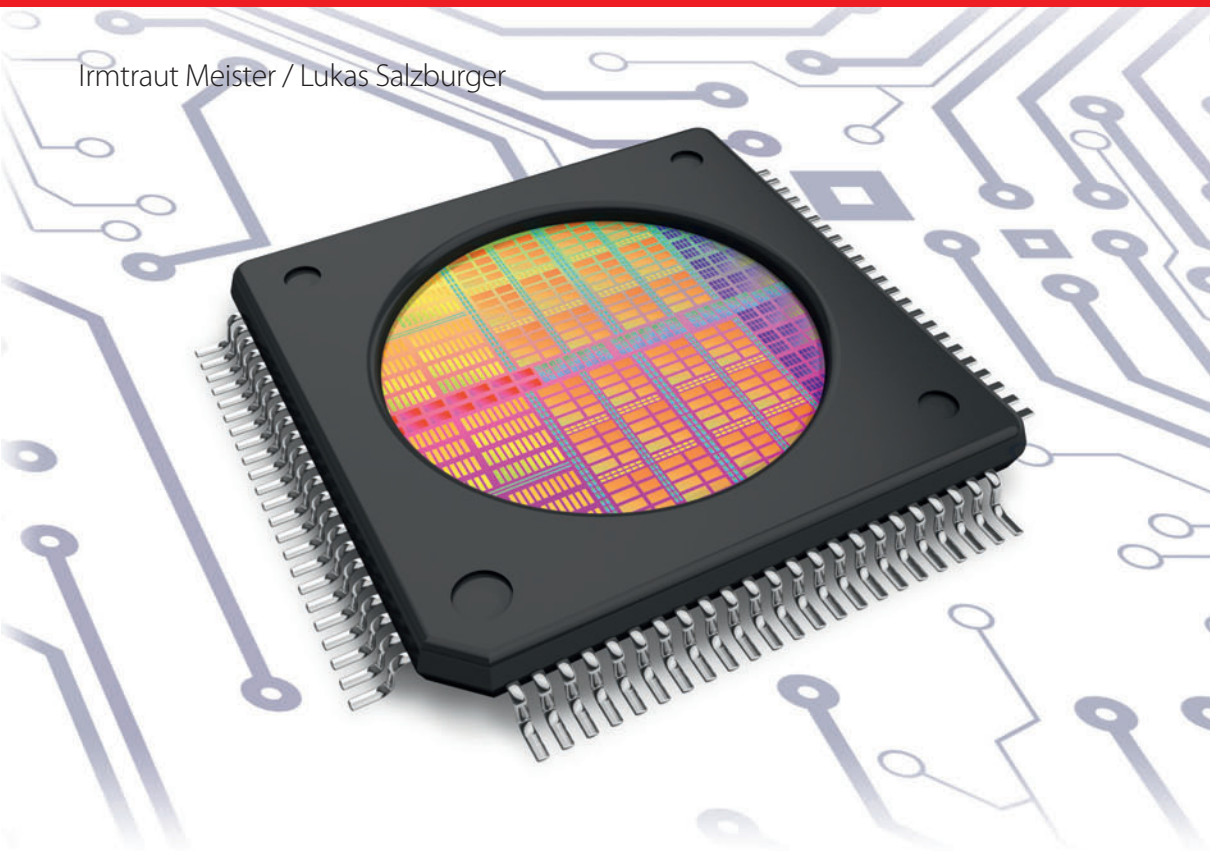
PC & ELEKTRONIK

Irmtraut Meister / Lukas Salzburger



AVR-
Mikrocontroller-Kochbuch

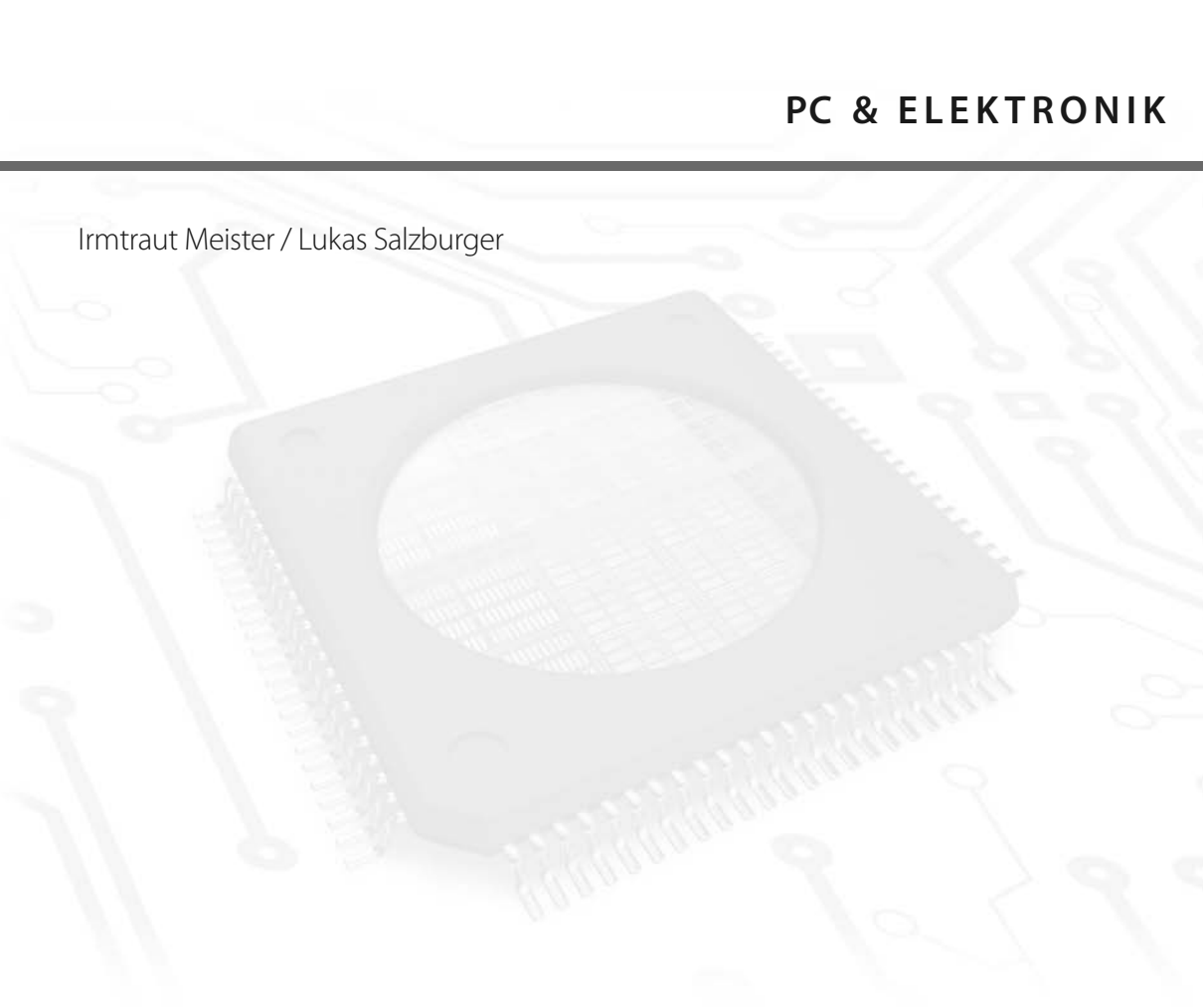
Entwurf und Programmierung praktischer Anwendungen

