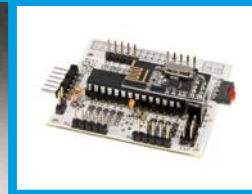
O'REILLY®

Hacks

Peter Recktenwald

Hacks für die Digitale Fotografie

Digitalfotografie mit Arduino und Raspberry Pi



Hacks für die Digitale Fotografie

Peter Recktenwald

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

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

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

Lektorat: Volker Bombien, Köln
Korrektorat: Tanja Feder, Bonn
Umschlaggestaltung: Michael Oreal, Köln (mit Abbildungen von Peter Recktenwald)
Produktion: Karin Driesen, Köln
Satz: le-tex publishing services GmbH, Leipzig, *www.le-tex.de*
Belichtung, Druck und buchbinderische Verarbeitung: Druckerei Kösel, Krugzell; *www.koeselbuch.de*

ISBN 978-3-95561-644-1

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Im Gedenken an meine Eltern.

Inhaltsverzeichnis

Vorwort

Im August letzten Jahres (2013) erhielt ich eine E-Mail von Volker Bombien, Lektor des O'Reilly Verlages. Ob ich nicht Lust hätte, mit ihm über ein Buchprojekt zum Thema „Arduino und Digitalfotografie" zu plaudern. Damit fing alles an, aber es war noch ein langer Weg und es brauchte schon eine gehörige Portion Überzeugungsarbeit, um mich dazu zu bringen dieses Buch zu schreiben, das Sie heute in den Händen halten. Letztendlich ließ ich mich doch breitschlagen und schnell änderte sich meine anfängliche Skepsis in Begeisterung. Wie so oft im Leben und erst recht beim Schreiben dieses Buches wuchsen kleine Dinge schnell zu großen Themen heran. So geschehen beim Foto Controller; der sich mit der Zeit zu einem wahren Mammutprojekt entwickelte und zum Schluss gleich mehrere Kapitel des Buchs in Anspruch nahm. Das war zwar anfangs so nicht geplant, aber letztendlich wurde daraus der rote Faden, der sich durch das Buch zieht.

Ich wünsche viel Spaß beim Nachbau der Projekte und hoffe das damit viele interessante Fotos entstehen. Wer möchte kann seine Fotos mit anderen in den sozialen Medien teilen oder darüber diskutieren. Aus diesem Grund habe ich verschiedene Gruppen eingerichtet:

- *Flickr Gruppe „DigitalfotografieHacks" (https://www.flickr.com/groups/2707154@N20/)*
- *Google+ „DigitalfotografieHacks" (https://plus.google.com/u/0/communities/102851126817521786624)*
- Twitter Hashtag `#DigitalFotografieHacks`

Peter Recktenwald

Berlin, im September 2014

Danksagung

Besonderen Dank geht an meinen Lektor, Volker Bombien, für die tatkräftige Unterstützung bei der Umsetzung des Buches. Er hatte immer ein offenes Ohr für meine Problemchen und fand stets für alles eine Lösung.

Ebenso geht ein großes Dankeschön an die Korrektorin Tanja Feder, die mein Yoda-Sprech in eine lesbare Form gebracht hat.

Meiner Familie danke ich für die Unterstützung und das Verständnis, dass ich mich für recht lange Zeit in mein Kämmerlein zurückgezogen habe und kaum noch Zeit für sie hatte.

Meinen Beta-Lesern, Freunden und Kollegen, die mir mir Rat und Tat zur Seite standen, möchte ich auch auf diese Weise meinen Dank aussprechen.

1
Basic Hacks

Dies ist ein etwas anderes Fotografiehandbuch. Eigentlich handelt es sich gar nicht um ein solches im eigentlichen Sinne. Es geht weder darum zu zeigen, wie man bessere Fotos schießt oder wie man seine Technik als Fotograf verbessert, noch darum, welches Equipment für die Fotografie erforderlich ist. Vielmehr richtet sich dieses Buch an Fotografen, die gerne auch mal experimentieren oder selber etwas basteln, oder an Bastler, die nebenbei auch noch fotografieren. Sicher kennt jeder die faszinierenden Bilder der Tropfenfotografie. Auch mich hat es da sofort gepackt: Wie macht man solche Fotos? Das will ich auch können!

Die Bastler- oder Maker-Szene, wie sie jetzt neudeutsch heißt, hat in den letzten Jahren einen bedeutenden Aufschwung erlebt. Durch preiswerte Boards wie Arduino und Raspberry Pi sowie zahlreiche Anleitungen im Internet können heutzutage auch technische Laien etwas nachbauen, ohne dass sie unbedingt alle Details verstehen müssen. Aber was ist, wenn eine Schaltung oder ein Programm für bestimmte Zwecke geändert werden muss? Wenn das hierfür erforderliche Wissen fehlt, ist man in solchen Fällen tatsächlich aufgeschmissen. Das vorliegende Buch soll hier eine Hilfestellung bieten, indem ein entsprechendes Grundwissen vermittelt wird und die Schaltungen und Programme in ihrer Funktion erläutert werden.

Bei den im Buch abgedruckten Quellcodes handelt es sich meistens nur um Auszüge und keine kompletten Programme. Der komplette Programm Code steht auf der Website zu diesem Buch zum *Download (https://github.com/robotfreak/dfhacks)* bereit, ebenso wie die Hardware-Layouts der vorgestellten Schaltungen. Allzu tief in die Materie Elektronik und Programmierung möchte ich in diesem Buch allerdings nicht eintauchen, denn man kann nicht immer qbei Adam und Eva beginnen. Auf weiterführende Links wird an entsprechender Stelle im Buch hingewiesen.

Alle Hacks dieses Buches sind in die Schwierigkeitsgrade leicht, mittel und schwierig eingeteilt. Dies erkennen Sie an den Ampelfarben der Hack-Icons, wobei grün für einen leicht zu verstehenden Hack steht.

HACK 01 Elektronik für Dummys

Ohne ein wenig Grundwissen geht es leider nicht. Beginnen wir mit der Elektronik und kratzen ein wenig an der Oberfläche. Es geht doch nichts über eine gute Portion gesunden Halbwissens.

Elektronische Größen und Einheiten

Strom, Spannung und Widerstand sind die elementaren Größen der Elektrotechnik. Das Ohmschen Gesetz ist vielleicht dem einen oder anderen bekannt. Danach verhält sich der Spannungsabfall über einem Widerstand proportional zum Strom. Der elektrische Widerstand hat das Formelzeichen *R* und die Maßeinheit Ohm. Es handelt sich dabei um die Spannung, die erforderlich ist, um eine bestimmte Stromstärke durch einen Leiter zu schicken.

$$R = \frac{U}{I} \tag{1.1}$$

Aus diesem Grundsatz ergeben sich zwei weitere Formeln zur Berechnung der Spannung und des Stroms. Der elektrische Strom hat das Formelzeichen *I* (lateinisch influare) und die Maßeinheit Ampere, kurz A. Als elektrischer Strom wird die Bewegung von elektrischen Ladungen bezeichnet, dazu gehören z. B. die freien Elektronen in einem Metall.

$$I = \frac{U}{R} \tag{1.2}$$

Die elektrische Spannung hat das Formelzeichen *U* (lateinisch urgere) und die Maßeinheit Volt, kurz V. Unter elektrischer Spannung versteht man einen Potentialunterschied zwischen zwei Punkten. Häufig wird ein Bezugspunkt – die Masse bzw. der Minuspol der Spannungsquelle – definiert, um innerhalb einer Schaltung verschiedene Spannungen zu messen und vergleichen zu können.

$$U = R \times I \tag{1.3}$$

Hilfreich für das Einprägen dieser drei Formeln ist das sogenannte URI-Dreieck.

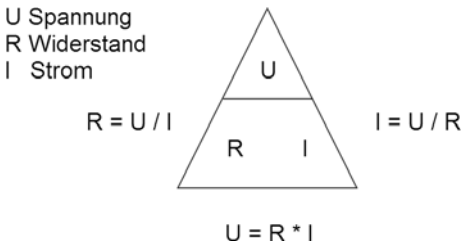


Abbildung 1-1: URI-Dreieck

Das Zusammenwirken von Strom, Spannung und Widerstand lässt sich am besten anhand eines Wasserkreislaufs veranschaulichen. Der Strom entspricht dabei dem Wasser, die Spannung dem Wasserdruck und der Widerstand dem Durchmesser des

Wasserrohrs. Je höher der Wasserdruck (Spannung) ist, desto mehr Wasser (Strom) fließt. Je dünner das Rohr ist (größerer Widerstand), desto weniger Wasser (Strom) kann durch das Rohr fließen.

Noch ein paar Anmerkungen zur Spannungsquelle, oder besser gesagt zur Energiequelle. Die Spannung kann man entweder vom E-Werk bzw. auf unser Wassermodell bezogen vom Wasserwerk beziehen. Die Spannung bzw. der Wasserdruck ist dabei konstant und wird vom Netzbetreiber vorgegeben. Die Spannung vom E-Werk muss allerdings noch gleichgerichtet werden, weil es sich ja um Wechselspannung handelt, aber das wollen wir hier vernachlässigen. Anders, wenn man eine mobile Energiequelle, sprich einen Akku oder eine Batterie verwendet. In unserem Wassermodell entspräche die Batterie einem Wasserbassin. Die Spannung wäre analog zur Wassertiefe des Bassins. Je tiefer der Wasserstand im Becken ist, desto höher die Spannung. Wenn nun Strom aus der Energiequelle entnommen bzw. Wasser aus dem Bassin gelassen wird, sinkt der Wasserspiegel bzw. die Spannung der Batterie. Übrigens spielt die Größe des Bassins für den Wasserdruck – sprich die Spannung – keine Rolle. Die Größe des Beckens entspricht dem Energieinhalt der Batterie, die entsprechende Maßeinheit lautet Amperestunde, kurz Ah. Je größer das Bassin bzw. der Energieinhalt, desto länger fließt das Wasser bzw. desto länger liefert die Batterie Strom.

Zum Wasserkreislauf bzw. zum Vergleich des elektrischen Stroms mit Wasser bliebe noch Folgendes anzumerken: Während Wasser der Schwerkraft folgend nur bergab fließt, gilt diese Einschränkung für Strom natürlich nicht. Auch die Annahme, dass der elektrische Strom von Pluspol zum Minuspol fließt, entspricht nicht ganz den Tatsachen, weil Elektronen nun einmal negativ geladen sind. Deshalb spricht man in diesem Zusammenhang auch von der technischen Stromrichtung.

Eine weitere elektrische Größe ist die *elektrische Leistung (http://de.wikipedia.org/wiki/Elektrische_Leistung)*. Die elektrische Leistung hat das Formelzeichen *P* (Power) und die Maßeinheit Watt, kurz W.

$$P = U \times I \tag{1.4}$$

Elektronische Bauteile

Nachfolgend werden einige der wichtigsten elektronische Bauteile vorgestellt.

Passive Bauteile

Zu den passiven Elektronik Bauteilen zählen unter anderem Widerstände und Kondensatoren.

