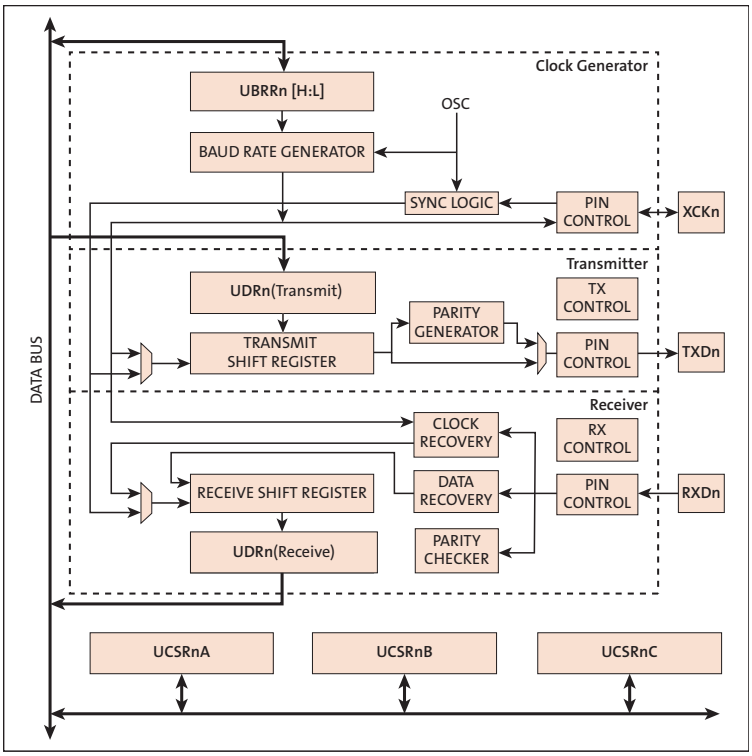

Abbildung 1.14 Vereinfachtes USART-Blockdiagramm (Quelle: ATmega328P Data Sheet)

Damit Sie verstehen können, wie die asynchrone serielle Kommunikation funktioniert, müssen Sie sich darüber im Klaren sein, wie Daten empfangen werden.

Alle Zeichen des seriellen Datenstroms, die nicht korrekt empfangen werden, gehen verloren – die Kommunikation ist somit gestört. Der Pin *RxD* wird mit der sechzehnfachen Baudrate abgetastet. Werden auf der Leitung Daten gesendet, dann geht diese von ihrem Ruhezustand (Hi) nach Lo und übermittelt dann ein Pulspaket entsprechend der Zahl der gesendeten Bits. Abbildung 1.15 zeigt die Abtastung des seriellen Datenstroms.

Durch den Hi-Lo-Übergang zu Beginn der Pulsfolge wird der Beginn des Startbits signalisiert, dem dann die Datenbits und ein oder zwei Stoppbits folgen. Wie in Abbildung 1.15 angedeutet, weisen die Abtastwerte 8, 9 und 10, die etwa in der Mitte jedes Bits liegen, eine Besonderheit auf. Mindestens zwei dieser Abtastwerte müssen den gleichen Wert aufweisen, um den Abtastwert des betreffenden Bits festzulegen. Diese Art der Rauschunterdrückung erhöht die Sicherheit der seriellen Kommunikation.

Wurde schließlich das Stoppbit erkannt, dann ist der Empfang des gesendeten Zeichens beendet und es kann in das Register *UDR (Receive)* gespeichert werden. Dort steht es dann zum Abholen durch das Anwenderprogramm bereit.

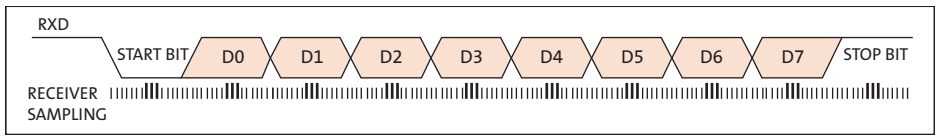

Abbildung 1.15 Abtastung des seriellen Datenstroms

Das Versenden von Daten wird durch Laden des Registers *UDR (Transmit)* mit den zu sendenden Daten eingeleitet.

Im Synchronmodus dient der Pin *XCK* für einen Slave als Takteingang und für einen Master als Taktausgang.

Die vielfältigen Features des USART werden Sie bei der Verwendung des ATmega328P im Arduino nicht ausnutzen. Sie werden die UART-Funktion zur asynchronen seriellen Datenübertragung nutzen und im Wesentlichen mit verschiedenen Baudraten arbeiten.

Der hier betrachtete ATmega328P weist ein USART auf. Andere Mikrocontroller können durchaus mehrere Hardware-U(S)ARTs aufweisen. Es besteht aber auch die Möglichkeit, ein USART in Software zu implementieren. Von dieser Möglichkeit werden wir beim Arduino Uno noch Gebrauch machen.

Timer/Counter

Die AVR-Mikrocontroller weisen 8-Bit- und 16-Bit-Timer/Counter auf, die unabhängig als Timer mit einem internen Takt oder als Counter mit externem Trigger arbeiten. Der *Prescaler* versorgt die Timer mit einem Taktsignal, das vom internen Takt abgeleitet wird. Das Vorteilerverhältnis wird über das Register *TCCRx* selektiert. Abbildung 1.16 zeigt ein Blockdiagramm eines Prescalers für die Timer/Counter0 und 1.

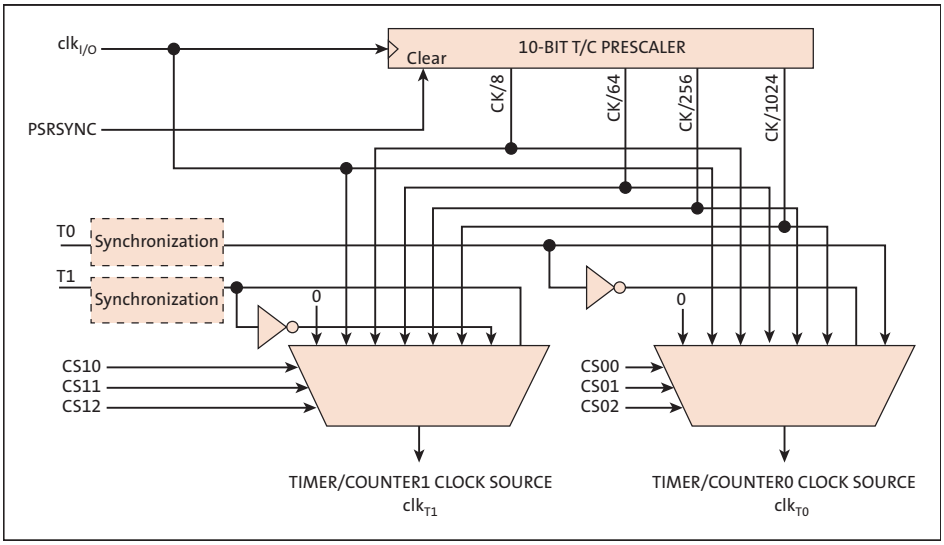

Abbildung 1.16 Timer/Counter0/1-Prescaler (Quelle: ATmega328P Data Sheet)

Der 8-Bit-*Timer/Counter0* des ATmega328P ist als einfacher Up-Counter mit Schreib-/Lesezugriff aufgebaut. *Timer/Counter0* weist die folgenden Eigenschaften auf:

- zwei unabhängige Output Compare Units
- doppelt gepufferte Output Compare Register
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- variable PWM-Periode
- Frequenz-Generator
- drei unabhängige Interrupt-Quellen (TOV0, OCF0A, OCF0B)

Abbildung 1.17 zeigt das Blockdiagramm von Timer/Counter0.

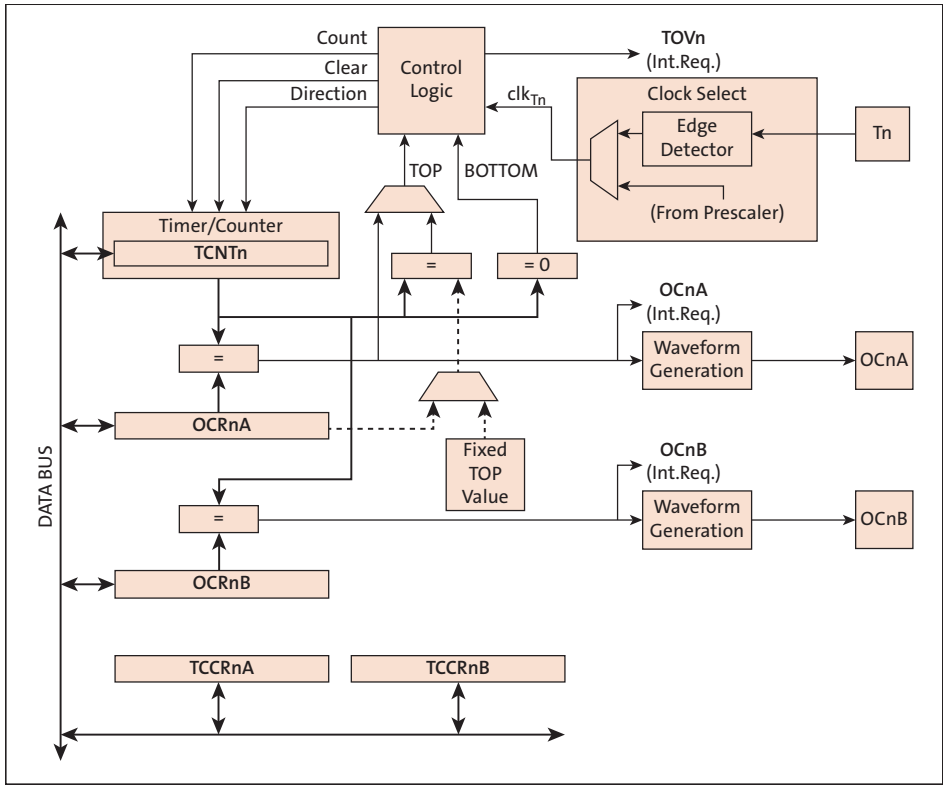

Abbildung 1.17 Blockdiagramm von Timer/Counter0 (Quelle: ATmega328P Data Sheet)

Die Konfiguration von Timer/Counter0 erfolgt über die beiden Register *TCCR0A* und *TCCR0B*.

Beim Overflow des Registers *TCNT0* (`0xFF` nach `0x00`) kann ein Interrupt angefordert werden.

Ein 8-Bit-Comparator vergleicht kontinuierlich das Register TCNT0 mit den Output-Compare-Registern *OCR0A* und *OCR0B*. Ist der Inhalt von Register TCNT0 gleich dem von OCR0A oder OCR0B, dann signalisiert das der Comparator (Match). Ein Match setzt dann das *Output Compare Flag* (*OCF0A* oder *OCF0B*). Ist der betreffende Interrupt freigegeben, dann erzeugt das gesetzte Output Compare Flag außerdem einen *Output Compare Interrupt*.

Der 16-Bit-*Timer/Counter1* ist komplexer als der 8-Bit-Timer/Counter0 und unterstützt Output-Compare- und Input-Capture-Funktionen. Timer/Counter1 weist die folgenden Eigenschaften auf:

- echtes 16-Bit-Design (erlaubt 16-Bit-PWM)
- zwei unabhängige Output Compare Units
- doppelt gepufferte Output Compare Register
- eine Input Capture Unit
- Input-Capture-Noise-Unterdrückung
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- variable PWM-Periode
- Frequenz-Generator
- externer Event-Counter
- vier unabhängige Interrupt-Quellen (TOV1, OCF1A, OCF1B, ICF1)

Abbildung 1.18 zeigt zur Verdeutlichung der wesentlich höheren Komplexität das Blockdiagramm von Timer/Counter1.