FRANZIS

Irmtraut Meister / Lukas Salzburger

AVR-Mikrocontroller-Kochbuch

Irmtraut Meister / Lukas Salzburger



AVR-Mikrocontroller-Kochbuch

Entwurf und Programmierung praktischer Anwendungen

FRANZIS

Bibliografische Information der Deutschen Bibliothek

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

Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: C.H. Beck, Nördlingen
Printed in Germany

ISBN 978-3-645-65126-4

Vorwort und »Gebrauchsanweisung«

Ein gutes Kochbuch erkennt man an zwei Dingen: Zum einen verfügt es über einen allgemeinen Teil, der nützliche grundlegende Tipps und Anweisungen rund ums Kochen enthält, ohne deren Kenntnis das tollste Rezept nicht gelingt. Darunter fallen beispielsweise die korrekte Zubereitung der verschiedenen Fleischsorten, die Lagerung von Gemüse oder einige einfache Saucen. Zum anderen sollen die Rezepte selbst von einer Qualität sein, die es erlaubt, aus möglichst einfachen Zutaten ohne großen Aufwand ein möglichst schmackhaftes Ergebnis zu zaubern. Idealerweise gefolgt von möglichen Variationen, falls eine Zutat gerade nicht zur Hand ist.

Die Grundlagen sind zwar vielleicht für erfahrene Hobbyköche eher uninteressant. Aber wenn sie nicht vorhanden sind – und der eine oder andere Hinweis ist mit an Sicherheit grenzender Wahrscheinlichkeit für jemanden neu – so können neue Versuche an Kleinigkeiten scheitern und rasch macht sich Frustration breit.

Nach dieser Logik haben wir den ersten Teil des Buches gestaltet. Die Grundlagen sollen eine gemeinsame Basis schaffen und Einsteigern Konzepte vermitteln, auf denen alles Weitere aufbaut. Auf diese folgen einfache Anwendungen, in welchen die Umsetzung der theoretischen Konzepte an einfachen praktischen Beispielen demonstriert wird.

Den Einstieg bilden also *Mikrocontroller-Grundlegenden (Kapitel 1),* gefolgt von Allgemeinem zur *Programmierung und Implementierung (Kapitel 2),* von Bitoperationen bis zu einfachen Codegerüsten, die den Brückenschlag zwischen den theoretischen Grundlagen und der realen Umsetzung bilden sollen. Letzteres geht auf *Erste Schritte (Kapitel 2.5)* mit einem Mikrocontroller ein, den *Allgemeinen Programmaufbau (Kapitel 2.6)* sowie detailliert auf die allgemeine Implementierung der einzelnen Peripherieeinheiten und *Grundbausteine (Kapitel 2.7).*

Schließlich kommen wir zu den eigentlichen »Rezepten« (Kapitel 3 bis 12), anhand derer wir unsere Mikrocontroller-»Süppchen« nach Lust und Laune kochen können. Hier finden sich Beispiele zum Schalten von LEDs und anderen Lasten, zur Messung verschiedener Größen (Strom, Spannung, Kapazität, Temperatur, Frequenz, …), zum Erzeugen von Signalen oder zur Kommunikation über diverse Schnittstellen – um einige zu nennen.

Wir haben versucht, eventuell auftretende Unklarheiten an Ort und Stelle etwa durch angeführte Beispiele im Text zu beseitigen. Für den Fall, dass trotzdem welche auftreten, bildet ein Anhang mit verschiedenen Hinweisen und Tabellen den Abschluss *(Kapitel 13)*.

Die Autoren möchten sich bei all jenen bedanken, die bei der Entstehung dieses Buches mitgewirkt und es dadurch erst ermöglicht haben.

Abschließend bleibt uns nur noch, dem Leser viel Freude und Erfolg auf dem spannenden Gebiet der Mikrocontrollerprogrammierung zu wünschen, und wir hoffen, mit diesem Buch dazu beizutragen.

Sicherheitshinweis:
Unfälle mit Strom können schmerzhaft oder sogar tödlich sein, daher ist besondere Vorsicht geboten. Bestehende Normen und Sicherheitsrichtlinien zum Umgang mit dem elektrischen Strom sind unbedingt zu konsultieren und zu beachten, folgende Hinweise gelten nur als Richtlinie.
Niedrige Spannungen unter 25 V Wechselspannung beziehungsweise 60 V Gleichspannung gelten als ungefährlich. Unter außerordentlichen Bedingungen kann aber auch eine geringe Wechselspannung unter Anderem den Herzrhythmus stören. Vor allem Mischspannungen (Gleichstrom mit Wechselstromanteil) sind gefährlich.
Zudem gelten viele in der Elektronik eingesetzte Chemikalien als gesundheitsschädigend, insbesondere Blei und Kadmium in Lötzinn. Daher sind diesbezüglich ebenfalls Vorsichtsmaßnahmen zu treffen.