Bei einem *Widerstand (http://de.wikipedia.org/wiki/Widerstand_(Bauelement))* handelt es sich um ein passives Bauteil, das – wie der Name schon vermuten lässt – dem Strom einen Widerstand entgegensetzt. Eingesetzt werden Widerstände zur Strombegrenzung (Vorwiderstand bei LEDs) und als Spannungsteiler, um definierte

Pegel einstellen (Pullup- und Pulldown-Widerstände). Die Kenngröße Widerstand, kurz *R* (Resistor), wird in der Maßeinheit Ohm angegeben.

Die folgende Abbildung zeigt zwei Arten von Widerständen, einen Festwiderstand und einen variablen Widerstand, oder besser ausgedrückt einen Widerstand mit variablem Mittenabgriff, auch Potentiometer oder Trimmer genannt. Rechts daneben befinden sich die entsprechenden Schaltbilder, oben die in Europa übliche DIN-Darstellung, darunter die amerikanische Version nach ANSI.

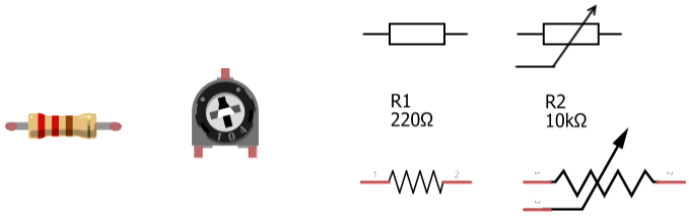


Abbildung 1-2: Widerstände

Es gibt noch einige weitere Bauelemente, die sich in die Rubrik Widerstand einordnen lassen und deren Widerstand sich unter bestimmten Umständen ändern kann. Dazu zählt der LDR (Light Dependent Resistor), dessen Widerstand von der Umgebungshelligkeit abhängig ist. Dann gibt es Bauteile, deren Widerstand von der Temperatur abhängig ist. Sie werden als PTC und NTC bezeichnet. Beim PTC (Positive Temperatur Coefficient) oder *Kaltleiter (http://de.wikipedia.org/wiki/Kaltleiter)* steigt der Widerstand mit steigender Temperatur, beim NTC (Negative Temperature Coefficient) oder *Heißleiter (http://de.wikipedia.org/wiki/Hei%C3%9Fleiter)* sinkt er entsprechend.

Als Grundschaltungen für Widerstände sind die Reihenschaltung und die Parallelschaltung zu nennen. Bei der Reihenschaltung teilt sich die Spannung *U* über den Widerständen *R*1 und *R*2 auf, deshalb spricht man auch von Spannungsteiler.

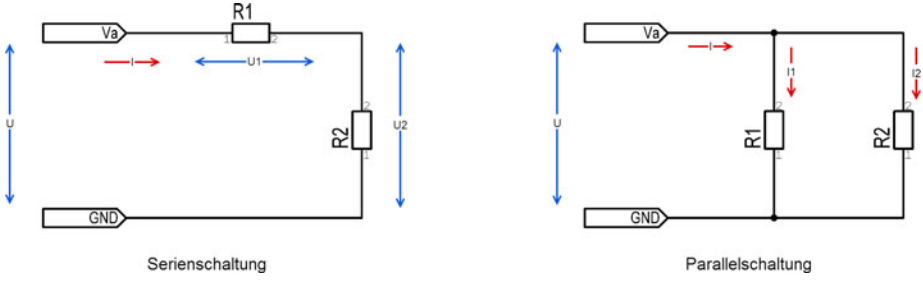


Abbildung 1-3: Grundschaltungen für Widerstände

$$I = \frac{U}{R1 + R2} \tag{1.5}$$

$$U = U1 + U2 \tag{1.6}$$

$$U = (R1 + R2) \times I \tag{1.7}$$

$$U1 = U \times \frac{R1}{R1 + R2} \tag{1.8}$$

$$U2 = U \times \frac{R2}{R1 + R2} \tag{1.9}$$

Bei der Parallelschaltung teilt sich der Gesamtstrom *I* auf die beiden Teilströme *I*1 und *I*2 auf.

$$I = I1 + I2 \tag{1.10}$$

$$U = R1 \times I1 \tag{1.11}$$

$$U = R2 \times I2 \tag{1.12}$$

$$I1 = \frac{U}{R1} \tag{1.13}$$

$$I2 = \frac{U}{R2} \tag{1.14}$$

Bei *Kondensatoren (http://de.wikipedia.org/wiki/Kondensator_(Elektrotechnik))* handelt es sich um Ladungs- und Energiespeicher. Eingesetzt werden sie z. B. als Filter bei Störsignalen und zum Glätten von Spannungseinbrüchen. Auch zum kurzzeitigen Puffern der Versorgungsspannung werden sogenannte Superkondensatoren (Gold-Caps) verwendet. Die wichtigste Kenngröße von Kondensatoren ist die Kapazität, kurz *C* (Capacity) genannt. Die Maßeinheit für die Kapazität ist das Farad, F. Da es sich bei einem Farad um eine recht große Kapazität handelt, weisen gängige Kondensatoren Kapazitäten auf, die im Bereich von µF (microFarad 10^{-6} F) bis pF (picoFarad 10^{-12} F) liegen.

Die folgende Abbildung zeigt einige Beispiele für Kondensatoren. Von links nach rechts sind dies ein Folien- oder Keramikkondensator, ein Tantal- und ein Elektrolytkondensator (Elko) sowie deren Schaltbilder. Bei den beiden letztgenannten handelt es sich um gepolte Bauelemente, dass heißt, beim Einbau muss auf die Polung geachtet werden.

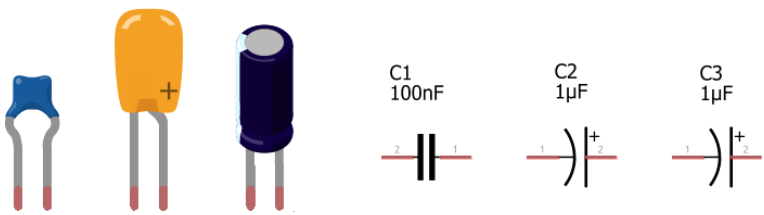


Abbildung 1-4: Kondensatoren

Halbleiter

Zu den Halbleiter-Bauteilen zählen Dioden, Transistoren und natürlich ICs, die wiederum aus einer Vielzahl an Dioden und Transistoren bestehen. Der Name Halbleiter

stammt von einer Eigenschaft dieser Bauteile, nämlich der, dass sie den Strom mal leiten und mal nicht. Silizium ist der Grundstoff, aus denen die meisten Halbleiter bestehen. Das "Silicon Valley" in Kalifornien hat nichts mit Silikon zu tun, sondern trägt das Wort Silizium im Namen, und das nicht etwa, weil es dort sehr viel Silizium gibt, sondern weil an diesem Ort viele Firmen ansässig sind, die mit Halbleitern ihr Geld verdienen. Silizium gibt es quasi wie Sand am Meer, es ist nämlich der Grundstoff vieler Mineralien, auch Quarzsand gehört dazu. Die Halbleitertechnik bescherte der Menschheit eine technische Revolution, sie machte Flüge zum Mond erst möglich und brachte uns Computer und Smartphones.

Dioden (http://www.wikipedia.de/Diode) sind die einfachsten Halbleiter-Bauelemente. Sie bestehen aus einem PN-Übergang und lassen den Strom nur in eine Richtung fließen, in der Gegenrichtung sperren sie den Strom fast ganz. Sie funktionieren also im Prinzip wie ein Rückschlagventil in einem Wasserkreislauf.

Neben den Standard-Dioden gibt es auch noch einige Sondertypen, z. B. die *Schottky-Dioden (http://de.wikipedia.org/wiki/Schottky-Diode)*, die besonders schnell schalten. Anwendung finden Schottky-Dioden z. B. als Freilauf-Dioden zum Schutz vor Überspannungen beim Schalten induktiver Lasten wie Motoren und Relais. Sogenannte *Zener-Dioden (http://de.wikipedia.org/wiki/Z-Diode)*, die auch als Z-Dioden bezeichnet werden, leiten ab einer gewissen Spannung auch in Sperrrichtung. Ebenfalls zu den Dioden zählen die *Leuchtdioden (http://de.wikipedia.org/wiki/Leuchtdiode)* oder kurz LEDs. LEDs haben die Eigenschaft, dass sie in Durchlassrichtung betrieben werden können und außerdem Licht abgeben. Zu guter Letzt ist noch die *Fotodiode (http://de.wikipedia.org/wiki/Fotodiode)* zu nennen, die Licht in elektrischen Strom umwandeln kann. Auch Solarzellen zählen übrigens zu den Fotodioden.

Transistoren (http://www.wikipedia.de/Transistor) sind die Alleskönner unter den Halbleitern. Hauptanwendung ist das Schalten und Verstärken von elektrischen Signalen. Transistoren verfügen über zwei PN-Übergänge: einem kleineren PN-Übergang, der den größeren PN-Übergang quasi steuert. In unserem Modell eines Wasserkreislaufs entspräche das einem kleinen Rückschlagventil, das mit einem kleinen Wassermenge gesteuert ein größeres Ventil mit großer Wassermenge mitsteuert. Je nachdem, wie die beiden PN-Übergänge zusammengeschaltet werden, unterscheidet man Transistoren nach NPN (NP-PN)- und PNP (PN-NP)-Typen.

Aus mehreren Transistoren lassen sich alle Grundschaltungen der Digitaltechnik zusammenbauen, Gatterschaltungen wie AND-, NAND-, OR- und NOR-Gatter. Diese wiederum bilden die Grundlage für noch komplexere Schaltungen wie Addierer, Speicher und schließlich Prozessoren. Hierbei werden dann die Transistoren nicht als Einzelbauelemente verwendet, sondern als integrierte Schaltungen (ICs). Hier ein paar Zahlen zum Vergleich: Kam eine integrierte Schaltung wie der 7400 (4-NAND-Gatter) noch mit 16 Transistoren aus, besitzt ein moderner Quad-Core-Prozessor von Intel aus der Haswell-Reihe schlappe 1,4 Milliarden Transistoren.

Die folgende Abbildung zeigt ein paar Beispiele für diskrete Halbleiter. Links oben sehen Sie verschiedene Dioden, rechts oben Transistoren, darunter verschiedene ICs und ganz unten einige Mikrocontroller.