Das Verhalten bei *Overflow* und *Output Compare* ist bis auf die Breite der Register von 16 Bit praktisch dasselbe wie beim Timer/Counter0.

Die Input-Capture-Funktion wird durch ein externes Ereignis getriggert, in dessen Folge der Inhalt des Registers *TCNT1* in das Input-Capture-Register *ICR1* geschrieben wird. Die Input-Capture-Funktion kann auch vom Analogkomparator getriggert werden.

Des Weiteren kann dieser Timer/Counter als PWM genutzt werden. Interrupts werden vom 16-Bit-Timer/Counter beim Overflow, beim Output Compare sowie beim Input Capture angefordert.

Der 8-Bit-*Timer/Counter2* des ATmega328P ist ein einkanaliger, bidirektionaler 8-Bit-Timer/Counter. Timer/Counter2 weist die folgenden Eigenschaften auf:

- Single Channel Counter
- Clear Timer on Compare Match (Auto Reload)
- Glitch-free, Phase Correct Pulse Width Modulator (PWM)
- Frequenz-Generator
- 10-Bit-Clock-Prescaler
- Overflow und Compare Match Interrupt Sources (TOV2, OCF2A, OCF2B)
- Erlaubt externen Takt von einem 32-kHz-Uhrenquarz unabhängig vom Takt der I/O-Clock.

Auf den ersten Blick ähneln sich Timer/Counter0 und Timer/Counter2. Im Detail betrachtet, bietet Timer/Counter2 zusätzliche Features und dadurch mehr Flexibilität.

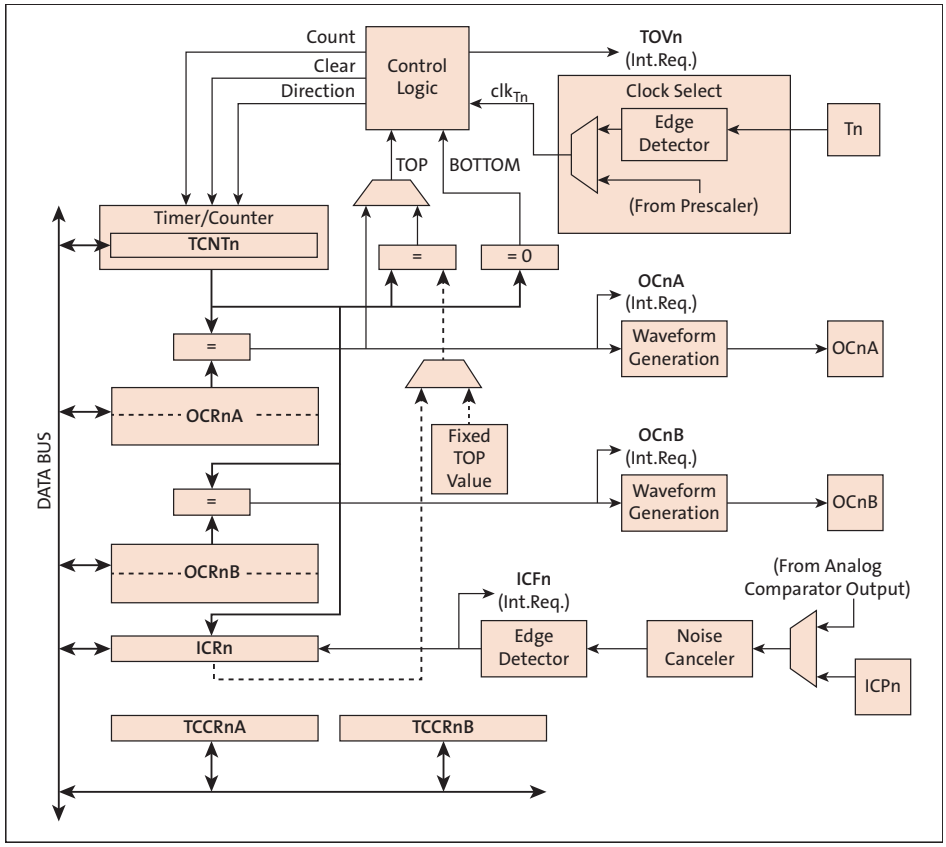

Abbildung 1.18 Blockdiagramm von Timer/Counter1 (Quelle: ATmega328P Data Sheet)

Watchdog

Watchdogs dienen in von Mikrocontrollern gesteuerten elektronischen Geräten dazu, einem Totalausfall durch Softwareversagen zuvorzukommen.

Der Watchdog muss innerhalb einer vorgegebenen Zeit von der Software zurückgesetzt werden, anderenfalls erfolgt nach Ablauf der vorgegebenen Zeit ein Reset des Anwendungsprogramms. Konnte das laufende Programm nicht rechtzeitig den Watchdog bedienen, muss davon ausgegangen werden, dass das Programm »hängen geblieben« oder ausgefallen ist. Der Watchdog setzt daraufhin durch einen Reset das Gerät in einen definierten Ausgangszustand zurück, damit das von der Software gesteuerte System wieder überwacht arbeiten kann.

Der Watchdog Timer im ATmega328P wird durch einen separaten On-Chip-Oszillator mit nominal 1 MHz getaktet. Es lassen sich Watchdog-Intervalle zwischen 16 ms und 1,9 s (VCC = 5 V) einstellen. Abbildung 1.19 zeigt das Blockdiagramm des Watchdogs.

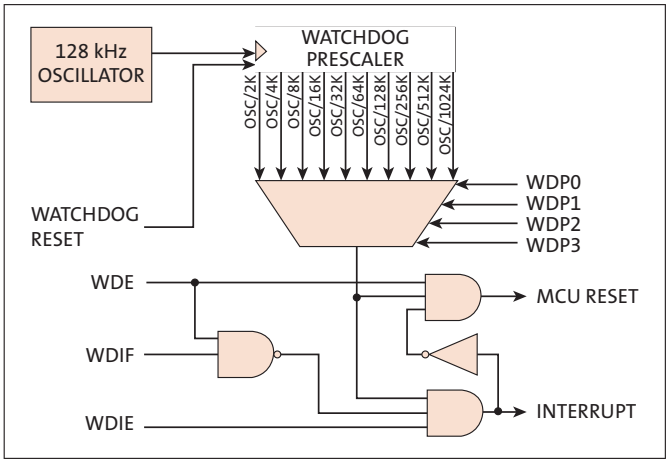

Abbildung 1.19 Blockdiagramm des Watchdogs (Quelle: ATmega328P Data Sheet)

Beim Festlegen der Watchdog-Periode muss berücksichtigt werden, dass die Frequenz des internen Oszillators sowohl stark betriebsspannungs- als auch temperaturabhängig ist.

Analog/Digital-Umsetzer

Der ATmega328P weist einen internen Analog/Digital-Umsetzer (ADU, auch ADC für *Analog/Digital Converter*) mit folgenden Merkmalen auf:

- Auflösung 10 Bit
- integrale Nichtlinearität 0,5 LSB
- absolute Genauigkeit ± 2 LSB
- Umsetzzeit von 65 bis 260 µs
- Umsatzrate bis zu 76,9 ksps (sps = *sample per second*)
- Umsatzrate bis zu 15 ksps bei maximaler Auflösung
- sechs massebezogene Analogeingänge (*single ended*)
- Ergebnis der AD-Umsetzung rechts- oder linksbündig
- Eingangsspannungsbereich 0 bis AVCC
- interne Referenzspannung von 2,56 V
- kontinuierliche (*free running*) oder Einzelumsetzung (*single shot*)
- interruptgesteuerte Auslösung der AD-Umsetzung durch Auto-Triggerung
- Interrupt am Ende der AD-Umsetzung
- Rausch-Unterdrückung

Abbildung 1.20 zeigt das Blockdiagramm des AD-Umsetzers eines ATmega328P.

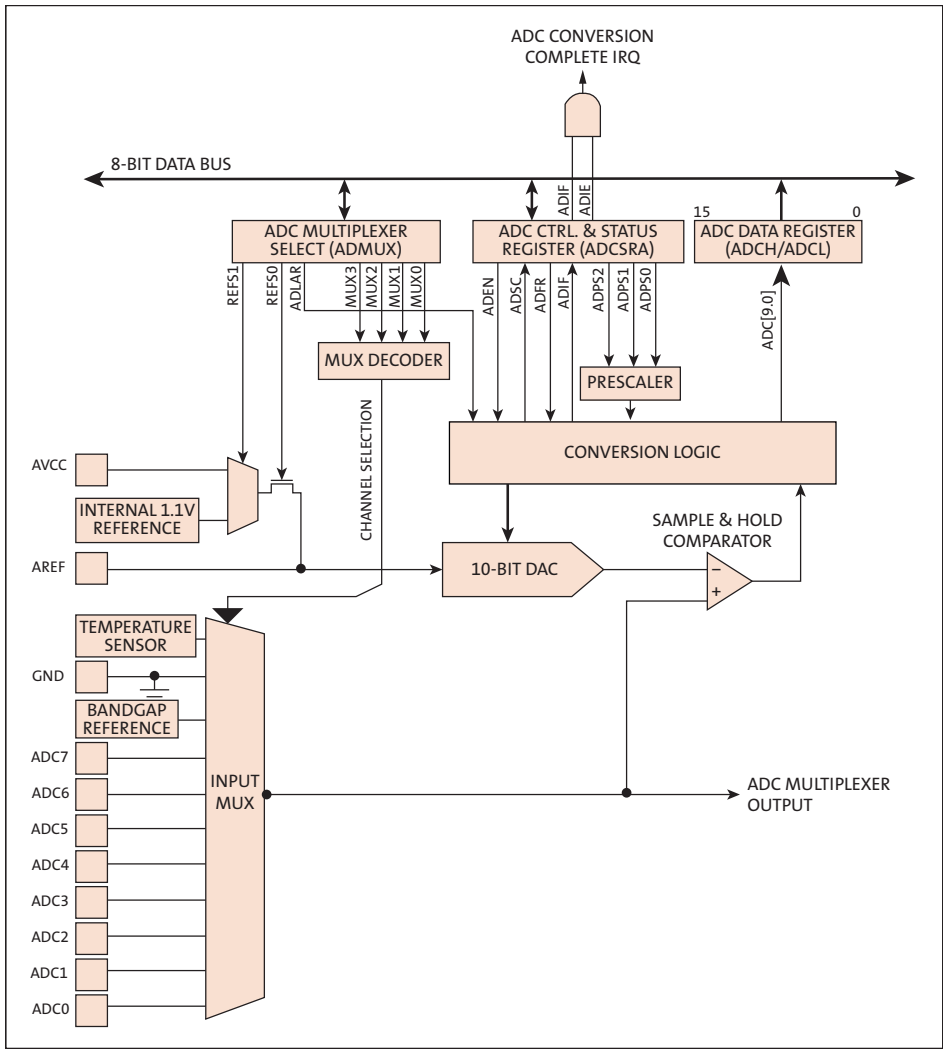

Abbildung 1.20 Blockdiagramm des AD-Umsetzers (Quelle: ATmega328P Data Sheet)

Bei einem AD-Umsetzer nach dem Verfahren der *sukzessiven Approximation* wird die über einen Analogmultiplexer zugeführte analoge Eingangsspannung mit der Ausgangsspannung eines DA-Umsetzers verglichen. Die Ausgangsspannung des 10-Bit-DA-Umsetzers wird durch die Steuerlogik (Register ADMUX, ADCSRA) sowie durch eine der Referenzspannungen festgelegt. Die Steuerlogik steuert den DA-Umsetzer bitweise an, und das Komparator-Ausgangssignal bestimmt, ob das jeweilige Bit im Ausgaberegister (ADCH, ADCL) gesetzt oder nicht gesetzt wird.