Einführung

Mikrocontroller können als kleine Computer verstanden werden, die im Gegensatz zu ihren großen Verwandten gezielt für spezielle, mehr oder weniger komplexe Aufgaben eingesetzt werden. Sie bilden den Kern von vielen *Eingebetteten Systemen (Embedded Systems)*, welche für die Funktion der meisten modernen elektronischen und mechatronischen Systeme verantwortlich sind, die inzwischen so selbstverständlich zum alltäglichen Leben gehören.

Die möglichen Einsatzszenarien sind so vielseitig, dass es womöglich einfacher wäre, jene modernen technischen Geräte herauszupicken, in denen ausnahmsweise *kein* Mikrocontroller steckt.[1] Von MP3-Playern und Handys über Haushaltsgeräte und Kraftfahrzeugkomponenten bis hin zu Kraftwerken und Industrie-Großanlagen – Mikrocontroller übernehmen eine kaum überschaubare Vielzahl unterschiedlichster Aufgaben. Allein in einem modernen Personenkraftwagen können über 100 Mikrocontroller verbaut sein.

Ein Mikrocontroller ist insofern mit einem *Personal Computer* vergleichbar, als er aus ähnlichen Komponenten aufgebaut ist: Ein Prozessor bildet sein Herzstück, es gibt einen Arbeits- und einen dauerhaften Programmspeicher sowie Zusatzperipherie wie beispielsweise Ein- und Ausgabeeinheiten. Man spricht allerdings erst von einem Mikrocontroller, wenn all diese Bausteine auf einem einzelnen Chip integriert sind.[2] Im Laufe des folgenden Abschnitts *Grundlegende Konzepte* wird genauer auf die Bestandteile und Charakteristiken eingegangen, die für das Verständnis der Funktionsweise, der Möglichkeiten und Grenzen eines Mikrocontrollers essenziell sind.

Zur Erläuterung wurden *AVR®-Mikrocontroller* von *Atmel®* gewählt, da sie weit verbreitet und leicht erhältlich sind. Auch ist der Umgang mit ihnen einfach zu erlernen, da sie einen eleganten und vergleichsweise einfachen Aufbau haben. Die Codebeispiele wurden in der Programmiersprache C geschrieben, um dem Industriestandard gerecht zu werden. Die Grundlagen und Beispiele gelten aber auch für Mikrocontroller anderer Hersteller beziehungsweise sind, mit geringen Abweichungen, auf jene übertragbar. Gerade dieses erste Kapitel ist so ausgelegt, dass es den Leser befähigen soll, sich rasch

---

[1] Neben Mikrocontrollern gibt es beispielsweise noch ASICs, FPGAs und DSPs , welche eine vergleichbare beziehungsweise spezialisiertere Rolle übernehmen können und je nach Anwendung vorzuziehen sind.

[2] Hier gibt es auch Ausnahmen, etwa ältere oder besonders leistungsstarke Mikrocontroller, bei welchen der Speicher auch extern angeschlossen werden kann. Die genannte Definition ist auch nur eine der möglichen.

auch in »fremde« Mikrocontroller einzuarbeiten sowie deren Besonderheiten zu erkennen.

Falls der Leser bereits einen Blick in einschlägige Internetforen geworfen hat, werden ihm womöglich die beinahe religiös anmutenden Diskussionen rund um die Frage auffallen: »Welche Mikrocontroller sind die besten?« Generell kann man diesen Punkt – wie bei den meisten ähnlich lautenden Fragen – nur beantworten mit: »Es kommt darauf an«.

Mikrocontroller werden je nach Anwendung bezüglich Architektur, Speicher und Zusatzfunktionen sowie Kosten, Leistungsverbrauch und Verfügbarkeit ausgewählt (nach Abschluss des Grundlagenkapitels wird das verständlicher). Üblicherweise hat jeder große Hersteller eine Vielzahl vergleichbarer Lösungen parat, wodurch es letztendlich nicht selten auf persönliche Präferenzen hinausläuft, ob man nun einen Mikrocontroller von *Atmel*®, *Freescale*™, *Microchip*, *NXP*, *Renesas*, *Texas Instruments* oder einem anderen »Global Player« wählt.

Dieses Buch setzt Basiskenntnisse in der Programmiersprache C voraus. Zum Erlernen sei auf entsprechende Fachbücher und Online-Tutorials verwiesen. Die Grundlagen werden jedoch auch in einem eigenen Abschnitt kurz wiederholt und um mikrocontrollerspezifische Details erweitert.

Hinweis:
Wir haben viele generell für Mikrocontrolleranwendungen wichtige Maßnahmen an der Stelle beschrieben, an der sie das erste Mal auftauchen beziehungsweise von elementarer Bedeutung sind. Das bedeutet nicht, dass sie nicht auch für andere Anwendungen zu beachten sind – wir benutzen daher viele Querverweise und haben uns bemüht, ein möglichst umfangreiches Stichwortverzeichnis zu pflegen, damit jeder Hinweis und jede Erklärung auffindbar sind.

Inhaltsverzeichnis

1 Mikrocontroller-Grundlagen

1.1 Was ist ein Mikrocontroller?

Man spricht von einem Mikrocontroller (µC, MicroController Unit MCU), wenn außer dem *Mikroprozessor* selbst noch Peripherieeinheiten in einen Chip integriert sind. Dieses erste Kapitel dient dem genaueren Verständnis der grundlegenden Funktionen, Möglichkeiten und Grenzen eines Mikrocontrollers. An dieser Stelle wird vieles noch bewusst vereinfacht dargestellt, auf das in späteren Kapiteln noch genauer eingegangen werden soll.