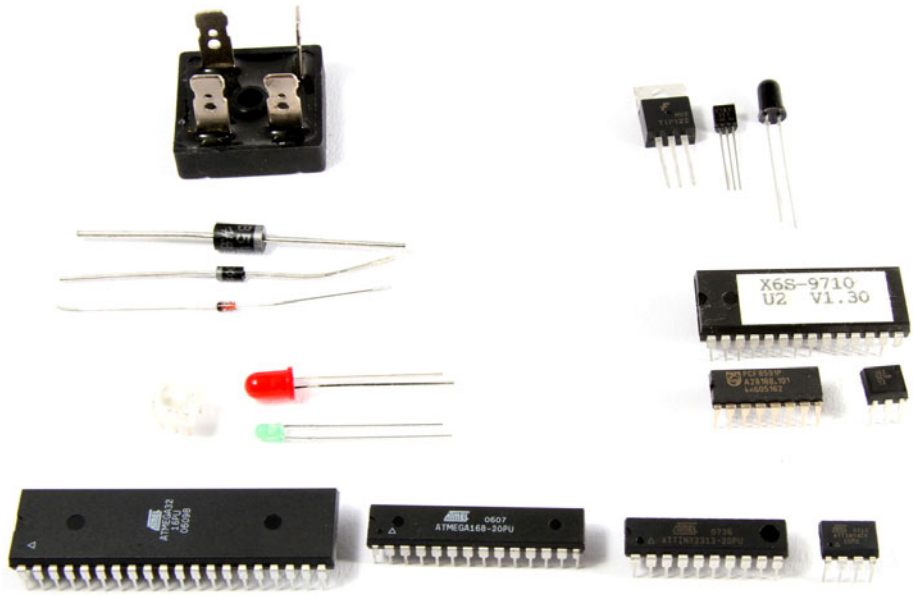


Abbildung 1-5: Halbleiter

Sehr lesenswerte Einführungen in die Grundlagen der Elektrotechnik finden Sie hier:

- *brucewilles.de (http://www.brucewilles.de/grundlagen.html)*
- *Elektronik-Kompendium.de (http://www.elektronik-kompendium.de/)*
- *Mikrocontroller.net – Elektronik Allgemein (http://www.mikrocontroller.net/articles/Elektronik_Allgemein)*

Integrierte Schaltkreise, kurz IC (Integrated Circuit) genannt, sind die Spezialisten unter den Halbleitern. Für praktisch jede denkbare Anwendung gibt es einen IC, angefangen von den noch recht allgemeinen Gatterschaltungen der 74er-Reihe bis hin zu den reinen Spezialisten wie ICs für Motor-Treiber oder LCD-Treiber.

Mikrocontroller

Mikrocontroller (http://de.wikipedia.org/wiki/Mikrocontroller) sind die nächst höhere Integrationsstufe in der Halbleitertechnik. Eine Besonderheit besteht hier darin, dass die Funktion des Bauteils nicht wie bei den "normalen" integrierten Schaltungen fest vorgegeben ist. Vielmehr kann ein Mikrocontroller programmiert werden, dass heißt, die eigentliche Funktion des Bauteils wird erst durch das Programm bestimmt, das der Programmierer in Form von Software entwerfen und als Programm-Code nie-

derschreiben muss. Dieser Programm-Code wird dann in eine für den Mikrocontroller verständliche Form, den Binär- oder Maschinen-Code, übersetzt.

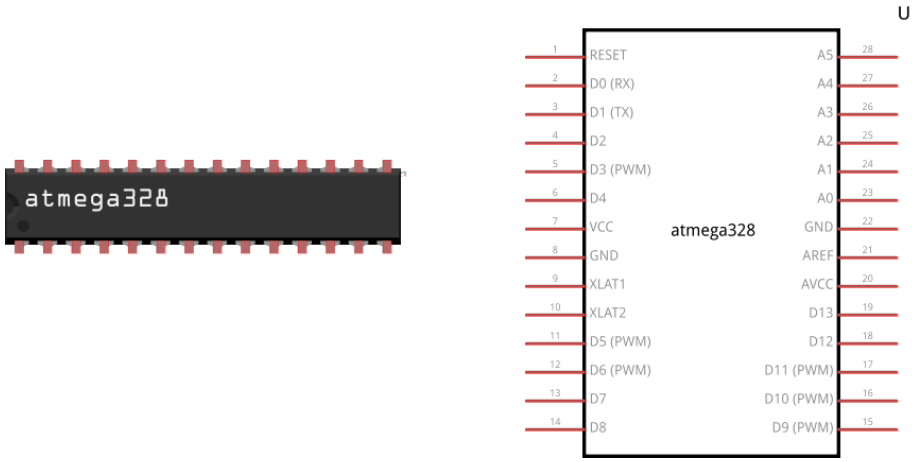


Abbildung 1-6: Mikrocontroller

Vereinfacht ausgedrückt besteht ein Mikrocontroller aus dem Prozessor (CPU) mit angeschlossener Peripherie. Das folgende Prinzipschaltbild zeigt die einzelnen Komponenten eines Mikrocontrollers.

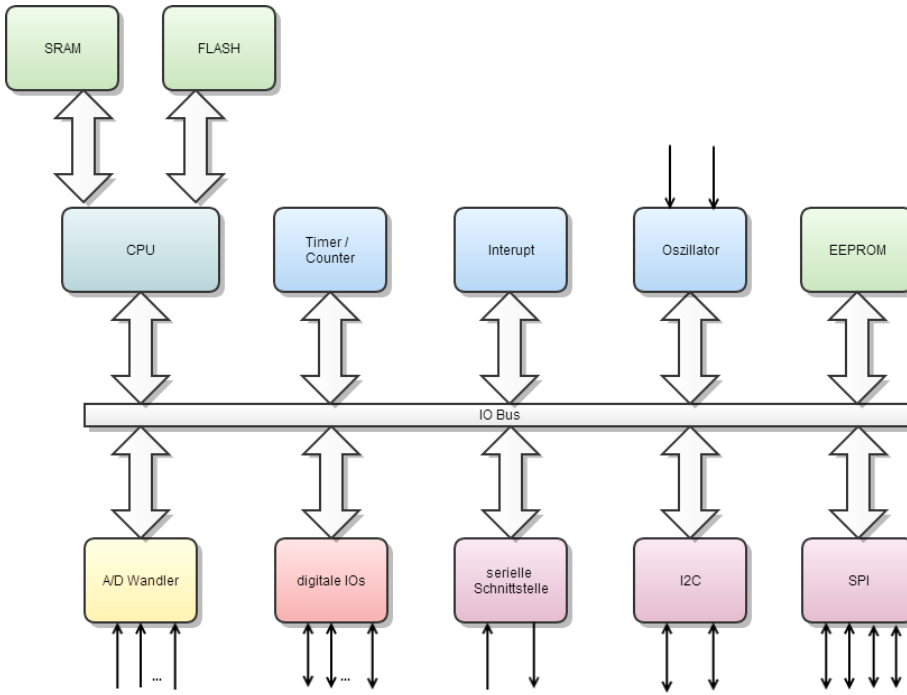


Abbildung 1-7: Prinzipschaltbild – Mikrocontroller

Prozessor

Beim Prozessor, kurz CPU (Central Processing Unit) genannt, handelt es sich um das Kernstück des Mikrocontrollers. In der CPU wird das Programm aus dem Flash-Speicher eingelesen und abgearbeitet. Neben dem Rechenwerk, kurz ALU (Arithmetic Logic Unit) genannt, verfügt die CPU außerdem über Registersätze (für Rechen- oder Vergleichsoperationen), einen Programmzeiger (zeigt auf den nächsten auszuführenden Befehl im Programmspeicher) und einen Stackzeiger (zeigt auf Rücksprungadressen von Funktionen, lokale Variablen und Registersätze, die im SRAM-Stack zwischengespeichert werden).

Speichertypen

Der *Programmspeicher (http://de.wikipedia.org/wiki/Flash-Speicher)*, auch Flash-Speicher genannt, enthält das ausführbare Programm. Der Programmspeicher ist ein nicht flüchtiger Speicher, dass heißt, sein Inhalt bleibt auch nach dem Abschalten des Stroms erhalten. Flash-Speicher kann nur in Blöcken geschrieben oder gelesen werden.

Beim *RAM (http://de.wikipedia.org/wiki/Random-Access_Memory)*, Random-Access-Memory (Speicher mit wahlfreiem Zugriff), handelt es sich um den Arbeitsspeicher des Prozessors. Dort werden alle Variablen abgelegt und bleiben bis zu einem Reset oder Abschalten des Stroms erhalten. Der Vorteil des RAM besteht in dem schnellen, wahlfreien Zugriff auf den Speicher.

EEPROM (http://de.wikipedia.org/wiki/Electrically_Erasable_Programmable_Read%2DOnly_Memory), Electrically Erasable Programmable Read Only Memory (elektrisch löschbarer und programmierbarer Nur-Lese-Speicher), ist ein weiterer nichtflüchtiger Speicher. Dieser kann auch wahlfrei gelesen und beschrieben werden. Die Zugriffe sind zwar recht langsam, dafür eignet sich der EEPROM-Speicher aber für Programm-Einstellungen, die zur Laufzeit geändert werden können, nach dem Abschalten allerdings erhalten bleiben sollen.

Programmierschnittstelle

Wie aber gelangt nun das Programm in den Programmspeicher des Prozessors? Normalerweise werden die Atmel-Prozessoren über die ISP-Schnittstelle programmiert. *ISP (http://de.wikipedia.org/wiki/In-System-Programmierung)*, In System Programming, bedeutet, dass sich der Prozessor direkt auf dem Board programmieren lässt. Zu diesem Zweck wird ein ISP-Programmier-Adapter benötigt. Dies ist ein kleines Board, meist mit eigenem Prozessor, das per Kabel mit der ISP-Schnittstelle auf dem Zielboard und seriell bzw. über USB mit dem PC verbunden wird. Auf diese Weise kann dann über eine spezielle Software für ISP-Programmierung (für einen Controller vom Typ Atmel AVR z. B. avrdude) ein kompiliertes Programm in den Programmspeicher geschrieben werden.

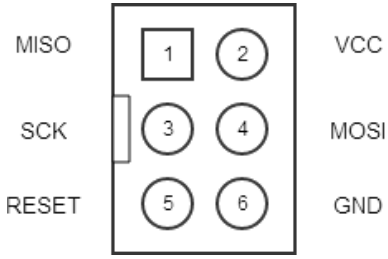


Abbildung 1-8: ISP-Steckverbinder

Aber keine Angst, bei vielen Mikrocontroller-Boards wie dem Arduino ist dies nicht erforderlich und es wird gar kein ISP-Programmer benötigt – der Prozessor "gaukelt" dem PC einfach vor, dass es sich bei ihm selbst um einen solchen Programmer handelt. Ermöglicht wird dies durch ein kleines Programm, den *Bootloader (http://de.wikipedia.org/wiki/Bootloader)*, der bereits vom Board-Hersteller in den Programmspeicher geschrieben wurde. Der Bootloader befindet sich am Ende des Programmspeichers in einem speziell geschützten Bereich. Der Prozessor ist so konfiguriert, dass nach jedem Prozessor-Reset zunächst das Bootloader-Programm ausgeführt wird. Der Bootloader überprüft, ob über die serielle Schnittstelle ein Programm-Upload erfolgen soll. Wenn dies der Fall ist, verhält sich der Bootloader wie ein ISP-Programmer, und das neue Programm wird in den Programmspeicher geschrieben. Anderenfalls wird einfach das letzte vorhandene Programm gestartet.