Auf diese Weise nähert sich die Ausgangsspannung des DA-Umsetzers sukzessive dem zu erfassenden analogen Spannungswert. Daher kommt auch der Name des Verfahrens: sukzessive Approximation.

Der zu erfassende Spannungswert darf sich während des Umsetzvorgangs nicht ändern, da sonst ein falscher Inhalt des Ausgaberegisters die Folge wäre. Eine dem Komparator vorgeschaltete Sample-and-Hold-Schaltung, die den abgetasteten Spannungswert für die Zeit der AD-Umsetzung zwischenspeichert, erfüllt hier diese Forderung.

Die erforderliche Umsetzungszeit ist unabhängig von der anliegenden Eingangsspannung und richtet sich nur nach der Auflösung des AD-Umsetzers. Ein 10-Bit-AD-Umsetzer benötigt genau zehn Umsetzschritte, deren Zeit durch die Taktung des DA-Umsetzers und die Schaltzeit des Komparators bestimmt wird.

Die Eingangsspannung wird über einen Analog-Multiplexer an den Komparator geführt. Für Kalibrations- und Testzwecke können zusätzlich die interne Band-Gap-Referenzspannung, das Massepotenzial und das Ausgangssignal eines internen Temperatursensors an den Komparator geführt werden.

Wichtig für die eingangsseitige Beschaltung des AD-Umsetzers ist die analoge Eingangsschaltung. Abbildung 1.21 zeigt ein Ersatzschaltbild für die massebezogene Messung.

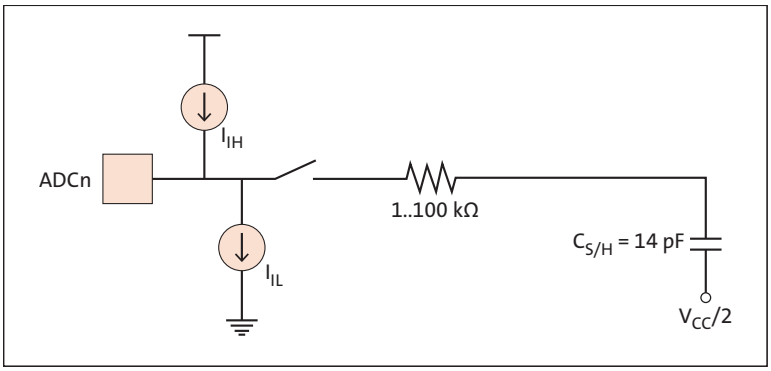

Abbildung 1.21 Analoge Eingangsschaltung (Quelle: ATmega328P Data Sheet)

Die analoge Spannungsquelle (Eingangsspannung) wird in jedem Fall durch die Eingangsleckströme und eine Eingangskapazität des Anschlusses (hier nicht dargestellt) belastet. Ist der betreffende Multiplexer-Kanal durchgeschaltet, dann muss die Spannungsquelle das RC-Glied aus dem Serienwiderstand und der Sample&Hold-Kapazität aufladen.

Die Eingangsschaltung ist für Impedanzen von 10 kΩ und weniger optimiert. Liegen die Impedanzen höher, dann kann eine Entkopplung durch einen Operationsverstärker angezeigt sein.

Das Resultat der AD-Umsetzung folgt idealerweise für die massebezogene Messung (*single ended*) der Beziehung

$$\mathrm{ADC} = \frac{V_{\mathrm{IN}}}{V_{\mathrm{REF}}} \times 1024$$

Eine fehlerfreie AD-Umsetzung ist kaum zu erwarten, weshalb Sie den Abweichungen vom Idealverhalten Beachtung schenken sollten.

Auf diese Weise verlangen Sie von der ausgewählten Baugruppe nur so viel, wie diese auch zu leisten vermag, und können die Fehler in bestimmtem Maße eliminieren. Abbildung 1.22 zeigt die grundsätzlichen Fehler bei der AD-Umsetzung.

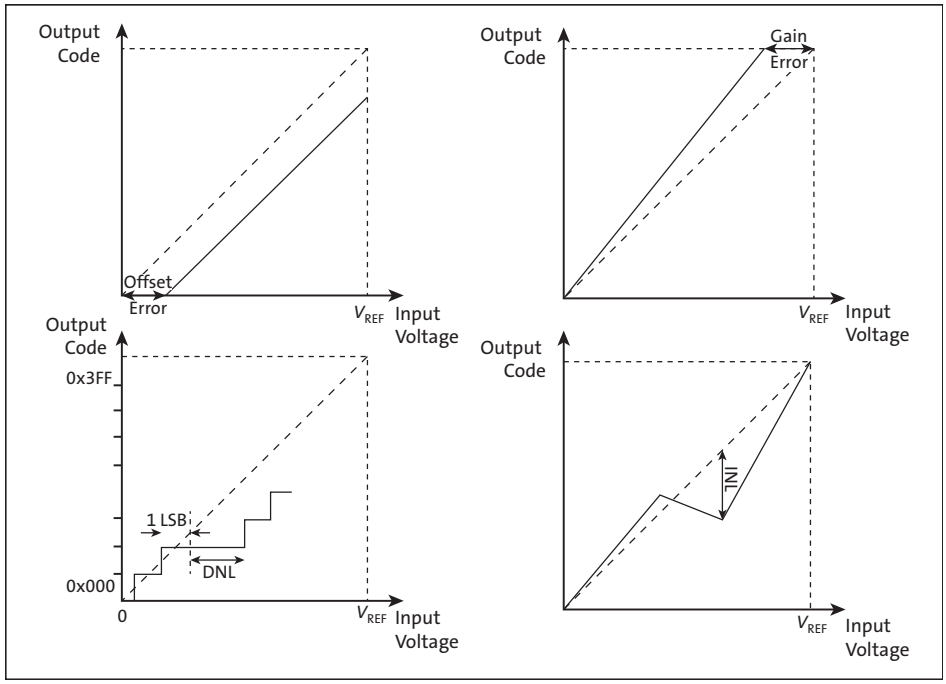

Abbildung 1.22 Fehler bei der AD-Umsetzung (Quelle: ATmega328P Data Sheet)

Die Abbildung links oben zeigt einen Offset-Fehler. Ein *Offset-Fehler (Offset Error)* liegt dann vor, wenn der Übergang des Resultats der AD-Umsetzung von 0 auf 1 nicht bei einer Eingangsspannung von ½ LSB (hier $V_{\mathrm{REF}}/2048$) erfolgt. Bei einer Referenzspannung V_{REF} = AVCC = 5 V beträgt das LSB 4,88 mV.

Die Abbildung rechts oben zeigt einen *Verstärkungsfehler* (*Gain Error*). Ein Verstärkungsfehler liegt dann vor, wenn der Übergang des Resultats der AD-Umsetzung von `0x3FE` auf `0x3FF` nicht bei einer Eingangsspannung von 1½ LSB unterhalb der Referenzspannung erfolgt.

Die Abbildung links unten zeigt die *differenzielle Nichtlinearität* (DNL). Die DNL kennzeichnet den maximalen Spannungshub pro Quantisierungsschritt. Ideal ist ein Spannungshub für jede Stufe von genau 1 LSB.

Die Abbildung rechts unten zeigt die *integrale Nichtlinearität* (INL). Die INL ist die maximale Abweichung von der Idealkennlinie des AD-Umsetzers.

Neben der statischen Kennlinie des AD-Umsetzers ist auch das Rauschverhalten von Interesse. Hierbei kann aber der AD-Umsetzer nicht isoliert betrachtet werden, sondern die analoge Eingangsschaltung (Leitungslängen etc.) muss Berücksichtigung finden.

Sicher werden Sie sofort ahnen, dass die hier angegebenen Fehler nicht isoliert auftreten, sondern sich überlagern. Die Umsetzung eines analogen Sensorsignals kann so schon zur Herausforderung werden.

Interne AD-Umsetzer in Mikrocontrollern haben im Allgemeinen weniger gute Eigenschaften als spezielle AD-Umsetzer, die eigenständige Bausteine sind. Das wird bereits anhand der Auflösung deutlich. Wenn also höhere Genauigkeiten und geringere Fehler zwingend sind, dann kann ein Mikrocontroller auch durch einen externen AD-Umsetzer ergänzt werden.

Analogkomparator

Ein Analogkomparator vergleicht Spannungen an zwei I/O-Pins. Der Ausgang des Analogkomparators wird bei positivem Vergleichsergebnis gesetzt. Dieses Ereignis kann zur Anforderung eines Interrupts oder zum Triggern der Input-Capture-Funktion des 16-Bit-Timer/Counters dienen.

Abbildung 1.23 zeigt das Blockschema des Analogkomparators.

Die Konfiguration des Analogkomparators erfolgt über das Register ACSR. Der Analogkomparator kann ausgeschaltet werden, wodurch die Stromaufnahme reduziert wird. Per Default ist der Analogkomparator aber eingeschaltet.

Der Komparatorausgang kann einen Interrupt auslösen. Über das Register ACSR wird das Trigger-Ereignis (fallende Flanke, steigende Flanke, Wechsel an ACI (Toggle)) selektiert. Die Auswertung des Komparatorausgangs kann dann über einen Interrupt oder ein Polling des Registers ACSR Bit ACO erfolgen.

Es ist außerdem möglich, einen der ADC-Pins auszuwählen, um den negativen Eingang des Analogkomparators AIN1 zu ersetzen. Der ADC-Multiplexer wird verwendet, um diesen Eingang auszuwählen, und folglich muss der AD-Umsetzer ausgeschaltet werden, um diese Funktion nutzen zu können.

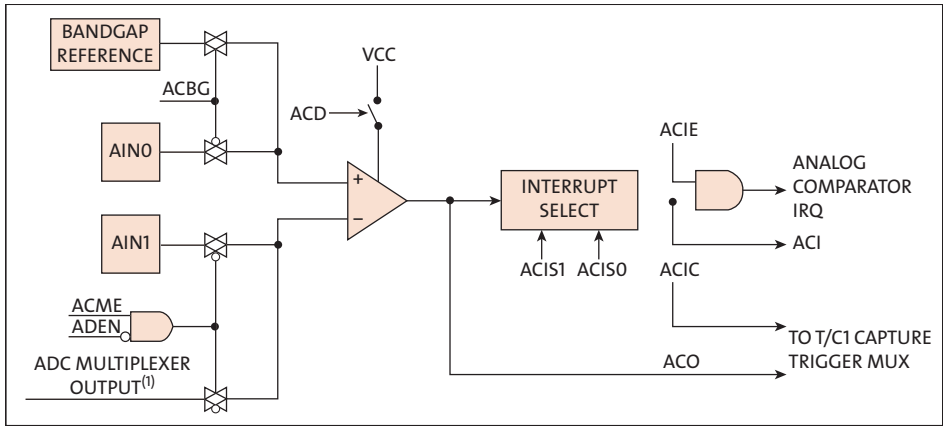

Abbildung 1.23 Blockdiagramm des Analogkomparators (Quelle: ATmega328P Data Sheet)

Interne Referenzspannung

Die Referenzspannung für den AD-Umsetzer (V_{REF}) gibt den Umwandlungsbereich für die AD-Umsetzung an. Eingangssignale, die den Wert V_{REF} überschreiten, führen zu Codes in der Nähe von 0x3FF.