Das Herzstück einer jeden MCU ist der beinhaltete *Prozessor* oder *Kern (Core),* also die für die Ausführung der Programme zuständige Recheneinheit. Je nach Anzahl der auf einmal verarbeitbaren Bits spricht man von 4-, 8-, 16- oder 32-Bit-Mikrocontrollern. Ein *Bit (Binary Digit)* ist die kleinste Informationseinheit in digitalen Systemen, mit der ein Prozessor auf unterster Ebene rechnet. Es kann entweder den Wert *null* oder *eins* haben[3]. Jeder Befehl eines Programms, jede Form von gespeicherten Daten – also alles, was eine digitale Recheneinheit verarbeiten soll – wird letztendlich in mehr oder weniger komplexe Bitmuster aus Einsen und Nullen umgewandelt, die der Prozessor dann der Reihe nach bewältigen kann.

Zum Betrieb benötigt ein Prozessor außerdem einen *Takt*, also ein Signal bestehend aus periodischen Taktimpulsen (abwechselnd 1 und 0) in einer gewissen Geschwindigkeit. Der Takt kann entweder intern generiert oder extern zugeführt werden. Angegeben wird er in *Hertz* (1 Hz = 1/s), also in Taktimpulsen pro Sekunde.

Da wir aber nur selten mit so niedrigen Frequenzen zu tun haben, findet man eher Angaben in kHz (Kilo-Hertz, also 1.000 Hertz) oder MHz (Mega-Hertz, also 1.000.000 Hertz). Signalen im GHz-Bereich (Giga-Hertz, 1.000.000.000 Hertz) wiederum begegnet man nur in Ausnahmefällen.

---

[3] Wie »null« oder »eins« physikalisch aussehen, dazu gibt es eine Vielzahl möglicher Umsetzungen. Auf einer Signalleitung wird bei »eins« ein festgelegter Spannungspegel über- bzw. bei »null« unterschritten. Sollen Daten gespeichert werden, so hängt es von der Technologie des Speichers ab, auf welche Art jede seiner Speicherzellen den Wert »null« oder »eins« annimmt.

Hinweis:
Im Zusammenhang mit Datenübertragung werden Taktfrequenzen eher in *Baud* (1 Bd = 1/s) angegeben. Ein Baud entspricht also einem Hertz und wird auch gleichermaßen skaliert (1 kBd = 1.000 Bd etc.).
Bei der Angabe von Übertragungsgeschwindigkeiten kann man auch von einer *Baudrate* anstatt einer Taktrate sprechen.

Mehr Hertz bedeuten also mehr Taktimpulse pro Sekunde und daher einen schnelleren Takt und eine größere *Ausführungsgeschwindigkeit*, also mehr Rechenoperationen pro Sekunde.

Die zuvor in Verbindung mit dem Prozessor erwähnte Bit-Zahl sagt nun aus, wieviele Bits ein Prozessor mit einem Taktschritt verarbeiten kann. Arbeitet beispielsweise ein 8-Bit-Mikrocontroller mit 20 MHz (Mega-Hertz), so kann er 20 Millionen Mal pro Sekunde jeweils 8 Bits verarbeiten.[4] Der AVR® ist ein Vertreter der Gattung der 8-Bit-Prozessoren.

Prozessoren, die mehr Bits gleichzeitig verarbeiten können (etwa 32 statt 8), sind also leistungsfähiger. Die Ausführungsgeschwindigkeit des Programms steigt außerdem mit dem Takt, mit dem der Prozessor betrieben wird. Ein 32-Bit-Prozessor ist aber bei gleichem Takt nicht automatisch viermal so schnell wie ein 8-Bit-Prozessor, da viele Faktoren eine Rolle spielen.

Bei der Angabe von Geschwindigkeiten und Stromverbrauch gilt bei vielen Mikrocontrollern die alte Steigerungsregel: Notlüge, Lüge, Benchmark. Nicht dass wir den Herstellern unterstellen würden, gefälschte Werte zu veröffentlichen, es wird aber so stark geschönt und für das eigene Produkt vorteilhaft gemessen, dass ein konkreter Vergleich zwischen den unterschiedlichen Architekturen und Herstellern nur sehr schwer möglich ist.

Aus Kosten- und Komplexitätsgründen wird tendenziell für eine gewisse Anwendung jener Mikrocontroller ausgewählt werden, der die Mindestanforderungen an Prozessorleistung, Speicher und Peripherie gerade noch erfüllt. Warum also einen 32-Bit-Prozessor benutzen, wenn auch ein 8-Bit-Prozessor ausreicht? 8- und 32-Bit-Prozessoren sind inzwischen gebräuchlicher als jene mit 4 oder 16 Bit. Für die meisten einfacheren Anwendungen genügen die tendenziell stromsparenderen 8-Bit-Prozessoren, und falls

[4] Dies ist eigentlich eine grobe Vereinfachung, mit der wir uns aber für den Anfang zufrieden geben. Der extern zugeführte Takt muss zudem nicht mit dem internen Takt übereinstimmen, mit dem der Prozessor tatsächlich arbeitet. Darüber hinaus wird nicht jeder Befehl in genau einem Taktzyklus ausgeführt, manche können etwa bis zu vier Taktzyklen benötigen.

nicht, springt man meistens gleich zu 32-Bit-Prozessoren, weil der Fertigungsaufwand und damit auch der Preis für 16-Bit-Prozessoren kaum geringer ist.

Abgesehen von wenigen Ausnahmen gibt es außerdem keine 8-Bit-Mikrocontroller, die mit mehr als 50 MHz getaktet werden. Der Grund liegt darin, dass es ab einer gewissen Taktrate einfach effizienter ist, einen 32-Bit-Mikrocontroller einzusetzen. Es sei aber auch erwähnt, dass es Mischformen gibt, also z. B. 8-Bit-Controller, die teilweise 16 Bit parallel verarbeiten können.