Der Vollständigkeit halber soll hier noch ein weiterer Programmier-Mode, nämlich die HV-Programmierung oder High Voltage-Programmierung, angeführt werden. HV-Programmierung erfolgt in einem speziellen HV-Programmer. Es handelt sich dabei normalerweise um das letzte Mittel, den Prozessor zu programmieren, falls man sich durch Unachtsamkeit "ausgesperrt" hat. Es besteht nämlich auch die Möglichkeit, die ISP-Schnittstelle abschalten, die Taktquelle zu ändern oder den Reset-Pin mit anderen Funktionen zu belegen. In solchen Fällen lässt sich der Prozessor dann nicht mehr über ISP oder Bootloader programmieren.

Ein- und Ausgänge

Digitale Eingänge liefern beim Lesen den Logik-Pegel zurück, der aktuell an diesem Pin anliegt. Der Logik-Pegel kennt nur zwei Zustände: an oder aus, `HIGH` oder `LOW` bzw. `1` oder `0`. Bei 5 V Betriebsspannung gilt eine Spannung von 1,7 V oder weniger als `LOW`-Pegel, eine Spannung von 3,5 V oder höher als `HIGH`-Pegel. In der Praxis werden aber auch schon Spannungen von 3,3 V und leicht darunter als `HIGH` erkannt. Negative Spannungen oder Spannungen über dem Betriebsspannungspegel von 5 V können zur Zerstörung des Eingangs führen. In der Praxis werden digitale Eingänge zur Abfrage von Sensoren bzw. Tastern verwendet.

Nach einem Reset bzw. nach dem Einschalten sind alle Pins zunächst als Eingang konfiguriert. Erst durch das gestartete Programm können Pins als Ausgang konfiguriert werden und liefern dann ein `LOW`- oder `HIGH`-Signal am Ausgang, je nachdem,

was vom Programmablauf her vorgesehen ist. Analoge Ausgänge dienen zum Ein- bzw. Ausschalten von LEDs oder zur Ansteuerung von Transistoren. Mikrocontroller können nur kleine Lasten wie LEDs direkt schalten, der maximale Strom, den ein Ausgang liefern kann, liegt z. B. beim ATMega328 bei 40 mA. Für Lasten mit größerem Strombedarf, wie Relais oder Motoren, ist eine zusätzliche Treiberstufe erforderlich, z.B ein Leistungstransistor

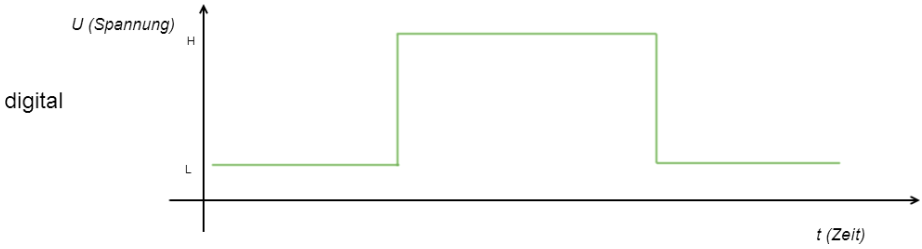


Abbildung 1-9: Digitales Signal

Da ein Mikrocontroller digital arbeitet, kann er mit analogen Signalen nicht viel anfangen. Für diese wird ein spezieller Peripherie-Baustein benötigt, der Analog-Digital-Wandler, kurz ADC (Analog Digital Converter) genannt. Analoge Eingänge liefern beim Lesen einen Digitalwert, der zur anliegenden Spannung äquivalent ist. Dazu wird eine Analog-Digital-Wandlung ausgelöst und das Ergebnis der Wandlung ist der Digitalwert. In der folgenden Abbildung wird ein Digital-Signal dargestellt, wie es der Mikrocontroller "sieht", nicht als kontinuierliches Signal, sondern als "Treppenstufen". Die Abstufung entspricht der Auflösung des A/D-Wandlers. Gängige A/D-Wandler besitzen eine Auflösung von 8 bis 16 Bits, was einer Auflösung von 256 bis 65536 Stufen entspricht.

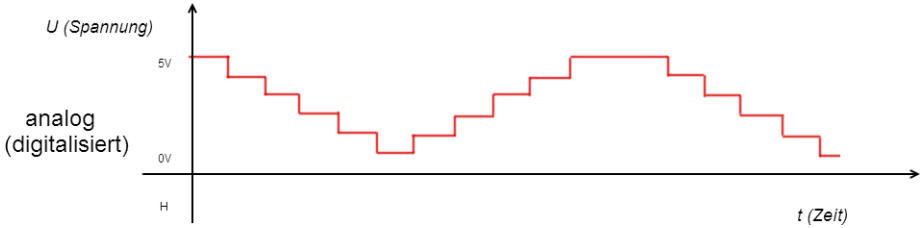


Abbildung 1-10: Analoges Signal

Analoge Ausgänge geben anstelle eines `LOW`- oder `HIGH`-Pegels einen beliebigen Spannungspegel zwischen 0 und der Betriebspannung aus. Um direkt analoge Spannungen ausgeben zu können, wird ein Digital-Analog-Wandler, kurz DAC (Digital Analog Converter) genannt, benötigt. Ein D/A-Wandler ist quasi das Pendant zum A/D-Wandler.

Die meisten Mikrocontroller besitzen keinen D/A-Wandler, stattdessen wird eine quasi-analoge Spannung in Form eines *PWM (http://de.wikipedia.org/wiki/Pulsweitenmodulation)*-Signals ausgegeben. PWM ist die Abkürzung für Pulse Width Modulation, auf deutsch Pulsweitenmodulation. Dabei wird ständig die Ausgangsspannung zwischen `HIGH` und `LOW`-Pegel umgeschaltet, wobei die Pulsweite (die Länge der `HIGH`- bzw. `LOW`-Phasen) variiert wird. Der Mittelwert ergibt dann die analoge Ausgangsspannung. Statt Pulsweite werden auch Begriffe wie Pulslänge (die Zeit, die das Signal auf `HIGH`-Pegel bleibt) und Pausenlänge (die Zeit, die das Signal auf `LOW`-Pegel bleibt) verwendet. Das Verhältnis von Puls- zu Pausenlänge wird dann in Prozent angegeben. In der folgenden Abbildung werden drei PWM-Signale mit unterschiedlicher Puls-/Pausenlänge von 25%, 50% und 75% dargestellt.

PWM-Signale werden z. B. zum Ansteuern von Motoren verwendet, um eine stufenlose Geschwindigkeitsregelung zu ermöglichen.

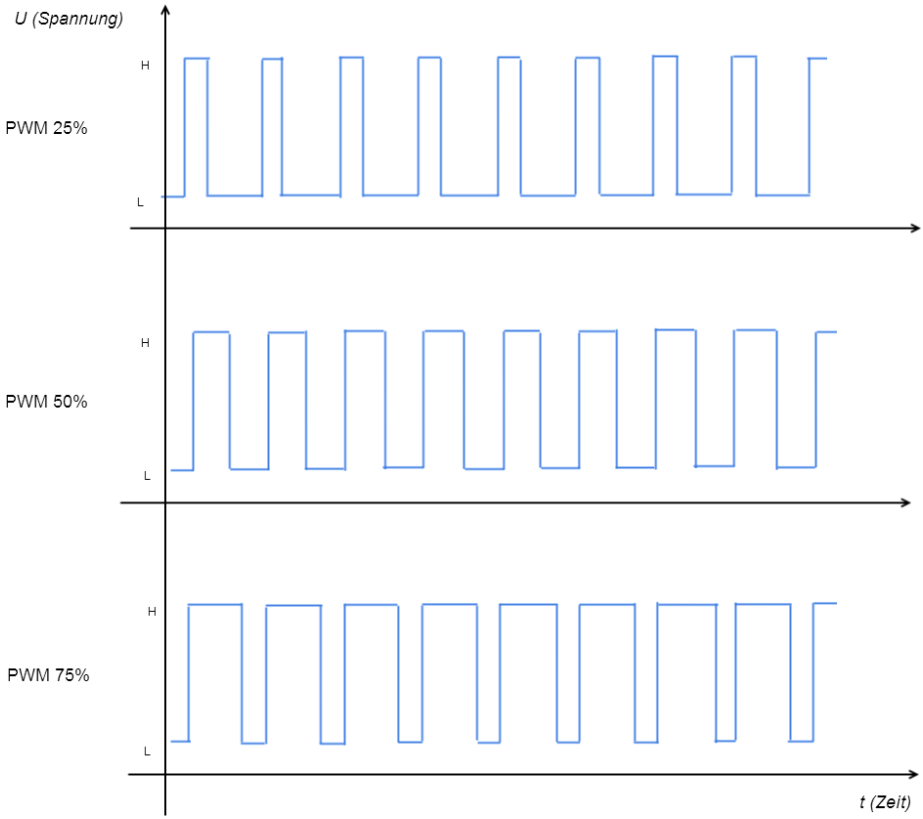


Abbildung 1-11: PWM-Signale

UART-Schnittstelle

Mikrocontroller verfügen über eine Reihe von Standard-Schnittstellen, über die sich Kommunikationswege zu anderen Geräten aufbauen lassen. Die UART- oder serielle Schnittstelle ist dabei die bekannteste und am meisten verwendete.

Beim *UART (http://de.wikipedia.org/wiki/Universal%5FAsynchronous%5FReceiver%5FTransmitter)* (Universal-Asynchron-Receiver-Transmitter), der auch als serielle Schnittstelle bezeichnet wird, handelt es sich um eine asynchrone serielle Punkt-zu-Punkt-Verbindung zwischen zwei Geräten. Asynchron deshalb, weil kein Taktsignal übertragen wird. Der Empfänger muss die Übertragungsrate des Senders kennen und sich anhand des Startbits und der empfangenen Datenbits mit dem Sender synchronisieren.

Zum Verbinden zweier Mikrocontroller über den UART werden mindestens drei Leitungen benötigt. `RX` (Receiver, Empfänger), `TX` (Transmitter, Sender) und `GND` (Masseverbindung). Zu beachten ist, dass die `RX`- und `TX`-Leitungen gekreuzt verdrahtet werden. Eine Verbindung über die serielle Schnittstelle kann zusätzlich noch diverse Steuerleitungen umfassen, wie `RTS` (Ready to Send), `CTS` (Clear to Send), `DTR` (Data Terminal Ready) und `DTS` (Data Set Ready), die wir hier der Einfachheit halber weglassen. Diese Steuerleitungen stammen noch aus Zeiten der Modem-Ära und haben inzwischen an Bedeutung verloren. Allerdings wird beim Arduino eine Steuerleitung (`DTR` oder `RTS`) verwendet, um auf dem Board einen Reset auszulösen, damit der Bootloader angesprungen wird und ein neues Programm hochgeladen werden kann.