Die Referenzspannung V_{REF} kann entweder als AVCC, als interne 1,1-V-Referenz oder als externer AREF-Pin ausgewählt werden.

AVCC ist über einen passiven Schalter mit dem ADC verbunden. Die interne 1,1-V-Referenz wird aus der internen *Band-Gap-Referenz* (VBG) über einen internen Verstärker erzeugt. In beiden Fällen ist der externe AREF-Pin direkt mit dem AD-Umsetzer verbunden. Die Referenzspannung wird unempfindlicher gegen Rauschen, wenn ein Kondensator zwischen den AREF-Pin und der Masse geschaltet wird.

Die Spannung V_{REF} kann auch mit einem hochohmigen Voltmeter am AREF-Pin gemessen werden. Beachten Sie, dass V_{REF} eine hochohmige Quelle ist und dass in einem System nur eine kapazitive Last angeschlossen werden sollte.

Wenn Sie eine feste Spannungsquelle an den AREF-Pin anschließen, können Sie die anderen Referenzspannungsoptionen in der Anwendung nicht verwenden, da diese mit der externen Spannung kurzgeschlossen werden.

Wenn keine externe Spannung an den AREF-Pin angelegt wird, können Sie zwischen AVCC und der Bad-Gap-Referenz (1,1 V) als Referenzauswahl umschalten.

Das erste Ergebnis einer AD-Umsetzung nach dem Umschalten der Referenzspannungsquelle ist möglicherweise ungenau und sollte stets verworfen werden.

EEPROM

Der ATmega328P enthält 1 KB EEPROM-Speicher. EEPROM steht für *Electrically Erasable Programmable Read-Only Memory* (dt. elektrisch löschbarer programmierbarer Nur-Lese-Speicher) und bezeichnet einen nichtflüchtigen Speicher, dessen Inhalt elektrisch gelöscht werden kann. Dieser Speicher ist als separater Speicherbereich organisiert, in den einzelne Bytes gelesen und geschrieben werden können.

Das EEPROM hat eine Lebensdauer von mindestens 100.000 Schreib-/Löschzyklen und sollte deshalb nicht für Inhalte eingesetzt werden, die sich rasch ändern. Konfigurationsdaten, Seriennummern, Kalibrierdaten und Ähnliches sind die bevorzugten Inhalte eines solchen Speichers.

Der Zugriff der CPU auf ein EEPROM erfolgt unter Angabe der EEPROM-Adressregister (EEARH, EEARL), des EEPROM-Datenregisters (EEDR) und des EEPROM-Steuerregisters (EECR).

In Abschnitt 9.1 werde ich den Zugriff auf das EEPROM des Arduino Uno erläutern, und Sie werden sehen, dass sich die Zugriffe sehr einfach gestalten, wenn Sie eine geeignete Library nutzen. Die Zugriffszeiten werden dadurch natürlich nicht reduziert.

Kapitel 3
Das Experimentierumfeld

In den beiden vorangegangenen Kapiteln habe ich Ihnen Grundlagen zu Mikrocontrollern und besonders der Arduino-Familie vorgestellt. Bevor wir zum Software-Teil kommen, 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 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.

Abbildung 3.1 zeigt das Schaltbild für den Widerstand. 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.

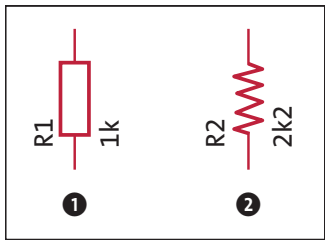

Abbildung 3.1 Schaltbild für den Widerstand

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 Downloadmaterial und Abschnitt A.4.2), 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 mindestens 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 in Abschnitt A.3 zusammengestellt, das Sie bei den Download-Materialien und im Anhang finden. 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.

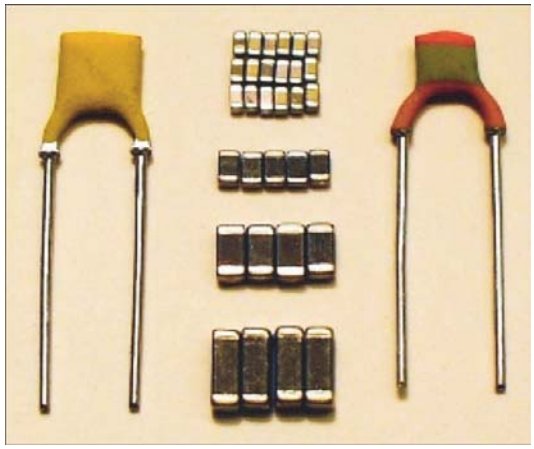

Abbildung 3.4 Keramikkondensatoren (Quelle: Elcap [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/)])

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. Solche Kondensatoren 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.

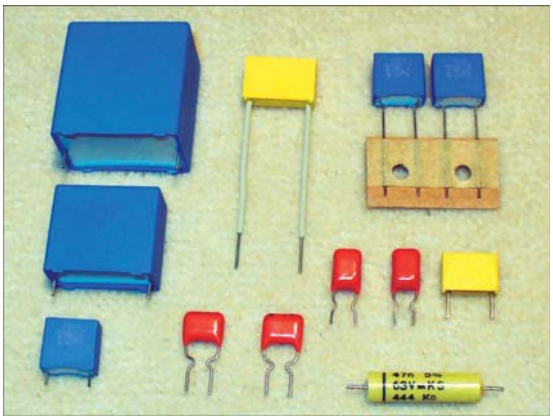

Abbildung 3.5 Folienkondensatoren (Quelle: Elcap [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)])

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* eine detaillierte Beschreibung zahlreicher Aspekte des Elektrolytkondensators. Abbildung 3.6 zeigt Elektrolytkondensatoren unterschiedlicher Bauformen.

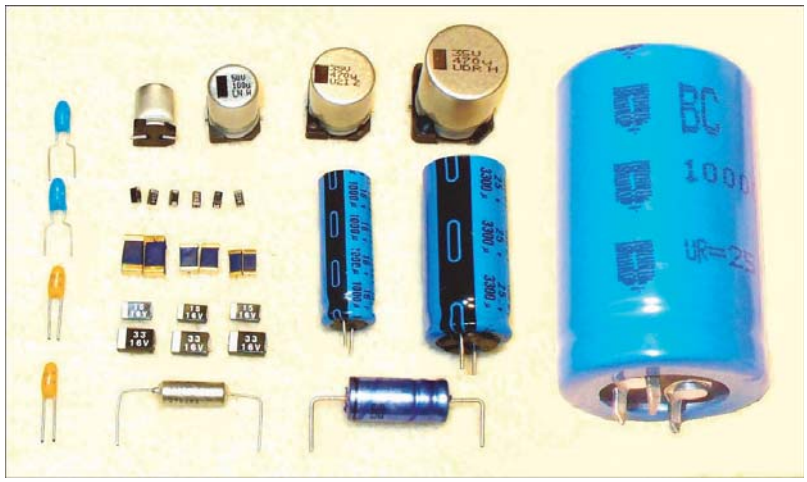

Abbildung 3.6 Elektrolyt- und Tantal-Kondensatoren (Quelle: Elcap [CC0 (https://creativecommons.org/publicdomain/zero/1.0/deed.de)])

Abbildung 3.7 zeigt das Schaltbild für den Kondensator. 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.

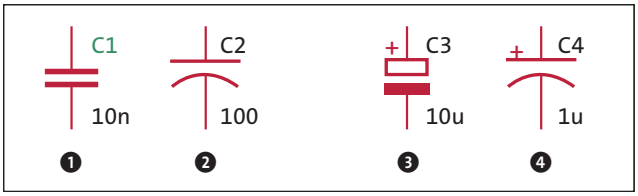

Abbildung 3.7 Schaltbild für Kondensatoren

Spule

Spulen können Teile eines Gerätes sein, z. B. eines Relais, eines Elektromagneten oder Elektromotors, eines Transformators oder eines Lautsprechers. 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, eine Stromänderung an einer Spule erzeugt 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 ❶, Schalter ❷, DIP-Schalter ❸, Drehschalter ❹ und ein Relais mit einem Umschaltkontakt ❺. Taster und Schalter werden durch die Kennzeichnung der mechanischen Betätigung an der Schaltzunge unterschieden.

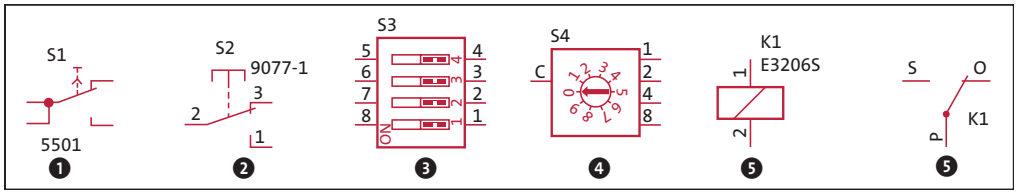

Abbildung 3.9 Taster, Schalter und Relais – Schaltbilder

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.

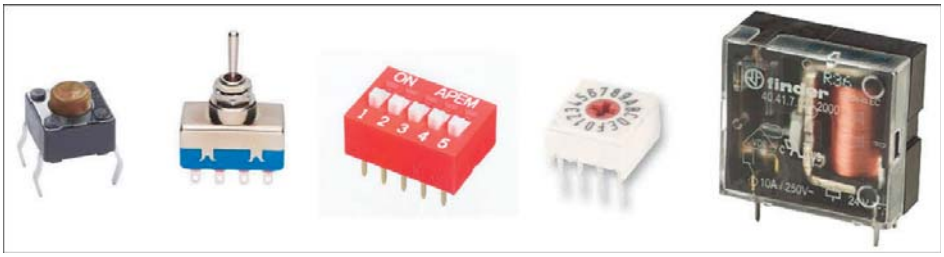

Abbildung 3.10 Taster, Schalter und Relais – Beispiele für Bauformen

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.

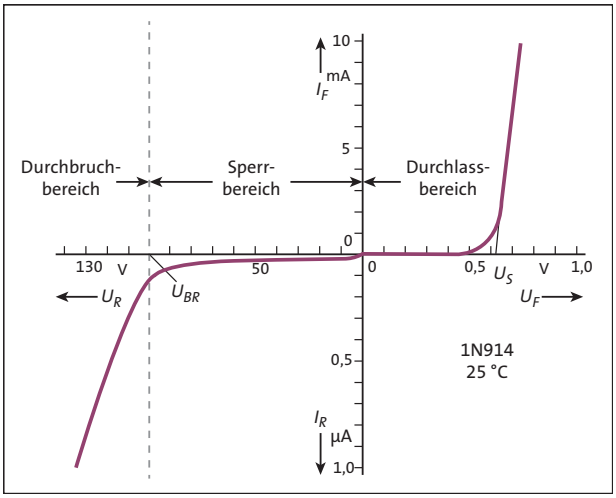

Abbildung 3.11 Dioden-Kennlinie

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:

- Dioden-Flussspannung U_D
- Strom durch die Diode in Flussrichtung I_D
- (Sättigungs-)Sperrstrom $I_S = 10^{-12} \ldots 10^{-6}$ A
- Emmisionskoeffizient $n = 1 \ldots 2$
- Temperaturspannung $U_T = kT/e$ (ca. 25 mV bei Raumtemperatur)
- Boltzmann-Konstante $k = 1{,}380649 \times 10^{-23}$ J/K
- Elementarladung $e = 1{,}602 \times 10^{-19}$ 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 ihre 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 mar-

kiert, die einfallende Strahlung symbolisieren. Umgekehrt ist das bei der LED ❹. Alle diese Bauteile werden Sie in den Anwendungen wiederfinden.