Ergänzend zum Prozessor enthält ein moderner Mikrocontroller auch zwei verschiedene Speicher: den *Arbeitsspeicher (Random Access Memory, RAM)* und den *Programmspeicher*. Sie unterscheiden sich dadurch, dass der Arbeitsspeicher die darin abgelegte Information bei Unterbrechung der Stromversorgung verliert (man sagt: »er ist *flüchtig*«), während der Programmspeicher sie behält (»er ist *nicht-flüchtig*« oder »*statisch*«). An dieser Stelle ist die Frage berechtigt, wieso nicht ausschließlich nicht-flüchtiger Speicher benützt wird – wozu soll ein Speicher gut sein, der beim »Ausschalten« des Mikrocontrollers seine Daten verliert?

Zur Beantwortung müssen wir etwas ausholen: Man spricht vom *Programmieren* des Mikrocontrollers, wenn ein zuvor erstelltes *Programm* (der Code dafür, was der Mikrocontroller »machen soll«) in den nicht-flüchtigen Programmspeicher geschrieben wird. Dieser kann allerdings technologiebedingt im Laufe seiner Lebenszeit lediglich rund 10.000 bis 100.000 Mal beschrieben werden. Deshalb werden sich häufig ändernde Daten im flüchtigen Arbeitsspeicher zwischengelagert, der (nahezu) unbegrenzt viele Schreibzyklen erlaubt. Hinzu kommt bei schnelleren, komplexeren Mikrocontrollern, dass die höhere Zugriffsgeschwindigkeit auf den Arbeitsspeicher zur Leistungsfähigkeit des Gesamtsystems beiträgt.

Außer Prozessor, Arbeits- und Programmspeichern beinhaltet ein Mikrocontroller eine oder mehrere Zusatzeinheiten und damit -funktionalitäten. Dazu zählen beispielsweise *digitale* und/oder *analoge Ein- und Ausgänge, Timer* und diverse Schnittstellen – dies wird später noch im Detail behandelt.

Vorerst genügt es, zu wissen, dass es diverse Zusatzfunktionen gibt, welche es ermöglichen, mit einem einzelnen Mikrocontroller bereits viele spezialisierte Aufgaben zu lösen, ohne dass eine aufwendige Zusatzbeschaltung oder zusätzliche Chips nötig sind.

Wir fassen zusammen:
Ein Mikrocontroller (µC, MCU) vereint einen Mikroprozessor und einige Peripherie auf einem Chip. Mikrocontroller unterscheiden sich durch:
- die Anzahl der Bits, die ein Controller in einem Taktschritt bearbeiten kann. Es gibt 4-, 8-, 16- und 32-Bit-µC.
- die Größe des internen Arbeits- und Programmspeichers
- die integrierten Zusatzbausteine und -funktionen, welche für die Anforderung benötigt werden; z. B. digitale und analoge Ein- und Ausgänge, Timer, Schnittstellen etc.

Bei der konkreten Auswahl eines Mikrocontrollers müssen diese Eigenschaften und zusätzliche wichtige Punkte wie Bauform, Betriebsspannungsbereich, Lieferbarkeit und natürlich der Preis berücksichtigt werden.

Nachdem wir nun wissen, was einen Mikrocontroller ausmacht, können wir im nächsten Abschnitt genauer ins Detail gehen.

1.2 Grundlegende Konzepte

1.2.1 Die Prozessorarchitektur

Generell unterscheidet man bei Prozessoren zwei verschiedene Architekturen. Die Prozessoren der meisten aktuellen Computer (z. B. der x86-Familie) basieren auf der *Von-Neumann-Architektur*. Sie besteht aus einer *CPU (Central Processing Unit)*, welche den Prozessor und das Steuerwerk umfasst, der *I/O Unit* (für *Input/Output*, also für die Kommunikation mit der Außenwelt) und dem Speicher. CPU und I/O Unit bzw. Speicher kommunizieren über ein gemeinsames Bussystem, über das sowohl Daten als auch Befehle ausgetauscht werden. Programme können problemlos auch im Datenspeicher ausgeführt werden.

Die *Harvard-Architektur* hingegen, welche für eine Vielzahl von Mikrocontrollern einschließlich der AVRs® typisch ist, besitzt zwei getrennte Bussysteme sowie getrennten Speicher für Daten und Programme. Das hat den Vorteil, dass simultan Befehle ausgeführt und Daten ausgelesen oder geschrieben werden können. Der Datenspeicher kann allerdings auch nicht einfach als Programmspeicher genutzt werden.[5]

[5] Zwischen diesen beiden Architekturen kann in der Praxis oft nicht so einfach unterschieden werden, da es inzwischen eine Vielzahl von Mischformen gibt.

Unter dem Familiennamen AVR® fasst Hersteller Atmel® mehrere Mikrocontrollertypen mit 8-, 8/16- oder 32-Bit-Harvard-Architektur zusammen. Wir werden uns im Rahmen dieses Buchs auf die »klassischen« 8-Bit-AVRs beschränken[6]. Dabei sind vor allem zwei Serien für unsere Zwecke interessant:

Die ATtiny-Serie umfasst die günstigeren und teilweise sehr stromsparenden oder mit Sonderfunktionen ausgestatteten Modelle. Sie tragen die Bezeichnung *ATtiny[x][y]* mit der Zahl [x] und eventuell einem Buchstaben [y], welche den konkreten Mikrocontroller genauer spezifizieren (Variante, Leistungsaufnahme, Spezialfunktionen). Beispiele sind der winzige *ATtiny10* oder der *ATtiny261A*.

Die Mikrocontroller der ATmega-Serie verfügen über deutlich mehr Peripherieeinheiten und Speicher, ihr Kern kennt einige Befehle mehr und besitzt eine Hardwaremultiplikationseinheit. Die Bezeichnung folgt demselben Schema, also *ATmega[x][y]* mit der Zahl [x] und dem/den Buchstabe/n [y] gemäß seiner Spezifikation. Ein Vertreter ist der *ATmega48*, der auch im Kapitel *2.5 Erste Schritte – ein einführendes Programm* verwendet wird.