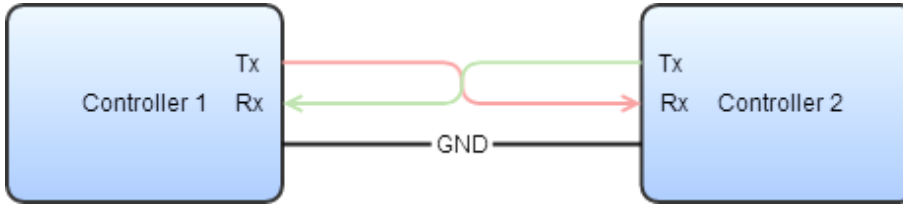
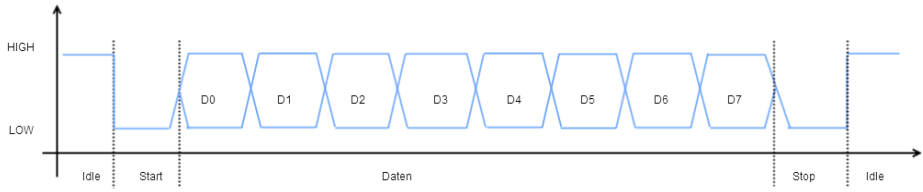


Abbildung 1-12: UART-Verbindung

Die Übertragungsrate für die serielle Schnittstelle, Bits pro Sekunde (Bit/s), Baud oder Baudrate genannt, kann zwischen 300 und 115200 Baud liegen, manchmal auch darüber. Eine gängige Baudrate ist z. B. 9600 Baud. Das bedeutet, es können 9600 Bit/Sekunde übertragen werden. Das ist bei den heutigen Technologien wie DSL/VDSL/LAN sehr, sehr wenig. VDSL erreicht inzwischen Übertragungsraten von bis zu 100 MBit/s ($100*10^6$ Bits/s) bei LAN ist 1 Gbit/s (10^9 Bits/s) Standard. Trotzdem reicht diese niedrige Übertragungsrate für kleinere Nachrichten durchaus aus und der Hardware-Aufwand ist natürlich im Gegensatz zu LAN oder DSL minimal. Die UART-Datenübertragung erfolgt in einem fest vorgegebenen Rahmen. Begonnen wird mit dem Startbit, gefolgt von 7–9 Datenbits (`data` LSB first, niedrigstes Bit zuerst), einem optionalen Paritätsbit (`parity`) für eine Fehlererkennung und schließlich 1–2 Stoppbits (`stop`) für das Paketende. In Kurzform finden sich Bezeich-

nungen wie `9600,n,8,1`. Das bedeutet 9600 Baud, no parity, 8 data, 1 stop bit. Diese Einstellungen müssen beim Sender und Empfänger übereinstimmen, damit eine erfolgreiche Datenübertragung stattfinden kann. Auch der Spannungspegel spielt eine wichtige Rolle. War hier früher in der PC-Technik die RS232-Schnittstelle von großer Bedeutung, die mit Pegel von +/–12 V arbeitete, so wird heute eher der TTL-Pegel (0 bis 5 V) verwendet. Heutige PCs besitzen kaum noch RS232-Schnittstellen, dafür gibt es aber eine Reihe von USB-UART-Seriell-Wandler, die direkt an die UART-Schnittstelle eines Mikrocontrollers angeschlossen werden können.

Der UART dient zum Anschluss serieller Geräte wie GPS-Empfänger oder Bluetooth-Module. Auch für den Anschluss an den PC (über USB-Seriell-Wandler) oder für die Kommunikation zwischen zwei Boards (Arduino und Raspberry Pi) wird die serielle Schnittstelle verwendet.



UART Signal (8 data, no parity, 1 stop bit)

Abbildung 1-13: UART-Signale

I²C-Schnittstelle

I²C (http://de.wikipedia.org/wiki/I2C), Inter-Integrated-Circuit, auch als TWI (Two-Wire-Interface) bezeichnet, ist ein synchroner serieller Bus. Hierüber lassen sich im Gegensatz zum UART auch mehrere Teilnehmer betreiben. Beim I²C handelt es sich um einen *Master/Slave (http://de.wikipedia.org/?title=Master/Slave)*-Bus mit gemeinsamer Daten- und Takt-Leitung (`SDA` und `SCL`). Master/Slave klingt zunächst einmal sehr archaisch, wird aber in der Technik sehr häufig verwendet. Der Master initiiert Datenverbindungen, während der Sklave den Anweisungen des Masters lauscht und nur auf direkte Ansprache des Masters antworten darf. Da es nur eine Datenleitung gibt, ist bei I²C nur Halb-Duplex-Betrieb möglich, also entweder Lesen oder Schreiben. Nur ein Master kann eine Kommunikation anstoßen und Daten an einen Slave senden oder Daten vom Slave lesen. Zu diesem Zweck verfügt jeder Slave über eine eindeutige 7-Bit-Adresse, die innerhalb eines Bus-Systems nur einmal vorhanden sein darf und bei jeder Datenübertragung zuerst vom Master gesendet wird.

Die Datenübertragungsrate beim I²C Bus beträgt 100 kHz oder 400 kHz. Zunächst wird vom I²C-Master die Startsequenz gesendet (die Datenleitung `SDA` wechselt von `HIGH` zu `LOW`, während die Taktleitung `SCL` auf dem `HIGH`-Pegel bleibt). Die Taktleitung wird prinzipiell nur vom Master gesteuert. Der Slave hat aber die Möglichkeit,

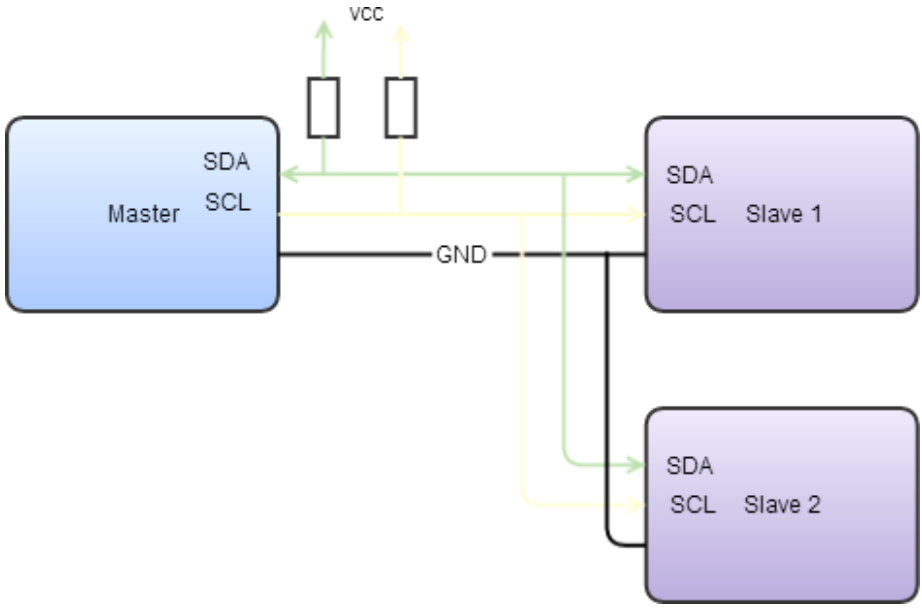


Abbildung 1-14: I²C-Verbindung

das Taktsignal zu verlängern, falls der Takt zu schnell für den Slave ist. Dieser Vorgang wird als Clock-Scretching bezeichnet. Anschließend folgt die 7-Bit-Adresse des Slaves (höchstes Adress-Bit zuerst), jeweils getaktet mit der `SCL` Leitung, gefolgt vom Read/Write-Bit (`LOW` bedeutet Schreibbefehl, `HIGH` Lesebefehl). Der angesprochene Slave antwortet mit einem Acknowlege (`ACK`), indem er die Datenleitung `SDA` für eine Taktlänge auf `LOW` schaltet. Bei einem Schreibbefehl beginnt der I²C-Master damit, die Daten zum I²C -Slave zu senden, der jedes übertragene Byte mit einem Acknowledge bestätigt. Umgekehrt sendet der Slave die Daten und der Master quittiert mit einem Acknowledge bei einem Lesebefehl. Abgeschlossen wird die Übertragung mit einer Stoppsequenz (Datenleitung `SDA` wechselt von `LOW` zu `HIGH`, während die Taktleitung `SCL` auf dem `HIGH`-Pegel bleibt).

Gängige Anwendungsbereiche der I²C-Schnittstelle sind die Anbindung von A/D- und D/A-Wandlern, Sensoren sowie digitalen Port-Erweiterungen.

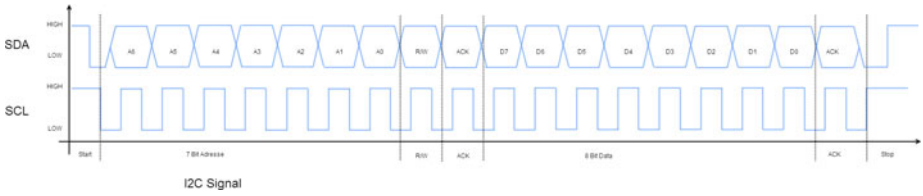


Abbildung 1-15: I²C-Signal

SPI-Schnittstelle

Beim *SPI (http://de.wikipedia.org/wiki/Serial_Peripheral_Interface)*, Serial-Peripheral-Interface, handelt es sich wie bei I^2C um einen seriellen synchronen Bus nach dem *Master/Slave (http://de.wikipedia.org/?title=Master/Slave)*-Prinzip. Im Gegensatz zu I^2C verfügt SPI über je eine eigene Datenleitung zum Lesen (`MISO`, Master-In-Slave-Out) bzw. Schreiben (`MOSI`, Master-Out-Slave-In). Der Empfänger wird durch eine extra Leitung – `SS` (Slave Select) oder `CS` (Chip Select) – ausgewählt. Die Geschwindigkeit bei SPI liegt deutlich höher als bei UART und I^2C. Für einen Arduino mit 16 MHz Quarz sollte der Takt für den SPI-Bus maximal 4 MHz betragen. Die Übertragung einer Slave-Adresse wie beim I^2C-Bus entfällt. Außerdem sind Lesen und Schreiben gleichzeitig möglich. Gängige Anwendungen für SPI sind A/D- und D/A-Wandler sowie EEPROMs. Des Weiteren können auch SD-Karten über SPI angesprochen werden. SPI steht in unzähligen Varianten zur Verfügung, die bekanntesten tragen die Bezeichnung Mode0 bis Mode4. Sie unterscheiden sich in der Art und Weise, wie sich Takt und Daten zueinander verhalten, dass heißt, wann die Daten gültig sind.

Für die Verbindung eines Mikrocontroller mit einem SPI-Gerät sind 5 Leitungen erforderlich, `MOSI`, `MISO`, `SCK`, `CS` und `GND`. Jeder weitere Slave benötigt eine weitere CS-Leitung, der Rest wird parallel verdrahtet. Auf dem Bus gibt es jeweils nur eine Verbindung zwischen einem Master und einem Slave, es sind keine gleichzeitigen Verbindungen möglich.