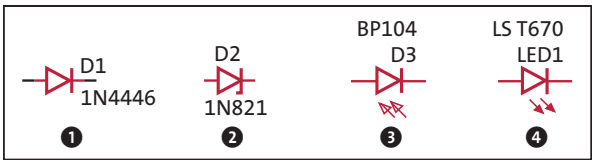

Abbildung 3.12 Dioden – Schaltbilder

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 Sie in unseren Anwendungen immer wieder vorfinden werden.

3.1.4 Transistoren und FETs als Schalter

Bei Bipolar- und Feldeffekt-Transistoren (*FETs*) 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.

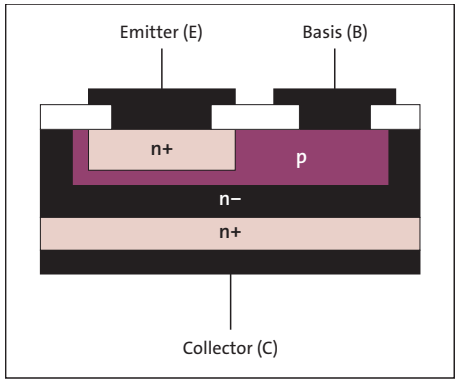

Abbildung 3.14 Aufbau des npn-Transistors

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.

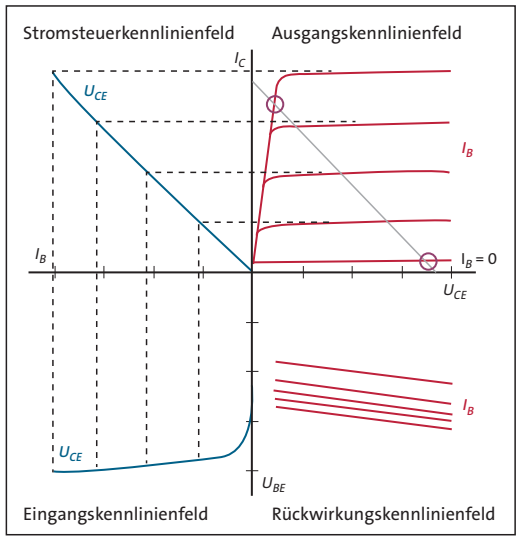

Abbildung 3.15 Transistor-Kennlinienfeld

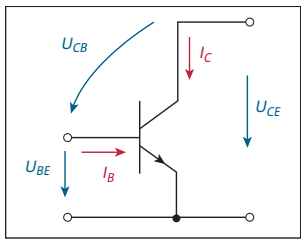

Abbildung 3.16 Spannungen am npn-Transistor

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).

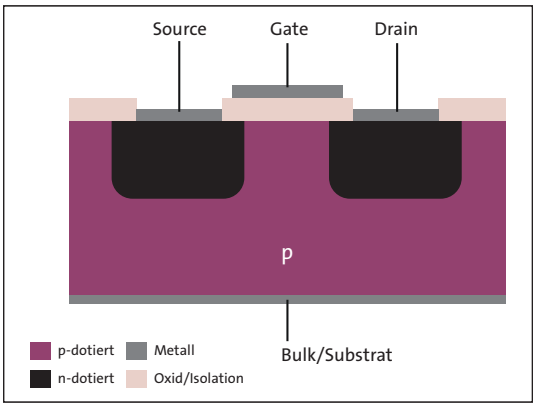

Abbildung 3.17 Aufbau eines n-Kanal-MOSFETs

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 in Tabelle 3.1 helfen, das zu verdeutlichen:

Typ	Bauteil	R_{DSon}	U_{DS} bei 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

Durch ihren Aufbau bedingt sind die Widerstände im eingeschalteten Zustand bei p-Kanal-FETs höher als bei n-Kanal-FETs.

Kommen wir zum praktischen Teil und betrachten wir 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 Type) habe ich hier nicht betrachtet. Der Pfeil am Gate kennzeichnet die Dotierung des Kanals (n-Kanal ❸, p-Kanal ❹).

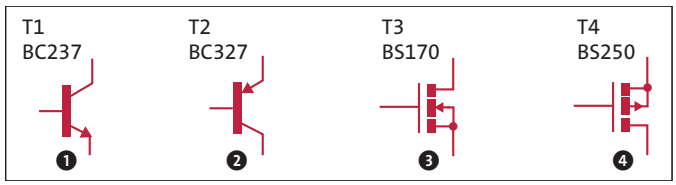

Abbildung 3.18 Schaltbilder für Transistor und FET

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.

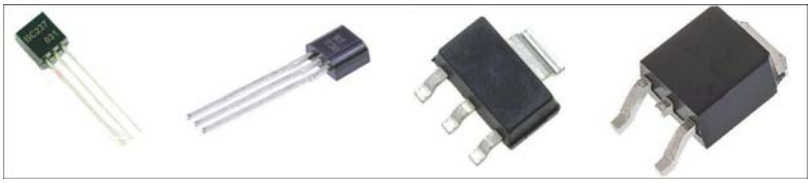

Abbildung 3.19 Bauformen für Bipolartransistoren und FETs

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.

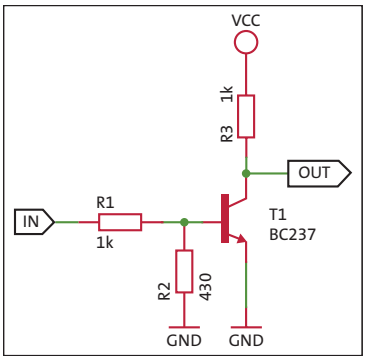

Abbildung 3.20 npn-Transistor als Schalter

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).

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.

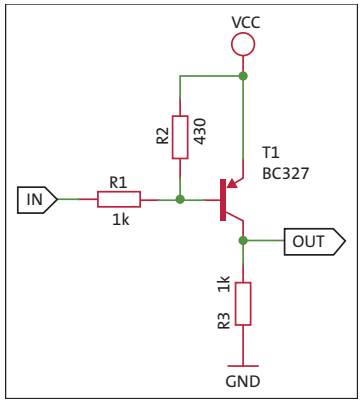

Abbildung 3.21 pnp-Transistor als Schalter

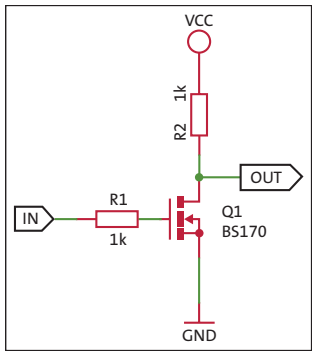

Abbildung 3.22 n-Kanal-FET als Schalter

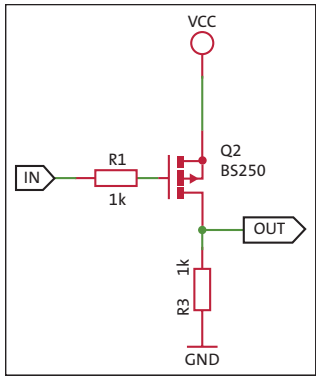

Abbildung 3.23 p-Kanal-FET als Schalter

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).

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 sehr klein bzw. geht gegen null. (Die Eingangsspannung ist die Differenz der Spannungen an den beiden Eingängen + und –.)

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.

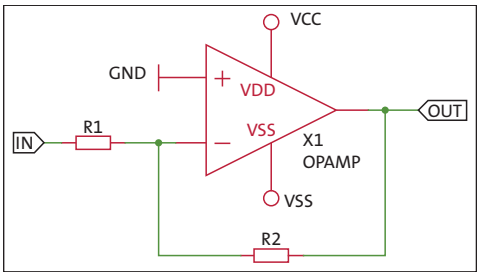

Abbildung 3.24 Invertierender Operationsverstärker

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

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

Wenn wir die oben angegebenen Bedingungen für einen idealen Operationsverstärker annehmen, bestimmen die Widerstände R1 und R2 das Verhalten der Gesamtschaltung. Die Spannungsverstärkung 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.

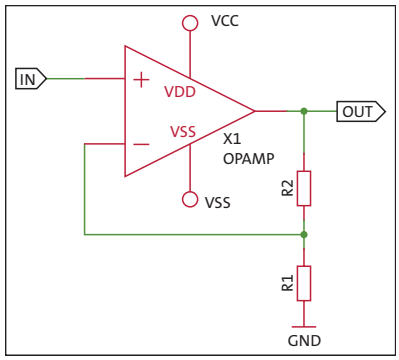

Abbildung 3.25 Nichtinvertierender Operationsverstärker

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, der Verstärkungsfaktor $v = U_{\mathrm{OUT}}/U_{\mathrm{IN}}$ ist daher 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* ($n = 4$).

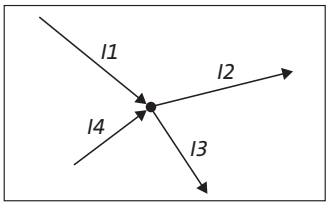

Abbildung 3.26 Knotenpunkt mit hinein- und hinausfließenden Strömen

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.

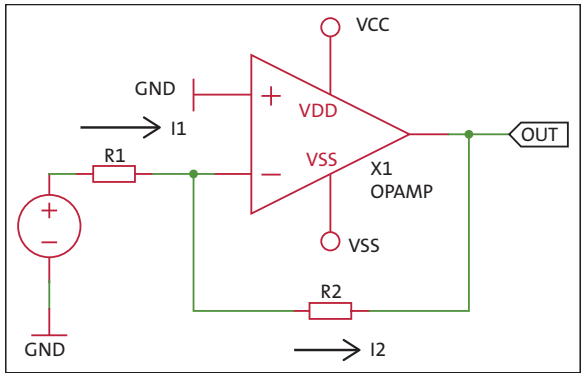

Abbildung 3.27 Invertierender Verstärker

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, »Operationsverstärker«, 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:

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

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

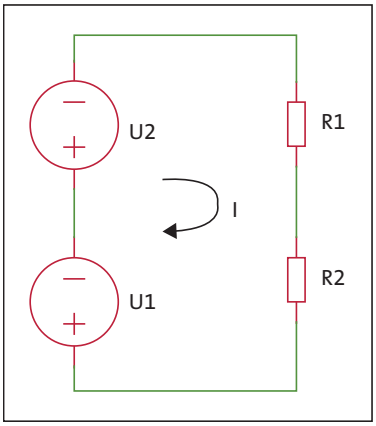

Abbildung 3.28 Maschensatz

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.

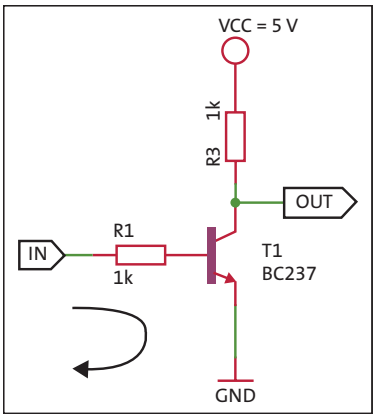

Abbildung 3.29 Inverterstufe

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 $B_{\mathrm{N}} = 100$, $m = 5$ und $U_{\mathrm{CES}} = 0{,}1$ 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.

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.

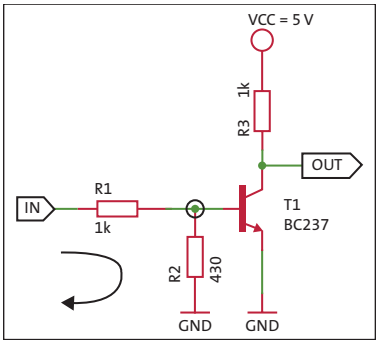

Abbildung 3.30 Erweiterte Inverterstufe

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).