Die Bezeichnung AT90 trugen die ersten Controller mit dem AVR-Kern. Heute wird dieser Name jedoch für »Spezialtypen« mit z. B. USB, CAN oder speziellen PWM-Einheiten verwendet.

Das Namensschema ist ziemlich kompliziert und auch nicht wirklich durchgängig. Wenn man einen geeigneten Mikrocontroller für den jeweiligen Einsatzzweck sucht, verwendet man am besten die – leider nicht besonders durchdachte – parametrische Suche auf der Atmel®-Webseite.

Im Hobbybereich, aber durchaus auch im professionellen Umfeld, sind der ATmega8 und seine »Nachfolger« ATmega48, -88, -168 und -328 in allen Varianten beliebt, vor allem da sie im 28Pin-DIP-Gehäuse erhältlich und recht preiswert sind. Ein weiterer Vorteil ist die Kompatibilität der Baureihe, deren Modelle sich nur im Speicherausbau (und einigen kleinen Details) voneinander unterscheiden. Man kann daher die Entwicklung mit der größten Variante beginnen und das Endprodukt dann nach Möglichkeit mit einer Version mit weniger Speicher (also billiger) ausliefern, ohne die Soft- oder andere Hardware zu ändern.

---

[6] Außerdem gibt es noch die 32-Bit-AVR-UC3-Familie, welche mit den hier behandelten, abgesehen vom Namen, nicht viel gemeinsam hat, und die 8/16-Bit-AVR-XMEGA-Reihe.

Hinweis
Es gibt laufend neue Typen und Modelle in den AVR®-Serien. So ist etwa der *ATmega48A* ein Nachfolger des *ATmega48*, der mit einem neuen Herstellungsprozess gefertigt wurde und zusätzlich zu einigen Detailverbesserungen ab 1,8 V statt 2,7 V betrieben werden kann. Die Varianten *ATmega48P* beziehungsweise *ATmega48PA* sind die zugehörigen stromsparenden Varianten mit einigen Zusatzfunktionen für diesen Einsatzbereich.
Controller wie der *ATmega328* allerdings haben keinen unmittelbaren Vorgänger. Obwohl kein »A« angehängt ist, wird er nach dem gleichen, neuen Herstellungsprozess gefertigt und läuft ab 1,8 V. Man sieht, die Mikrocontrollerbezeichnungen sind eine Wissenschaft für sich, daher sollten immer Hersteller-Homepages und Datenblätter konsultiert werden.

1.2.2 Gehäuse (Package)

Wie die meisten massengefertigten ICs[7] sind AVR®-Mikrocontroller in verschiedenen *Packages*, den Gehäusetypen, erhältlich. Der eigentliche Controller nimmt als *Die* (engl.: Chip, Halbleiterplättchen) nur sehr wenig Platz im finalen Gehäuse ein. Dieses übernimmt gewissermaßen die Aufgabe, den Halbleiterchip gegen äußere Einflüsse zu schützen und den Einbau in eine Schaltung zu ermöglichen.

Um die Funktionen des Chips zugänglich zu machen, muss er *gebondet* werden (eingedeutschter Begriff, von engl. *Bonding*: Binden, Verbindung), also mittels dünner Drähte eine Verbindung zwischen den Kontaktflächen des Halbleiterchips und den Außenkontakten des Gehäuses – den *Pins* – hergestellt werden.[8]

Der Anfänger wird wahrscheinlich als erstes mit AVRs im DIP-Package zu tun haben *(Abb. 1.1)*. DIP steht für *Dual In-line Package* und bezeichnet, wie der Name sagt, einen länglichen Gehäusetyp mit paarweisen Pins an jeder Seite.

---

[7] IC steht für *Integrated Circuit,* bezeichnet also eine in einen einzelnen Baustein integrierte Schaltung.

[8] Mit »Pin« ist also meistens der »physikalische«, äußere Kontakt eines Controllers gemeint. Es muss aber nicht immer jeder Pin am Gehäuse eine elektrische Funktion haben. Es gibt auch nicht verbundene (not connected; NC) Pins, die z. B. nur eine mechanische Funktion haben.

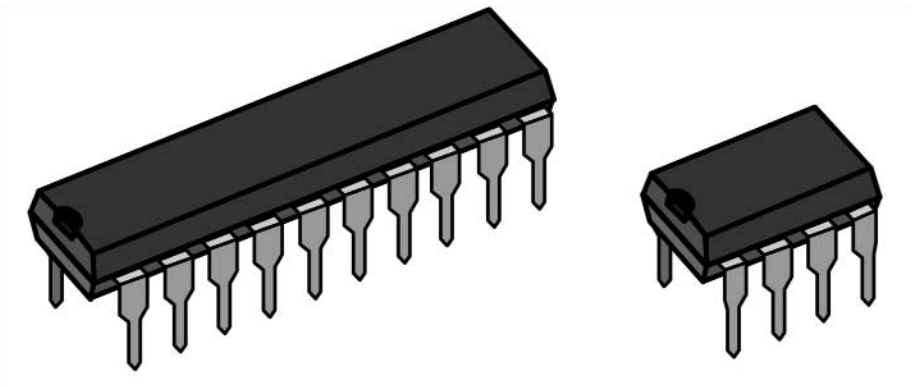


Abb. 1.1: 20- und 8-Pin-DIP-Gehäuse

Das ist der »größte« und »bastlerfreundlichste« Gehäusetyp, die Pins sind verhältnismäßig große, nach unten abgeknickte Kontaktstifte, mit denen der Controller etwa in Steckbretter oder Lochrasterplatinen passt. Bauelemente, die zum Schaltungsaufbau »in Löcher gesteckt« werden können, werden auch als »Through-Hole«-Komponenten bezeichnet. Daher (und weil sie ihrer Größe wegen einfacher zu handhaben und schwerer zu verlieren sind) eignen sich DIP-Gehäuse gut für erste Schaltungsexperimente und Prototypen.