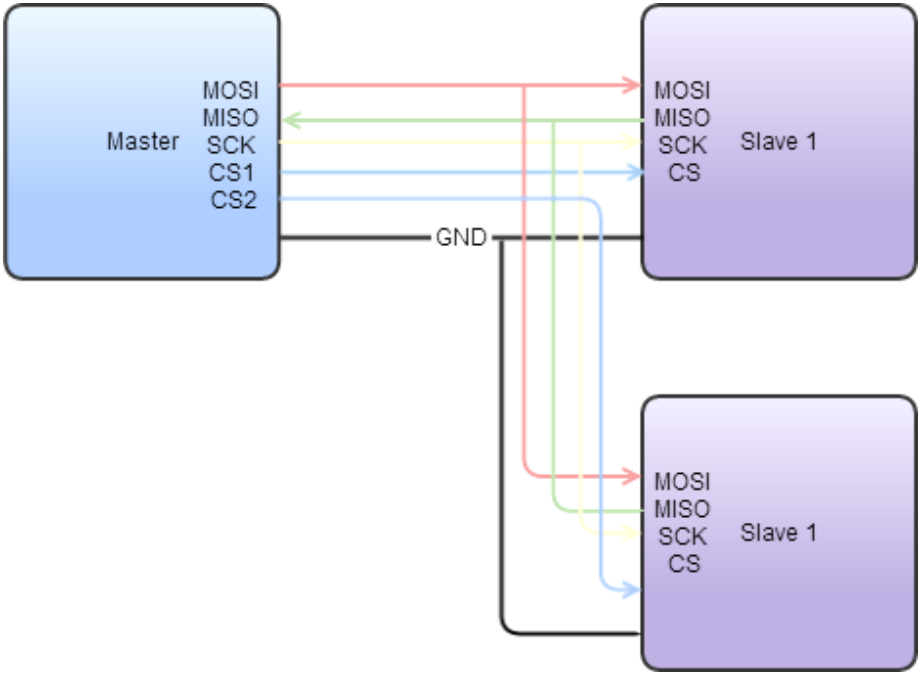


Abbildung 1-16: SPI-Verbindung

Bei der Datenübertragung über SPI gilt, dass jede Übertragung vom Master gesteuert wird, wohingegen der Slave nur auf Anfragen seitens des Masters reagiert. Zunächst zieht der Master die Chip-Select-Leitung (`CS`-Leitung) für den gewünschten Slave nach `LOW`. Zusammen mit jedem Takt werden dann die Daten über die `MOSI`-Leitung an den Slave gesendet. Gleichzeitig sendet der angesprochene Slave seine Daten über die `MISO`-Leitung zum Master.

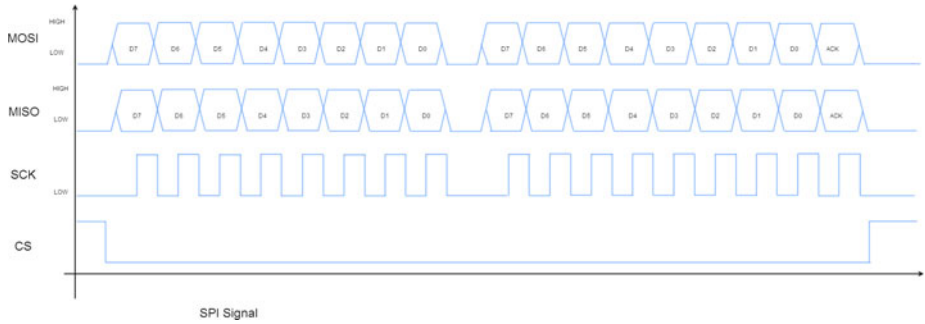


Abbildung 1-17: SPI-Signale

HACK 02 Grundlagen der C/C++-Programmierung

Wie ein Programm auf den Controller übertragen wird, wissen wir nun. Aber wie schreibt man denn ein solches Programm? Beginnen wir mit der *Programmiersprache C (http://de.wikipedia.org/wiki/C_(Programmiersprache))*, die zwar schon ein wenig betagter ist, aber im Embedded-Bereich immer noch als die Nummer 1 gilt. Die Programmiersprache C wurde in den 1970er von Dennis Richie in den Bell Laboratories für das Betriebssystem Unix entwickelt. Auch heute noch wird der Linux-Kernel in C programmiert.

Was wird benötigt?

Um Programme oder besser gesagt den Quelltext oder Source-Code eines Programms schreiben zu können, genügt zwar ein einfacher Text-Editor wie Notepad, besser ist aber ein Text-Editor, der *Syntaxhervorhebung (http://de.wikipedia.org/wiki/Syntaxhervorhebung)* beherrscht. Dazu zählen *Vim (http://www.vim.org/index.php)* (für alle Plattformen verfügbar) und *Notepad-Plus-Plus (http://notepad-plus-plus.org/)* (nur für Windows).

Der *Quelltext (http://de.wikipedia.org/wiki/Quelltext)* beinhaltet das Programm in einer für Menschen lesbaren Form. Dazu gehören die Funktionen, die Variablen und die entsprechenden Deklarationen. Damit das C-Programm ausgeführt werden kann, ist eine `main`-Funktion erforderlich, die beim Programmstart als Erstes ausgeführt wird. Quelltext- Dateien in der Programmiersprache C weisen die Dateiendung `.c` auf.

```
#include <stdio.h>

void main(void)
{
  printf("Hallo Welt!");
}
```

Das "Hallo Welt"-Beispiel in der Programmiersprache C ist das einfachste lauffähige Programm. Es gibt die Zeichenkette "Hallo Welt!" auf dem Bildschirm (Konsolenfenster) aus. Die Funktion **`printf()`** ist Teil der Standard-Bibliothek von C und in der Header-Datei **`stdio.h`** definiert.

Falls der Quelltext auf mehrere Module, d. h. mehrere Dateien, aufgeteilt wird, sind bestimmte Informationen erforderlich, damit die Module sich untereinander verstehen. Diese Informationen, zu denen Funktionsprototypen, gemeinsam genutzte Variablen, Strukturen und Definitionen zählen, werden in *Header-Dateien (http://de.wikipedia.org/wiki/Header-Datei)* abgelegt. Header-Dateien in der Programmiersprache C weisen die Dateiendung **`.h`** auf. Eine Header-Datei wird in einer Quelltext-Datei mit der Präprozessor-Anweisung **`#include`** eingebunden.

```
#include <stdio.h>
```

Damit aus dem Programm-Quellcode ein lauffähiges Programm wird, ist noch eine Compiler- Suite erforderlich. Bei Linux-Systemen ist *GCC (https://gcc.gnu.org/)*, die Gnu Compiler Collection, bereits im Lieferumfang enthalten. Für Windows steht z. B. mit *Dev-C-plus-plus (http://www.bloodshed.net/devcpp.html)* eine komplette Entwicklungsumgebung mit Editor und Compiler-Suite zur Verfügung. Die C/C++-Programme in diesem Buch beschränken sich auf Arduino-Boards. Mit der Arduino-IDE ist hier aber auch eine brauchbare Entwicklungsumgebung für alle Plattformen (Linux, Mac OS X, Windows) gegeben.

Wie wird nun aus dem von uns geschriebenen Programmcode ein ausführbares Programm? Drei Dinge braucht der Programmierer: einen Präprozessor, einen Compiler und einen Linker.

Präprozessor

Der *Präprozessor (http://de.wikipedia.org/wiki/C-Pr%C3%A4prozessor)* führt zunächst alle Präprozessor-Anweisungen aus und schreibt alles in die Ausgabedatei für den Compiler. Präprozessor-Anweisungen beginnen mit einem **`#`**. Zu ihnen zählen folgende Aktionen:

- Das Einbinden von Header-Dateien mit der Anweisung **`#include`**
- Das Ersetzen von Konstanten und Einfügen von Makros mit der Anweisung **`#define`**
- Bedingtes Ersetzung mit den Anweisungen **`#if`**, **`#ifdef`**, **`#ifndef`**, **`#else`**, **`#elif`** und **`#endif`**

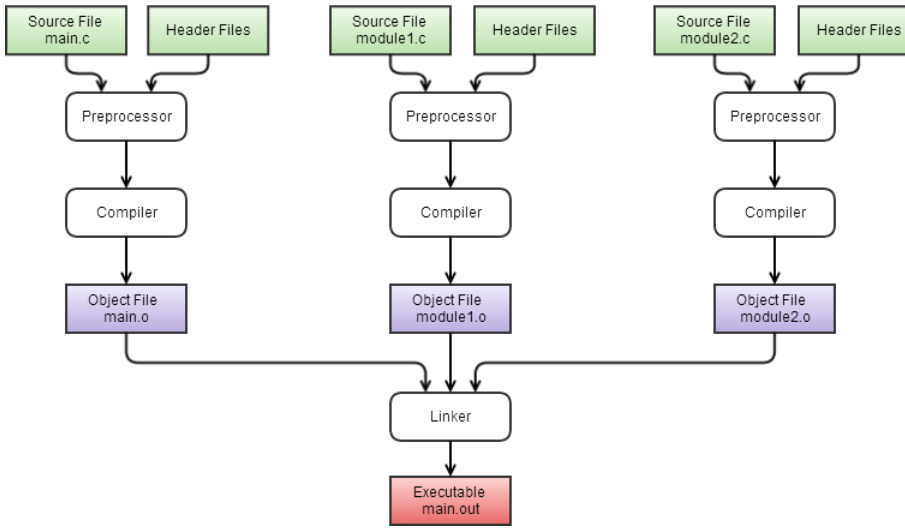


Abbildung 1-18: Compiler, Präprozessor, Linker

```
#include <stdio.h>

#define _DEBUG

void main(void)
{
#ifdef _DEBUG
  printf("Debugging is on");
#else
  printf("Debugging is off");
#endif
}
```

Das leicht modifizierte "Hallo Welt"-Programm mit einigen Präprozessor-Anweisungen. Zunächst wird mit `#include` die `stdio.h` Header-Datei eingebunden. Mit `#define` wird das Makro `_DEBUG` definiert, das für die `#ifdef`-Abfrage verwendet wird. Da `_DEBUG` definiert ist, lautet die Ausgabe "Debugging is on". Der `#else`-Zweig wird gar nicht in die Ausgabedatei für den Compiler übernommen.

Compiler

Der *Compiler (http://de.wikipedia.org/wiki/Compiler)* übersetzt jede Quelltext-Datei einzeln in einen Objekt-Code. Der Compiler übernimmt auch die Aufgabe der Optimierung und die Überprüfung der Programmsyntax. Im Falle eines Fehlers wird der Compiler-Vorgang mit der Angabe der fehlerhaften Zeilennummer im Quelltext abgebrochen. Häufigste Fehler sind vergessene `;` am Ende einer Anweisung sowie fehlende oder zu viele Klammern.

Linker

Der Linker fügt alle Objektdateien des Compilers zusammen, packt zusätzlich notwendige Laufzeit-Bibliotheken mit dazu und erzeugt schließlich das ausführbare Programm.