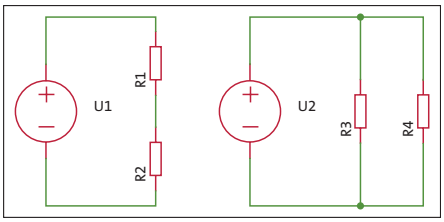

Abbildung 3.31 Reihen- und Parallelschaltung von Widerständen

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 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.

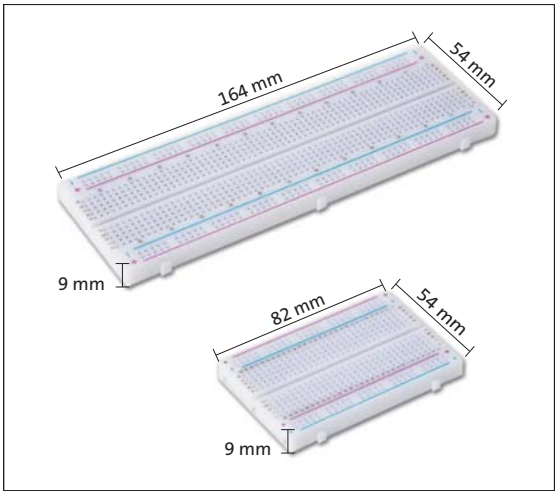

Abbildung 3.32 Breadboards verschiedener Größe

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

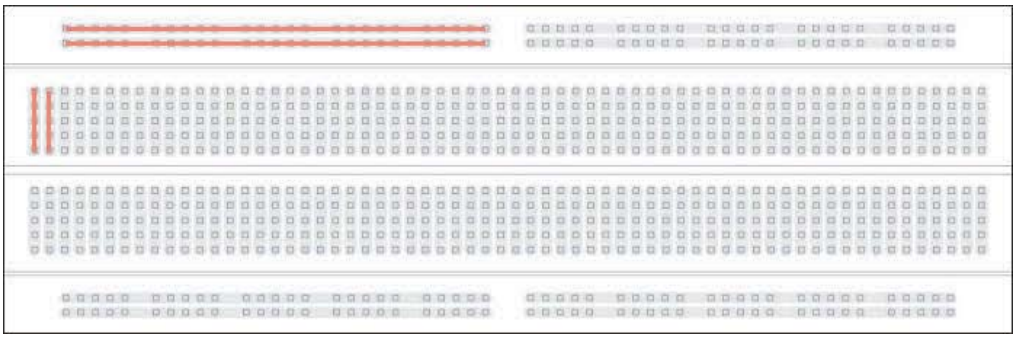

Abbildung 3.33 Verbundene Lochreihen

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.

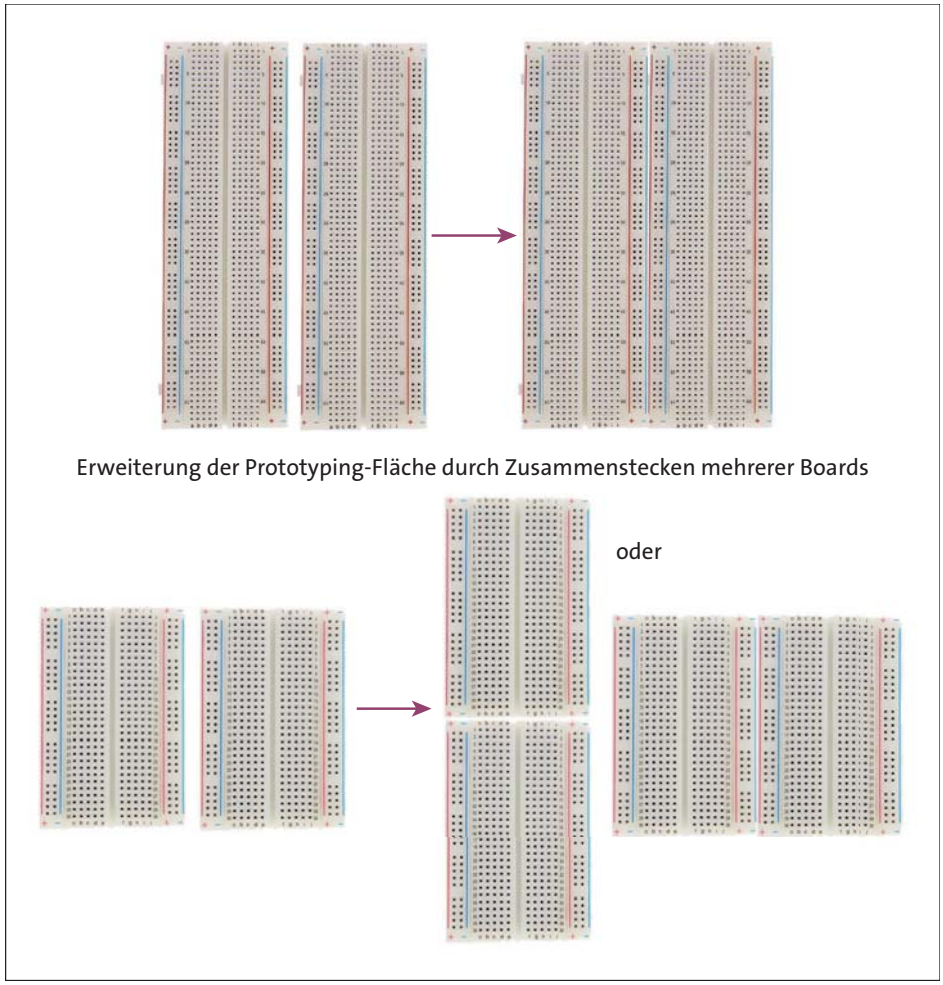

Abbildung 3.34 Kombination von Breadboards

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.

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.

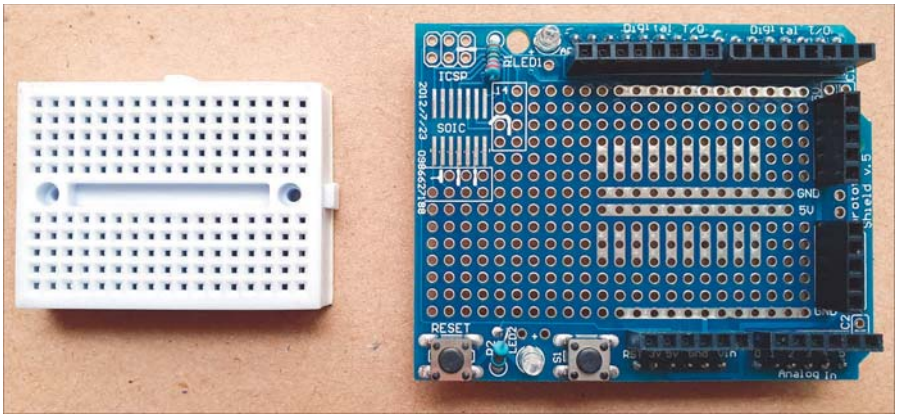

Abbildung 3.35 Prototype Shield v.5

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.

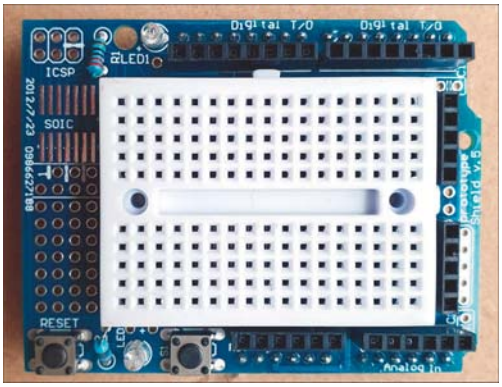

Abbildung 3.36 Prototype Shield v.5 mit aufgesetztem Breadboard

Benötigen Sie etwas mehr Platz auf dem Breadboard, dann können Sie den von SparkFun für 3,95 USD 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.

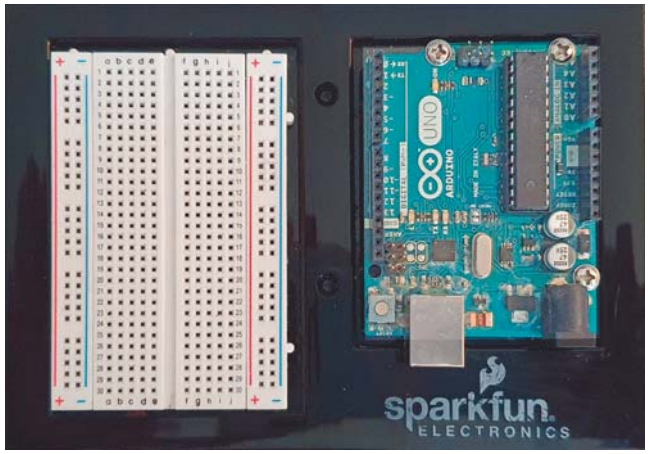

Abbildung 3.37 Breadboard Holder (Steckbrett-Halter)

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.

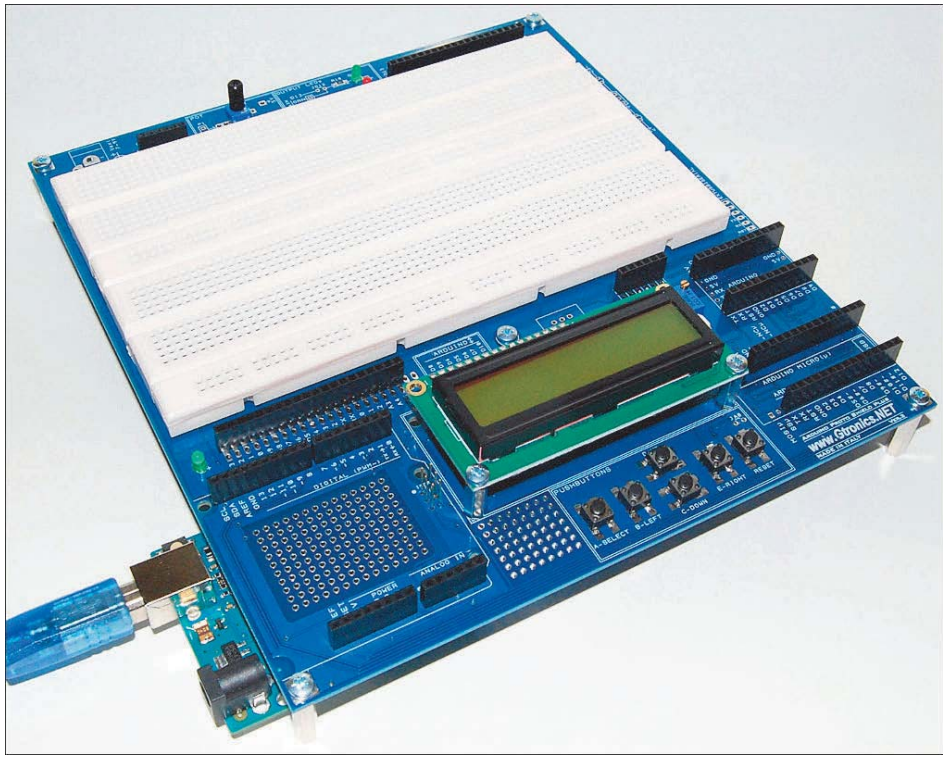

Abbildung 3.38 Arduino Protoshield (Quelle: Gtronics.NET)

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.

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.