Bausteine im DIP-Gehäuse sind zudem sockelbar, das heißt anstatt des Bausteins selbst kann auch ein Sockel eingelötet werden (*Abb. 1.2*). Der Baustein, beispielsweise der Mikrocontroller, kann daraufhin in den Sockel eingesetzt und wie gewohnt genutzt werden. Wird er beschädigt oder soll er aus einem anderen Grund ausgetauscht werden, so kann er einfach wieder aus dem Sockel gelöst werden, ohne dass aufwendiges und materialbeanspruchendes Löten notwendig ist.

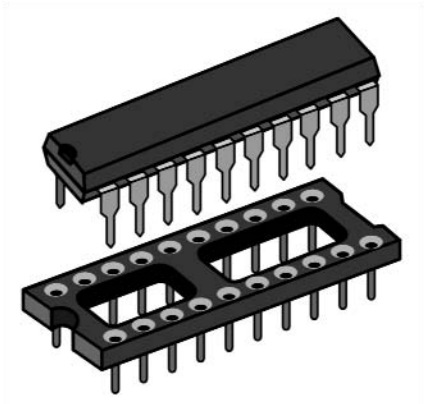


Abb. 1.2: Sockel für DIP-Bauteile

Professionelle Schaltungen beinhalten heutzutage kaum mehr Bauelemente in DIP-Gehäusen. Anders als bei Through-Hole-Komponenten sind bei der Oberflächenmontage (SMT, *Surface Mount Technology*) keine Bohrungen nötig und die Schaltung kann wesentlich kompakter ausgeführt werden. Die hierbei verwendeten Bauteile werden als SMD (*Surface Mounted Device*) bezeichnet. Ihre Anschlusspins sind so gefertigt, dass sie plan auf dafür vorgesehenen *Pads* auf den Platinen zu liegen kommen und an diesen angelötet werden können. Es gibt mehrere Bauformen von SMD-Bauteilen in unterschiedlichen Größen. SO-Gehäuse (*Small Outline*) sind die »größten« SMD-Gehäuse und noch sehr gut per Hand lötbar (*Abb. 1.3*). Ihr Pinabstand ist halb so groß wie bei den DIP-Gehäusen.

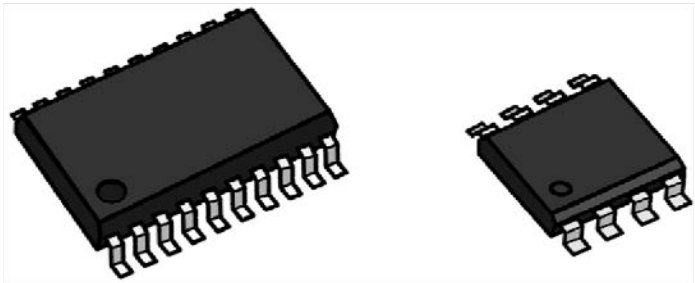


Abb. 1.3: 20- und 8-Pin-SO-Gehäuse

Neben dem geringeren Platzbedarf haben SMD-Komponenten auch elektrisch bessere Eigenschaften als ihre großen Verwandten. So sind beispielsweise Hochfrequenzschaltungen in der Regel mit DIP-Bauteilen undenkbar. Wenn es die Anforderungen hingegen erlauben, spricht nichts dagegen, Entwicklung und Programmierung zunächst mit einem

Controller in DIP-Bauform vorzunehmen und später, falls die Schaltung in Massenfertigung hergestellt werden soll, denselben Controller in einem SO-Package oder einem noch kleineren SMD-Gehäuse vorzusehen,[9] etwa in einem QFP-Gehäuse (*Quad Flat Package*, *Abb. 1.4a*). Bei den AVR®-Mikrocontrollern verbreitet ist die dünnere Bauform TQFP (*Thin Quad Flat Package*) mit 0,8 mm bis 0,5 mm Pinabstand. Benötigt man noch kleinere Gehäuse, greift man zum »beinchenlosen« QFN (*Quad-Flat No-leads package*) oder den fast nur mit professioneller Ausrüstung zu verarbeitenden BGA-Gehäusen (*Ball Grid Array*), wo die Anschlüsse auf der Unterseite positioniert sind (Abb. 1.4b).

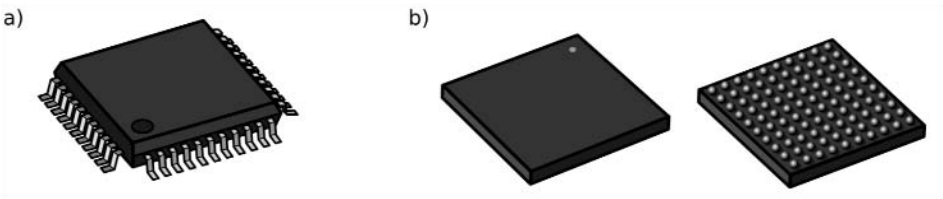


Abb. 1.4: QFP und BGA

Wir fassen zusammen:
Ein und derselbe Mikrocontroller kann in verschiedene Gehäuse eingebaut sein. Für einfache Prototypen sowie zu Experimentierzwecken eignet sich insbesondere das verhältnismäßig große DIP-Gehäuse, welches sich in Steckbretter und Lochrasterplatinen einsetzen lässt.
Im professionellen Bereich hingegen kommen hauptsächlich SMD-Gehäuse zum Einsatz, welche kleiner sind und bessere elektrische Eigenschaften aufweisen. Sie werden plan auf der Platine aufgelötet, Bohrungen wie beim Einsatz von Through-Hole-Bauteilen wie DIP-Gehäusen sind nicht nötig.

1.2.3 Datenblätter, Manuals und Errata

Jeder Controller hat ein Datenblatt, in dem (idealerweise) alle seine relevanten technischen Daten aufgelistet werden. Wie von nahezu jedem elektronischen Bauteil findet man sie zum kostenlosen Download auf der Webseite des Herstellers, in diesem Fall also bei Atmel® (www.atmel.com/avr).