Zahlensysteme

Aus dem Bereich der Digitaltechnik wissen wir, dass es nur 2 Signalpegel gibt, `0` oder `1` bzw. `LOW` oder `HIGH`. Deshalb rechnen auch Computer digital mit Nullen und Einsen. Die Grundlage hierfür bildet das *binäre Zahlensystem (http://de.wikipedia.org/wiki/Dualsystem)* auch Dualsystem genannt. Das hat übrigens nichts mit dem Dualen System und dem grünen Punkt zu tun. Während beim bekannten Dezimalsystem die Zahlen 0 bis 9 verwendet werden, mittels derer sich auch große Zahlen darstellen lassen, stehen beim Dualsystem nur die beiden Zahlen 0 bis 1 zur Verfügung. Entsprechend mehr Stellen werden benötigt, um größere Zahlen darzustellen. So sind z. B. für die Dezimalzahlen von 0 bis 15 vier Binärstellen oder vier Bits, wie die Binärstellen auch gerne bezeichnet werden, erforderlich. Für die Dezimalzahlen 0 bis 255 werden 8 Binärstellen (Bits), die dann ein Byte ergeben, benötigt.

Weil das Schreiben von solch langen Zahlen zu mühselig ist, wurde das *Hexadezimalsystem (http://de.wikipedia.org/wiki/Hexadezimalsystem)* erfunden. Hier stehen die 10 Zahlen 0 bis 9 und die Zeichen A bis F zur Verfügung. Es wird dabei nur noch eine Stelle für die Dezimalzahlen 0 bis 15 benötigt. Für die Dezimalzahlen 0 bis 255 sind nur 2 Hexadezimalstellen erforderlich.

Hier noch einige Anmerkungen zu Bits und Bytes: Um in Programmen die verschiedenen Zahlensysteme unterscheiden zu können, werden Präfixe verwendet. Diese teilen dem Compiler mit, wie er eine Zahl zu interpretieren hat. So steht `0x` für Hexadezimalzahlen und `0b` für Binärzahlen. Dezimalzahlen haben keinen Präfix.

In der folgende Tabelle werden in verkürzter Form die Dezimalzahlen 0 bis 255 in den verschiedenen Zahlensystemen dargestellt.

Tabelle 1-1: Vergleich der Zahlensysteme

Dezimal	Binär	Hexadezimal
0	0b00000000	0x00
1	0b00000001	0x01
2	0b00000010	0x02
3	0b00000011	0x03
4	0b00000100	0x04
5	0b00000101	0x05
6	0b00000110	0x06

7	0b00000111	0x07
8	0b00001000	0x08
9	0b00001001	0x09
10	0b00001010	0x0A
11	0b00001011	0x0B
12	0b00001100	0x0C
13	0b00001101	0x0D
14	0b00001110	0x0E
15	0b00001111	0x0F
		..
250	0b11111010	0xFA
251	0b11111011	0xFB
252	0b11111100	0xFC
253	0b11111101	0xFD
254	0b11111110	0xFE
255	0b11111111	0xFF

Konstanten

Konstanten sind wie der Name schon sagt zur Laufzeit eines Programmes fest vordefiniert und können zur Laufzeit nicht mehr geändert werden.

Bei Defines handelt es sich um Präprozessor-Anweisungen. Sie sind durch ein Doppelkreuz **#** gekennzeichnet. Eigentlich versteckt sich dahinter nur das Ersetzen von Zahlen durch einen einprägsamen Namen. Wenn der Name mehrmals in einem Programm verwendet wird, muss nur die Zahl in der **#define**-Zeile geändert werden. Es ist nicht erforderlich, alle entsprechenden Stellen innerhalb des Programms zu modifizieren.

```
#define   MOD_INIT 0
#define   MOD_IDLE 1
#define   MOD_RUN  2
#define   MOD_EXIT 3
```

Aufzählungen haben einen eingeschränkten Wertebereich, der in der Deklaration festgelegt wird. Aufzählungen finden Verwendung bei Eigenschaften, Betriebsmodi etc. Auch diese Werte stehen damit zur Laufzeit fest und sind damit konstant wie **#define**. Aufzählungen haben aber gegenüber **#define** einige Vorteile:

- Werte innerhalb des `enum`-Blocks werden automatisch hochgezählt.
- `enum`-Variablen können nur Werte aus dem eingeschränkter Wertebereich annehmen.

```
enum eMode {
  MOD_INIT,
  MOD_IDLE,
  MOD_RUN,
  MOD_EXIT
};
```

Das `enum`-Beispiel entspricht exakt dem vorherigen Beispiel mit `#define`.

Variablen

Das Gegenstück zu den Konstanten bilden die Variablen. Bei Variablen handelt es sich einfach ausgedrückt nur um Behälter für Daten, die Speicherplatz im Arbeitsspeicher bzw. RAM eines Prozessors belegen. Variablen unterscheiden sich nach ihrem Sichtbarkeitsbereich, so gibt es globale und lokale Variablen.

Globale Variablen werden außerhalb von Funktionen deklariert, gelten für die gesamte Laufzeit des Programms und können innerhalb aller Funktionen des Programms gelesen und geändert werden. Das Arbeiten mit globalen Variablen ist einerseits sehr bequem, stellt aber andererseits aber auch eine große Fehlerquelle dar, denn es geht schnell der Überblick darüber verloren, in welchen Funktionen die Variable geändert wird.

Lokale Variablen werden innerhalb einer Funktion deklariert und gelten damit nur innerhalb dieser Funktion. Auch der Wert der Variablen ist nach dem Verlassen der Funktion unbestimmt. Lokale Variablen finden in folgenden Bereichen Anwendung:

- bei Schleifenzählern
- bei nicht weiter benötigen Zwischenergebnisse
- bei Rückgabewerten von Funktionen

Zusätzlich gibt es noch einige Schlüsselwörter, mittels derer Variablen mit speziellen Eigenschaften versehen werden können.

Soll der Wert der Variablen nach Verlassen bzw. erneutem Aufrufen der Funktion erhalten bleiben, so muss die Variable mit dem Schlüsselwort `static` deklariert werden. Diese Variablen werden als statische Variablen bezeichnet. Sie sind innerhalb eines Moduls bzw. einer Quelldatei gültig.

Volatile Variablen stellen einen absoluten Sonderfall in der Programmierung dar, haben aber durchaus ihre Berechtigung. Es handelt sich dabei um Variablen, die ihren Wert ändern können, während eine Funktion durchlaufen wird, ohne das explizit in der Funktion eine Zuweisung erfolgt. Das klingt seltsam, ist aber so, denn Variablen dieses Typs können in einer Interrupt-Funktion verwendet und dort geändert werden. Ein Interrupt kann zu einem beliebigen Zeitpunkt auftreten und den Programmdurchlauf unterbrechen. Um diese Ausnahmen dem Compiler kenntlich zu machen, wird

das Schlüsselwort `volatile` verwendet. Für den Compiler bedeutet dies, dass er die betreffende Variable nicht in einem Register puffert, sondern sie bei jedem Lesevorgang immer wieder aus dem RAM lädt.

Datentypen

Durch Datentypen wird der Wertebereich von Variablen festgelegt. Werte vom Datentyp `signed` sind vorzeichenbehaftet, während `unsigned`-Datentypen vorzeichenlos sind und nur positive Zahlen darstellen können.

Die folgende Tabelle zeigt die wichtigsten Datentypen, deren Bitbreite und den Wertebereich.

Tabelle 1-2: Vergleich der Datentypen

Datentyp	Bits	Wertebereich
bool	8	0,1
char	8	—128 bis 127
signed char	8	—128 bis 127
unsigned char	8	0 bis 255
int	16 oder 32	—32768 bis 32767 bei 16 Bit
signed int	16 oder 32	—32768 bis 32767 bei 16 Bit
unsigned int	16 oder 32	0 bis 65535 bei 16 Bit
short int	16	—32768 bis 32767
signed short int	16	—32768 bis 32767
unsigned short int	16	0 bis 65535
long int	32	—2147483648 bis 2147483647
signed long int	32	—2147483648 bis 2147483647
unsigned long int	32	0 bis 4294967295
float	32	10^{-38} bis 10^{38}, 7 Nachkommastellen
double	32 oder 64	10^{-308} bis 10^{308}, 16 Nachkommastellen

Die Bitbreite bei `int` und `double` ist abhängig von der Prozessor-Hardware.

Schauen wir uns am besten ein paar einfache Beispiele zur Deklaration von Datentypen an:

```
bool flag = true;       /* boolsche Variable mit dem Wert 'true' = 1 */
char c = 'A';           /* eine Zeichenvariable mit dem Wert 'A' */
char c2 = 41;           /* ebenfalls eine Zeichenvariable mit dem Wert 41, */
                        /* entspricht ebenfalls 'A' */
```

```
char newline = '/n';    /* ein Sonderzeichen mit dem Wert 10, */
                        /* entspricht Line Feed */
char zerochar = '/0';   /* ein Sonderzeichen mit dem Wert 0 */
int i = 5;              /* eine Integer-Variable mit dem Wert 5 */
long l = 1L;            /* eine Variable vom Typ Long Integer */
                        /* mit dem Wert 1 */
double d = 3.14;        /* ein Double-Variable mit dem Wert 3,14 */
```

Ein paar Besonderheiten fallen dabei ins Auge:

Variablen vom Typ `bool` werden für Wahrheitswerte verwendet und können nur zwei Zustände einnehmen: *True*, also wahr, oder *false*, d. h. unwahr.

Variablen vom Typ `char` werden für Text- bzw. *ASCII-Zeichen (http://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange)* verwendet. Die ASCII-Tabelle besteht aus 7 Bit langen Werten, mittels derer die häufigsten Buchstaben, Zahlen und Sonderzeichen dargestellt werden. So hat z. B. der Buchstabe `A` den Hexadezimal Wert 0x41. Solche `char`-Variablen lassen sich auf mehrere Arten initialisieren:

- Textzeichen werden mit einfachen Hochkommata initialisiert, z. B. `a` oder `B`.
- Sonderzeichen werden mit einfachen Hochkommata und einem vorangestellten Schrägstrich `/` sowie einem Symbolzeichen für das Sonderzeichen initialisiert, z. B. `/r` für Carriage Return (0x0C) oder `/n` für New Line bzw. Line Feed (0x0A).
- Eine andere Möglichkeit besteht in der Verwendung von Hochkommata, Schrägstrich und einer Zahl, z. B. `/13` oder `/0`.
- Eine weitere Alternative ist der direkte Weg durch eine Zahl im zulässigen Bereich.

Bei der Zuweisung der `long`-Variable fällt das `L` hinter der Zahl 1 auf. Es steht für `Long` und teilt dem Compiler mit, dass hier auf jeden Fall auch mit `long`-Variablen zu rechnen ist, auch wenn die 1 locker in einem `int` Platz hätte. Zuweisungen zu `double`-Variablen sind durch den Dezimalpunkt gekennzeichnet.

Variablen müssen nicht immer initialisiert werden. Das kann auch später zur Programmlaufzeit erfolgen. Aber Vorsicht: Wenn eine Variable gelesen wird, bevor sie initialisiert wurde, kann das unerwünschte Folgen haben – sie kann möglicherweise zufällige Werte aufweisen Normalerweise sollte der Compiler beim Übersetzen des Programms eine Warnung ausgeben, aber verlassen kann man sich darauf nicht, da solche Warnungen nämlich über entsprechende Compiler-Schalter auch abgeschaltet werden können.