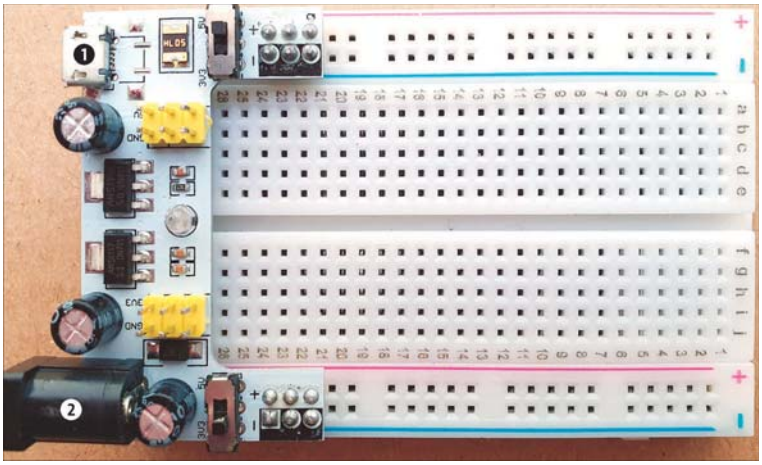

Abbildung 3.39 Breadboard Power Adapter

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.

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 es ändert sich 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 die »Knightbright Full Color LED Lamp WP154A4SUREQBF-ZGC« (Quelle: Auszug aus dem Datenblatt)

Bei der Berechnung des jeweiligen LED-*Vorwiderstands* müssen Sie diese Unterschiede und die Art der Anschaltung an den Arduino 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 Steuerung von LEDs einsetzen, ist Geschmackssache. In den Anwendungen finden Sie beide.

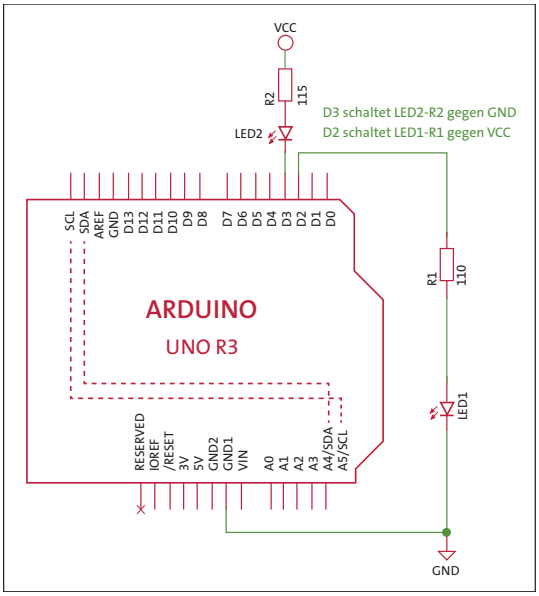

Abbildung 7.1 LEDs am Arduino Uno

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, und I_F steht 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 rührt auch der Name der Maßeinheit *Candela* her (lateinisch 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.