[9] Natürlich darf deswegen nicht auf entsprechende Tests verzichtet werden – eine Schaltung kann sich auf dem Steckbrett signifikant anders verhalten als auf einer professionell gefertigten Platine!

Der Leser mag sich vielleicht wundern: Datenblätter, so »früh« im Buch? Tatsächlich ist ein gut geschriebenes und strukturiertes Datenblatt im Idealfall ein Crashkurs für Betrieb und Programmierung des jeweiligen Mikrocontrollers, bei dem kaum Fragen offen bleiben. Viele Informationen und Erklärungen aus diesem Buch ließen sich auch aus einem guten Datenblatt extrahieren. Die Voraussetzung dafür – und das kann nicht oft genug betont werden – ist natürlich, dass das Datenblatt auch gelesen wird. Wenn dem Entwickler zum ersten Mal ein Mitglied einer neuen Mikrocontrollerfamilie unterkommt, ist es sehr hilfreich, das Datenblatt zumindest einmal grob zu überblicken. Viele Fragen könnten geklärt, Fehler von vornherein vermieden und damit so einiger frustrierender Erfahrung vorgebeugt werden, wenn Datenblätter gewissenhafter konsultiert würden.

Sehen wir uns ein solches Datenblatt also genauer an: Auf der ersten Seite steht üblicherweise eine kurze Zusammenfassung aller wichtigen Eigenschaften wie Speichergrößen, Geschwindigkeit, Betriebsspannung, Leistungsaufnahme und eine Liste der Peripherieeinheiten.

Daneben sind alle Mikrocontroller-Typen aufgelistet, für die das Datenblatt gültig ist. Sehr viele Mikrocontroller unterscheiden sich nämlich nur im Speicherausbau und höchstens noch einigen Details und werden daher gemeinsam behandelt. Eine Software, die für einen Controller dieser Familie geschrieben wurde, kann meist ohne Änderungen am Code auf ein anderes »Familienmitglied« portiert werden (natürlich nur, wenn der Speicher ausreicht).

Von einigen Controllern gibt es auch besonders stromsparende Varianten, welche mit einer niedrigeren Versorgungsspannung auskommen. Diese werden im gleichen Datenblatt behandelt, wobei abweichende Werte für den jeweiligen Untertyp einzeln angegeben werden. Als Nächstes im Datenblatt folgt die Pinbelegung für alle erhältlichen Gehäusearten und eine Beschreibung aller Anschlusspins. Anschließend wird ein Überblick über die AVR-Architektur gegeben.

Praxistipp:
Meistens gibt es (abgesehen von Masse- und Versorgungsspannungsanschlüssen) pro Pin mehrere Möglichkeiten, wie dieser intern belegt werden kann. So könnte ein und derselbe »physikalische« Pin beispielsweise als allgemeiner Ein-/Ausgang (GPIO, General Purpose Input/Output), als ADC-Eingang oder als Datenleitung für eine Schnittstelle gewählt werden.[10]

[10] Es liegt am Entwickler, die Belegung sinnvoll zu wählen, damit nicht plötzlich ein ADC benötigt wird, nachdem sämtliche in Frage kommenden Pins als GPIO eingesetzt wurden.

Stichwortverzeichnis

S

Irmtraut Meister / Lukas Salzburger

AVR-
Mikrocontroller-Kochbuch

Programmieren ist wie Kochen

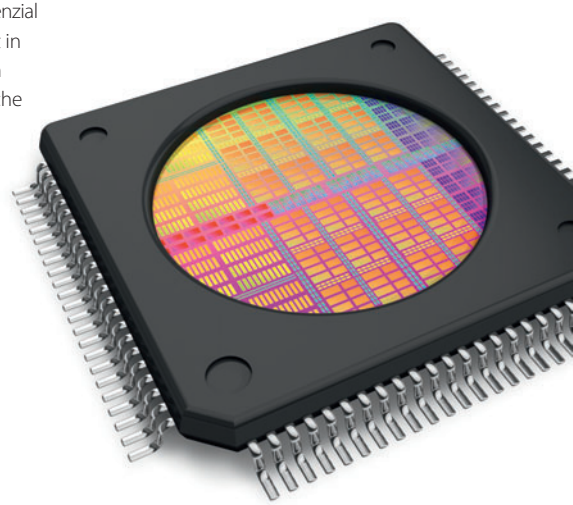
Rezept auswählen, Zutaten zusammenstellen – und genießen. Nach genau diesem Konzept finden Sie in diesem Buch alles, um Ihr „Mikrocontroller-Süppchen" zu kochen: Von den ersten Programmierschritten über Messungen unterschiedlichster Größen bis zum Erzeugen von Signalen und zur Kommunikation über diverse Schnittstellen.

Entdecken Sie die schier endlosen Möglichkeiten der Mikrocontroller! Mit nur wenig Programmieraufwand verwirklichen Sie im Handumdrehen Ihre Ideen. Schritt für Schritt begleitet dieses Buch Sie von den allgemeinen Grundlagen zur praktischen Umsetzung und erleichtert so auch komplexe Programmierungen.

Am Beispiel des AVR®-Mikrocontrollers von Atmel® lernen Sie das Potenzial von Mikrocontrollern kennen und können sich dadurch auch leicht in „fremde" Mikrocontroller einarbeiten. Für Einsteiger bietet das Buch auch Hinweise zur Programmierung von Bitoperationen und einfache Codegerüste – so bleiben keine Fragen offen.

Die Rezepte aus diesem Buch:

- Mikrocontroller-Grundlagen
- Programmierung und Implementierung
- Digitale Ein- und Ausgänge
- Spannungsmessung
- Spannungen ausgeben
- Widerstandsmessung
- Strommessung
- Zeit- und Frequenzmessung
- Kapazitäts- und Induktivitätsmessung
- Temperaturmessung
- Kommunikation mit Menschen
- Daten speichern



9 783645 651264

39,95 EUR [D]
ISBN 978-3-645-65126-4

Besuchen Sie unsere Website
www.franzis.de

FRANZIS