Abgeleitete Datentypen

Aus den Basis-Datentypen lassen sich weitere Datentypen ableiten. Dazu zählen Arrays, Strukturen, Funktionen und Zeiger.

Arrays

Auf diese Weise lassen sich gleiche Datentypen in Feldern zusammenpacken. Ein solches Feld wird als *Array (http://de.wikipedia.org/wiki/Feld_(Datentyp))* bezeichnet. Bei der Initialisierung eines Arrays werden die geschweiften Klammern `{}` verwendet. Die einzelnen Werte werden durch Kommata separiert.

```
int myArray[5] = { 0, 2, 4, 6, 8 };
```

Die Elemente eines Arrays können auch zur Laufzeit einzeln initialisiert werden. Im Prinzip kommt dabei dieselbe Initialisierung wie vorher zum Tragen. Beachten Sie den Indexzähler, mit der die einzelnen Elemente adressiert werden. Dieser beginnt bei 0, dem 1. Element, und endet hier bei 4, dem 5. Element des Arrays.

```
int myArray[5];

myArray[0] = 0;
myArray[1] = 2;
myArray[2] = 4;
myArray[3] = 6;
myArray[4] = 8;
```

Arrays von `char`-Variablen werden als Zeichenketten oder englisch Strings bezeichnet. Hierin lassen sich Wörter und Texte speichern. Strings werden durch doppelte Hochkommata initialisiert. Eine Besonderheit besteht dabei darin, dass der Compiler an diese Zeichenkette noch ein Null-Zeichen `/0` anhängt, wobei es sich um das Zeichen für das String-Ende handelt. Auf eine Angabe der Größe des String-Arrays kann hierbei verzichtet werden. Diese berechnet der Compiler automatisch.

```
char myString[] = "Hallo Welt";
```

Wenn aber eine Größe angegeben wird, muss diese auch korrekt sein. Folgende Anweisung erzeugt einen Fehler beim Übersetzen, weil das Ende-Zeichen nicht mitgezählt wurde, 11 wäre die korrekte Länge.

```
char myString[10] = "Hallo Welt";   /* erzeugt eine Fehlermeldung */
```

Strukturen

Für das Gruppieren beliebiger Datentypen zu gruppieren stehen unter C entsprechende Strukturen zur Verfügung – das Schlüsselwort `struct` gefolgt von den einzelnen Struktur-Elementen in geschweifter Klammer:

```
struct myStruct {
    int id;
    unsigned short cmd;
    unsigned short len;
    char buffer[64];
    unsigned short crc;
};
```

Strukturen werden häufig eingesetzt, um Nachrichten zwischen verschiedenen Systemen seriell oder per Netzwerk zu versenden. Beide Seiten kennen die Struktur und deren Elemente, um die Nachrichten verarbeiten zu können. Auch hier ergeben sich Fehlerquellen in puncto Programmierfehler. Unterschiedliche Systeme können auch bedeuten, dass `int` oder `short` auf den verschiedenen Systemen unterschiedliche Bitbreiten oder eine andere Byte-Reihenfolge besitzen.

Byte-Reihenfolge? Was hat es denn damit auf sich? Bei der *Byte-Reihenfolge (http://de.wikipedia.org/wiki/Byte-Reihenfolge)*, englisch Byte-Order oder Endianess, wird festgelegt, wie einzelne Bytes von Integer-Werten im Arbeitsspeicher abgelegt werden. Dabei wird folgende Unterscheidung getroffen:

- Big Endian – hier wird das höchstwertige Byte zuerst gespeichert, dann die niedrigstwertigen Bytes.
- Little Endian – hier wird in umgekehrter Reihenfolge gespeichert, das niedrigstwertige Byte auf der Startadresse.

Tabelle 1-3: Vergleich Byte-Reihenfolge

Datentyp	Little Endian	Big Endian
unsigned short 0x0123	0x23, 0x01	0x01, 0x23
unsigned int 0x01234567	0x67, 0x45, 0x23, 0x01	0x01, 0x23, 0x45, 0x67

Wenn Nachrichten zwischen Little- und Big-Endian-Maschinen ausgetauscht werden, muss die Byte-Reihenfolge beachtet werden. Auch das zu übertragende Medium ist wichtig, so wird bei Netzwerkverbindungen praktisch immer in Big Endian übertragen, während Intel-PCs oder auch der Arduino mit Little Endian arbeiten. Netzwerk-APIs bieten hierfür spezielle Umwandlungsfunktionen an, wie `ntohs()` (network to host short) oder `htons()` (host to network short), um 16-Bit- oder 32-Bit-Werte vor der Übertragung zum Netzwerk umzuwandeln (hton..) bzw. sie nach dem beim Empfang vom Netzwerk wieder zurückzuwandeln (ntoh..).

Funktionen

Über die Umwandlungsfunktionen kommen wir zum nächsten abgeleiteten Datentyp, den *Funktionen (http://de.wikipedia.org/wiki/Funktion_(Programmierung))*. Bei Funktionen handelt es sich um Teile eines Programms, in der häufig verwendete Rechenschritte oder Aktionen gesammelt werden. Statt die betreffenden Rechenschritte endlos im laufenden Programm zu wiederholen, was zu unleserlichem Spaghetti-Code führen würde, werden diese in einer Funktion gesammelt. Die Funktion erhält einen eindeutigen Namen. Es können zusätzlich noch Parameter hinzukommen, die für die Rechenoperation erforderlich sind. Zu guter Letzt gibt die Funktion einen Wert oder das Rechenergebnis zurück, oder eine OK-Meldung bzw. eine Fehlermeldung, je nachdem, ob die Funktion erfolgreich beendet wurde oder nicht. Als einfaches Beispiel muss noch einmal das Ohmsche Gesetz herhalten.

```
/* Funktion zur Spannungsberechnung aus Widerstand und Strom
float calcVoltage(float Resistor, float Current)
{
  float Voltage;

  Voltage = Resistor * Current;
  return (Voltage);
}
```

Um die Funktion an anderer Stelle im Programm aufrufen zu können, müssen der Name und die Parameter der Funktion bekannt sein. Dazu wird ein Prototyp benötigt, der den Namen der Funktion, Rückgabe- und Parameter-Datentypen enthält.

```
float calcVoltage(float Resistor, float Current);
```

Falls kein Rückgabewert oder Parameter erforderlich ist, kann der Dummy-Datentyp `void` verwendet werden.

Wie die Funktion letztendlich aufgerufen werden kann, zeigt das folgende Beispiel. Die Funktionsparameter (`resistor`, `current`) werden mit den gewünschten Werten initialisiert. Der Zuweisungsoperator `=` übergibt den Rückgabewert der Funktion an die Ergebnisvariable (`voltage`)

```
float voltage, resistor, current;

resistor = 100.0;
current = 0.1;
voltage = calcVoltage(resistor, current);
```

Makros

Makros (http://de.wikipedia.org/wiki/Makro) ähneln den Funktionen und lassen sich auch wie solche verwenden. Allerdings beginnen Makros wie Konstanten mit der Präprozessor-Anweisung **#define**, d. h., der Präprozessor ersetzt vor dem Übersetzen des Programms den im Programmcode eingesetzten Makro-Namen mit dem entsprechenden Programmauszug des Makros. Durch die häufige Verwendung von Makros wird das Programm leicht aufgebläht, da der Präprozessor immer den kompletten Auszug einfügt.

```
#define SWAP_INT(a,b) {\
int c; \
c=a; a=b; b=c;\
}
```

Das Makro `SWAP_INT(a,b)` tauscht den Inhalt der Variablen `a` und `b` aus. Ein weiterer Nachteil von Makros soll hier nicht verschwiegen werden: Es findet keine Typ-Überprüfung der Parameter statt. Darin besteht wieder eine häufige Fehlerquelle. Die übergebenen Parameter sollten immer geklammert werden.

Zeiger

Bleibt noch der *Zeiger (http://de.wikipedia.org/wiki/Zeiger_(Informatik))*, englisch Pointer, hinter dem sich eine Speicheradresse auf einen Datentyp verbirgt. Ein Zeiger kann auf Daten oder Funktionen zeigen und sogar auf andere Zeiger.

```
int a = 0xaa;   /* Integer-Variable a */
int *pa = &a;   /* Zeiger auf Integer-Variable a */
```

Das obige Beispiel zeigt einen `int`-Zeiger. Das `*`-Zeichen hinter dem Datentyp gibt an, dass es sich sich um einen Zeiger auf den entsprechenden Datentyp handelt. Die Adresse einer Variablen erhält man mit dem `&`-Zeichen vor dem Variablennamen.

Null Zeiger werden verwendet, um das Ende einer Tabelle von Zeigern darzustellen oder ungültige Einträge zu markieren.

Der `void`-Zeiger ist ein besonderer Zeiger: Er hat keinen bestimmten Datentyp und kann deshalb als Zeiger auf alle Datentypen angewendet werden.

Operatoren

In C stehen zahlreiche Operatoren für Rechen- und Vergleichsoperationen zur Verfügung.

Arithmetische Operatoren

Arithmetische Operationen werden für Rechenoperationen verwendet. Neben den vier Grundrechenarten gibt es noch die speziellen Operatoren Modulo, Inkrement und Dekrement.

Tabelle 1-4: Arithmetische Operatoren

Operator	Beispiel	Beschreibung
+	4 + 5	Addition
-	3 – 2	Subtraktion
*	2 * 3	Multiplikation
/	9 / 3	Division
%	7 % 4	Modulo – Rest der Integer Division
++	i++	Inkrement – erhöhe um 1
--	--i	Dekrement – reduziere um 1

Logische Operatoren

Logische Operatoren werden für Vergleichsoperationen genutzt. Das Ergebnis der Vergleichsoperation ist wahr (true) oder unwahr (false).

Tabelle 1-5: Logische Operatoren

Operator	Beispiel	Beschreibung
==	`i==0`	ist gleich
!=	`i!=0`	ist ungleich
>	`i > 0`	ist größer
>=	`i>=0`	ist größer oder gleich
<	`i < 0`	ist kleiner
<=	`i<=0`	ist kleiner oder gleich
&&	`i && k`	logisch UND
||	`i||k`	logisch ODER
!	`!i`	Negation (NOT)

Bit-Operatoren

Mittels Bit-Operatoren werden bitweise Veränderungen an Operanden vorgenommen.

Tabelle 1-6: Bit-Operatoren

Operator	Beispiel	Beschreibung
&	`i&1`	UND
|	`i|1`	ODER
^	`i^1`	XOR Exclusives ODER
<<	`i<<1`	Schieben nach links
>>	`i>>1`	Schieben nach rechts
~	`~i`	Einerkomplement

Verzweigungen

Zum Steuern des Programmablaufs kommen unter Verwendung der C-Schlüsselworte `if`, `else` und `switch`, `case`-Verzweigungen zum Einsatz.