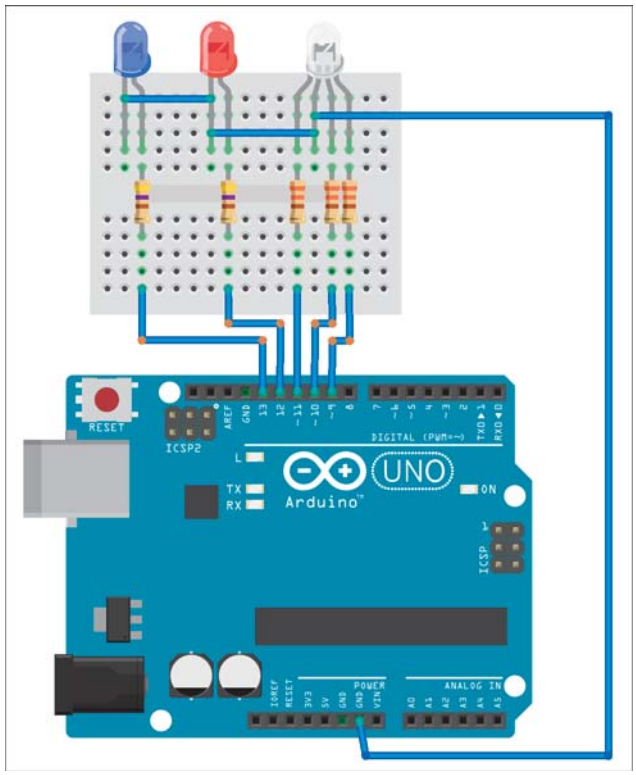

Abbildung 7.2 LEDs an Arduino Uno

Wenn Sie den Schaltungsaufbau vermeiden wollen und im Besitz eines *YwRobot Easy Module Shields* sind, das ich in Abschnitt 2.3.1 vorgestellt habe, dann können Sie dieses Shield 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);
```

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.

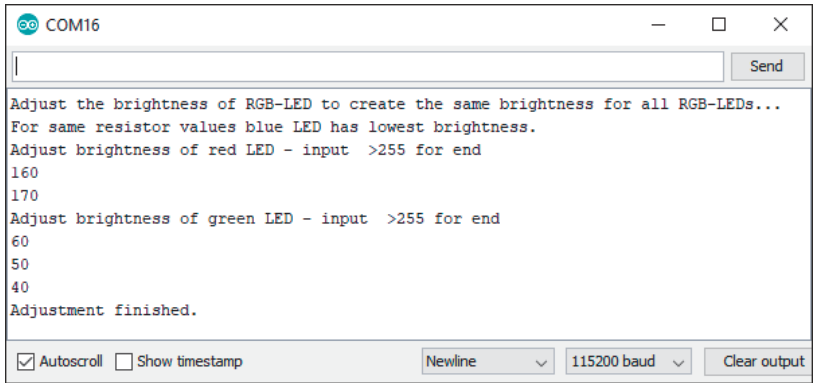

Abbildung 7.3 Abgleich der Helligkeiten einer RGB-LED

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-Anzeigen

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 zwischen dem Sieben-Segment-Display (mit TM1637) und dem 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*).

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.

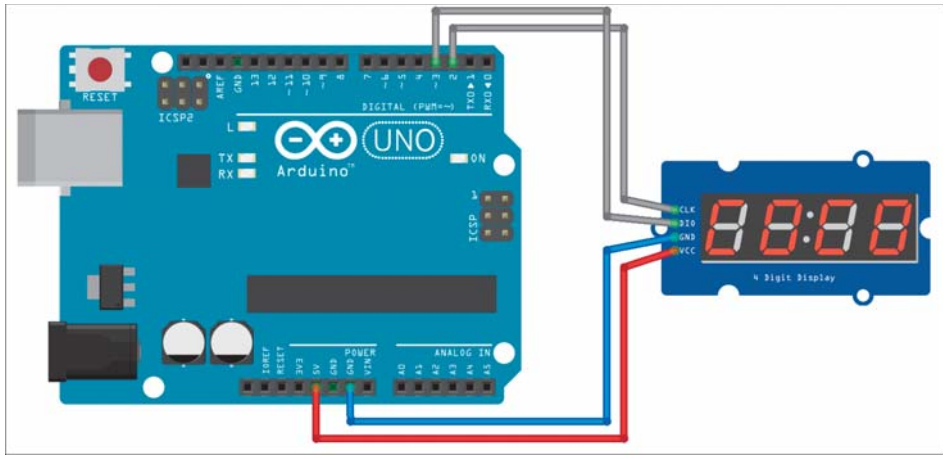

Abbildung 7.5 Sieben-Segment-Anzeige mit TM1637 am Arduino Uno

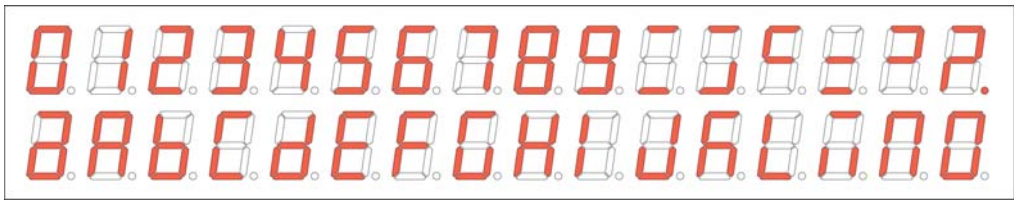

Abbildung 7.6 Alphanumerische Anzeige mit einem Sieben-Segment-Display

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 bzw. MAX7221 als Treiber einsetzen.

7.3 LED-Dot-Matrix-Anzeigen

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.

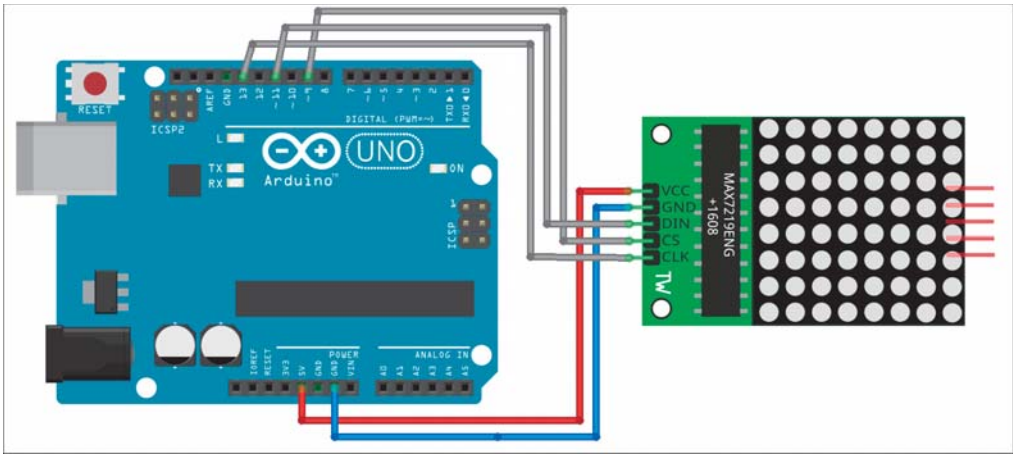

Abbildung 7.10 Dot-Matrix-Anzeige am Arduino Uno

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).

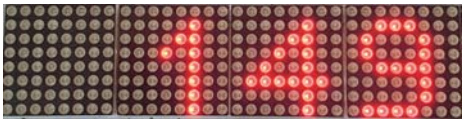

Abbildung 7.11 Anzeige des Programms »Uno_LEDMatrix.ino«

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.

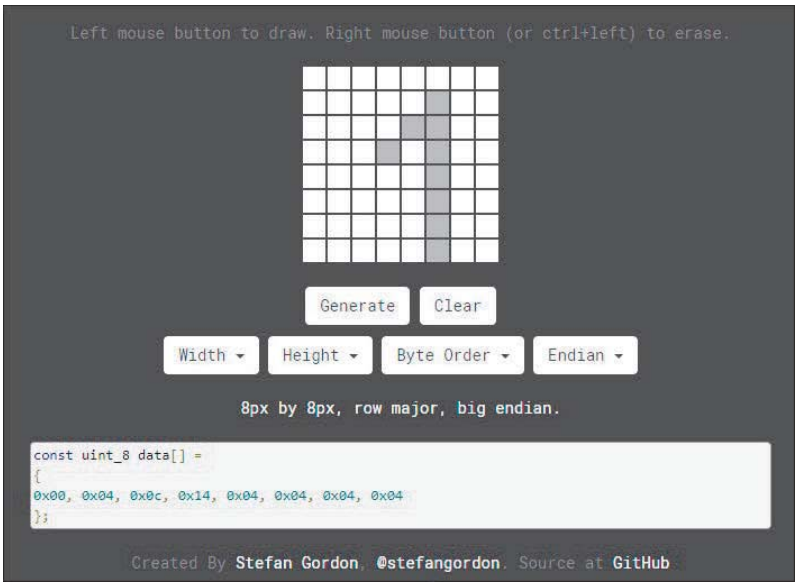

Abbildung 7.12 DotMatrixTool

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, die RGB-LEDs und die 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

Die chinesische Firma *WorldSemi* (*http://www.world-semi.com/*) bietet mit dem *WS2812* eine Kombination aus einer Steuerschaltung WS2811 und einem RGB-LED-Chip an, 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 × 5 mm) dargestellt ist. Adafruit bezeichnet diese Bauelemente als *NeoPixel*, und dieser Begriff hat sich eingebürgert. Das 5050-Gehäuse eignet sich 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

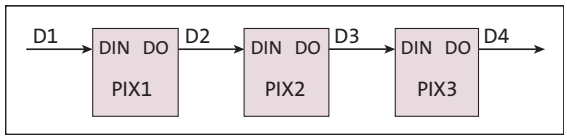

Abbildung 7.14 NeoPixel-Kaskadierung

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.

Kapitel 12
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 Kapitel 13 in Programmen noch einsetzen, sodass Sie hier erste Grundlagen zu ihnen finden.

12.1 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 und 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*). Im Januar 2024 wurde ArduinoJson 6 durch ArduinoJson 7 ergänzt.

ArduinoJson – Version beachten

Die 8-Bit-Mikrocontroller werden auch im Arduino-Umfeld weitgehend durch 32-Bit-Mikrocontroller ergänzt bzw. sogar abgelöst. Diese Hardware-Revolution erforderte ein Redesign von ArduinoJson.

Obwohl sie einst ein Wettbewerbsvorteil von ArduinoJson 6 war, wurde die feste Speicherzuweisung nach und nach irrelevant. ArduinoJson 7 definiert die Speicherverwaltungsstrategie neu, um eine elegante und unkomplizierte API zu bieten.

Allerdings ist ArduinoJson 7 auch deutlich größer als Version 6. Die Programme nehmen auf einem Arduino Uno R3 erheblich mehr Platz ein (Parser-Beispiel: +41%, Generator-Beispiel: +45%).

Beachten Sie deshalb unbedingt die verwendete Version von ArduinoJson.

Das in Listing 12.1 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* hilft, die Größe des JSON-Buffers festzulegen, und bietet Unterstützung bei der Anwendung von ArduinoJson (*https://arduinojson.org/v7/assistant*). Achten Sie auch hier unbedingt auf die Version, denn es sind noch viele Programme im Umlauf, die die (veraltete) Version v5 oder v6 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 12.1 Quelltext von »Uno_JsonOutputv6.ino«

Wie die Terminalausgabe des Programms *Uno_JsonOutputv6.ino* in Abbildung 12.1 zeigt, gibt es zum einen den *minified JSON-Output* (minimale Ausgabe) und zum anderen den *prettified JSON-Output*, also eine »schöne« Ausgabe, die sicher etwas einfacher lesbar ist. Entscheiden Sie, welche Variante Sie bevorzugen.

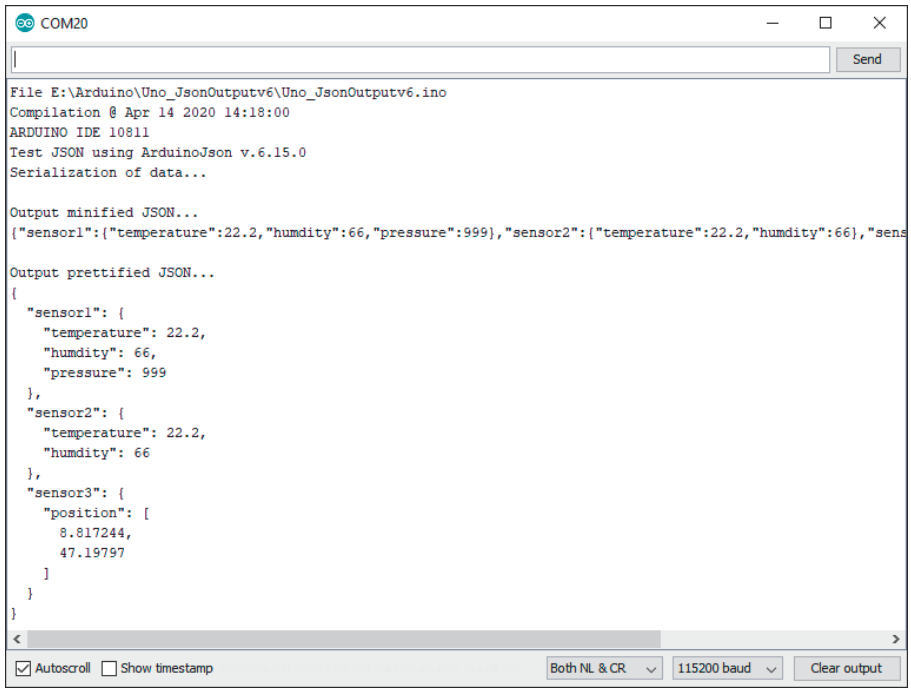

Abbildung 12.1 Terminalausgaben im JSON-Format

12.2 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.

12.2.1 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, bei 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 12.2 zeigt das betreffende Modell.

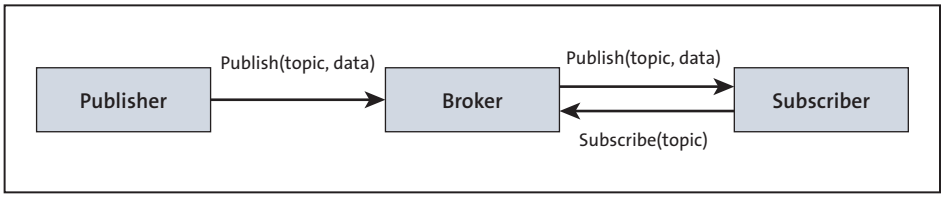

Abbildung 12.2 Publish/Subscribe Message Queueing Model

Wie Abbildung 12.2 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 das 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 muss er das Topic `DHT11/+/#` abonnieren.

MQTT wird oft in unsicheren Netzwerken eingesetzt, und es kann vorkommen, 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.

12.2.2 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 aber nicht diesen Weg gehen.

Eine andere Variante ist, einen MQTT-Broker zu verwenden, der als Cloud-Applikation zur Verfügung steht (*Hosted MQTT-Broker*). Diese Variante hat den Vorteil, dass man die Grenzen seines abgeschlossenen Netzwerks 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.

Die *HiveMQ*-MQTT-Broker sind eine gute Adresse zum Testen von MQTT. Über den *MQTT Websocket Client* (*https://www.hivemq.com/demos/websocket-client/*) haben Sie Zugriff auf den *HiveMQ Public Broker* (siehe Abbildung 12.3).

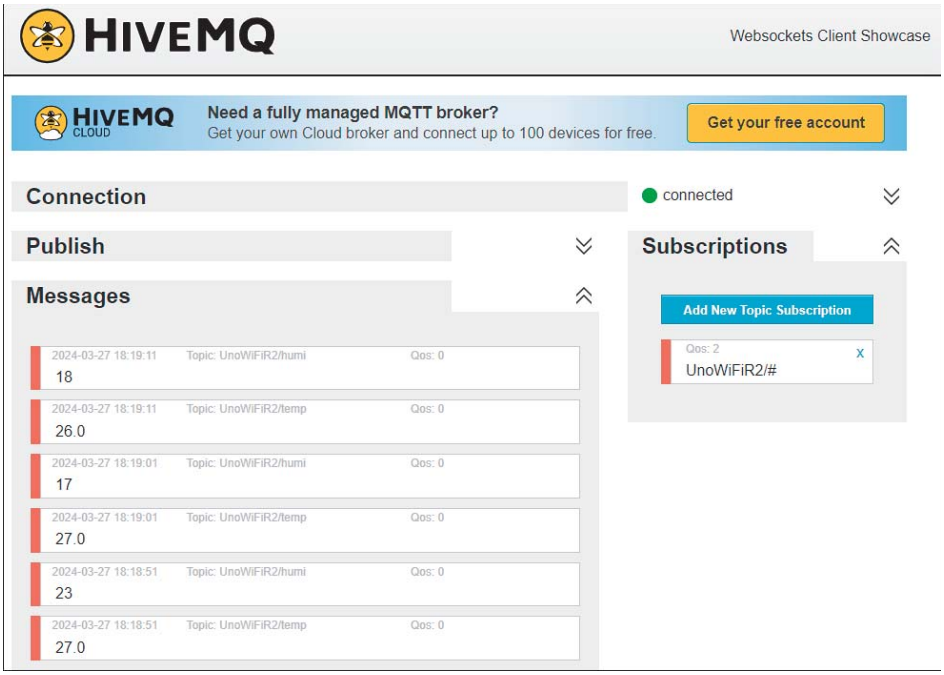

Abbildung 12.3 MQTT-Messages im »HiveMQ Public Broker«

12.2.3 MQTT-Client

Damit der MQTT-Broker auch die in Abbildung 12.3 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. (Für den ESP32 gibt es ein vergleichbares Programm namens *ESP32_DHT11_MQTT.ino*.)

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 abgelegt (wie Zugangsdaten und Vergleichbares).

```
#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 `PubSubClient`s 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()` (siehe Listing 12.2) wird 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 12.2 Quelltext von »UnoWiFiR2_DHT11_MQTT.ino« – die »setup()«-Funktion

In der Funktion `loop()` (siehe Listing 12.3) spielt sich nach dem Verbindungsaufbau alles Weitere ab. Alle 30 Sekunden wird der DHT11-Sensor nach seinen Messwerten abgefragt. 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 12.3 Quelltext von »UnoWiFiR2_DHT11_MQTT.ino« – die »loop()«-Funktion

Die schrittweise Abarbeitung des Programms können Sie leicht anhand der Ausgaben in Abbildung 12.4 nachverfolgen:

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

Auf meinem Windows-PC verwende ich gern den *HiveMQ Public Broker* und auf dem Smartphone *MQTT Dashboard*.

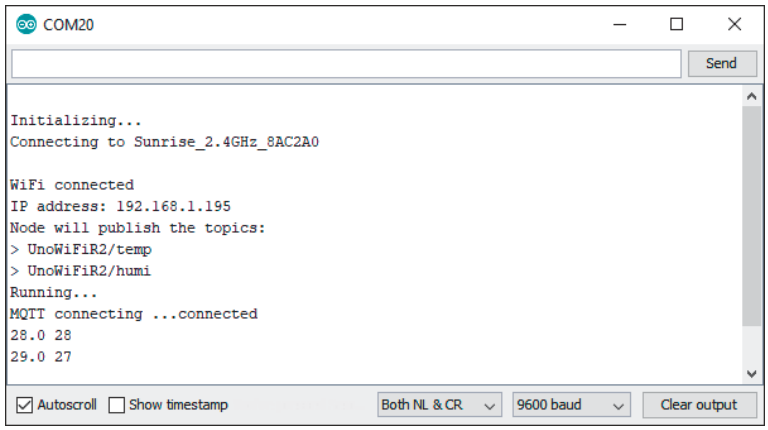

Abbildung 12.4 Ausgaben des Programms »UnoWiFiR2_DHT11_MQTT.ino«

Die Daten im HiveMQ Public Broker haben Sie bereits mit Abbildung 12.3 gesehen. Die Subscription lautet hier `UnoWiFiR2/#`. Das heißt, alle Topics, die mit `UnoWiFiR2/` beginnen, sind abonniert und werden angezeigt.

Einen weiteren 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.