Florian Schäffer

AVR

Hardware und C-Programmierung in der Praxis

Elektor-Verlag, Aachen

1. Auflage 2008

Umschlaggestaltung: Etcetera, Aachen
Umschlagfoto: Leo Valen
Satz und Aufmachung: Ulrich Weber, Aachen
Druck: WILCO, Amersfoort (NL)

Printed in the Netherlands

ISBN 978-3-89576-200-0

Elektor-Verlag Aachen
079033-1/D

Inhaltsverzeichnis

Vorwort

Sie haben Elektronikkenntnisse und kennen sich auch bereits in einer (mehr oder weniger beliebigen) Programmiersprache aus? Nun wollen Sie beide Themengebiete miteinander verknüpfen und sich der Entwicklung sowie Programmierung von Mikrocontroller-Schaltungen widmen. Dann sollte dieses Buch genau das Richtige für Sie sein, denn mit den AVRs von Atmel stehen Ihnen zeitgemäße Prozessoren zur Verfügung, die alles in einem Chip integriert haben, was für den Einstieg und den fortgeschrittenen Anwender wichtig ist. Zudem bietet die Programmierung in C eine moderne, zukunftssichere Plattform für die Softwareentwicklung. Nach einer Einführung in die Thematik und der Vorstellung der notwendigen Entwicklungsumgebung können Sie mit Projekten, die Sie schrittweise zum Ziel bringen, loslegen. Für die meisten Projekte wird dabei das Mini-Mega-Board – eine Experimentierplatine aus der Zeitschrift Elektor – eingesetzt. Dadurch ist gewährleistet, dass die vorgeschlagenen Aufbauten auch reibungslos nachvollzogen werden können. Natürlich können Sie aber auch eine eigene oder eine andere Experimentierschaltung verwenden. Schließlich werden Sie auch später eigene, für Ihre Aufgabe maßgeschneiderte Schaltungen entwerfen und nutzen wollen. Die hier gezeigten Experimente sollen Sie keinesfalls an eine bestimmte Technik binden, sondern lediglich den Einstieg erleichtern.

Auch wenn an der einen oder anderen Stelle im Buch grundlegendes Elektronikwissen kurz aufgefrischt wird und auch mal eine etwas seltener benutzte C Programmiertechnik ausführlicher erläutert wird, soll hier keine Einführung in diese Themen erfolgen. Vielmehr stehen Inhalte aus der Praxis im Vordergrund, um möglichst umfangreich auf die Einsatzgebiete und Techniken der Mikrocontroller einzugehen. Aber keine Sorge: Da analoge elektrotechnische Klimmzüge die Ausnahme bilden, sind die Schaltungen leicht verständlich. Haben Sie Ihre Programmiererfahrung bisher in einer anderen Hochsprache gesammelt oder sind in C (noch) nicht ganz firm, dann finden Sie im Internet z. B. auf der Webseite http://www. galileocomputing. de/openbook/c_von_a_bis_z/ oder
http://www. schellong.de/c.htm eine sehr gute Einführungen in die Sprache, wo Sie auch mal kurz nachschauen können, wenn Sie Ihr Wissen nur etwas auffrischen wollen. Viele elektrotechnische Fragen werden in der FAQ der Newsgruppe de.sci.electronics bereits sehr anschaulich beantwortet. Die FAQ und auch eine Linkliste zu sehr vielen Herstellern von elektronischen (Spezial-) Bauteilen befinden sich u. A. auf der Webseite http://dse-faq. elektronik- kompendium.de.

Die im Buch vorgestellten Programme finden Sie als Download auf der Webseite des Verlages. Im Text wird dann stets der Dateiordner genannt, in dem Sie alle notwendigen Dateien finden. Die Schaltungsentwürfe, aber vor allem die Programme, sind möglichst einfach gehalten, um ein besseres Verständnis zu ermöglichen. Dadurch sind sie nicht immer optimal und universell geraten, sollen Sie aber auch vor allem dazu anregen, selber eigene Ideen auszuprobieren. Schließlich geht es nicht darum, die Beispiele stur durch zu exerzieren, sondern Sie sollen nach dem Studium befähigt sein, das Besprochene anzuwenden und kreativ weiter zu entwickeln.

Bevor Sie loslegen, noch ein kurzer Tipp: Der einfache Taschenrechner des Windows Betriebssystems (*Start/ Programme/ Zubehör*) beherrscht in der wissenschaftlichen Ansicht (*Ansicht/Wissenschaftlich*) die sehr oft benötigte Umrechnung zwischen hexadezimalen (im Buch beispielsweise als A1h angegeben), dezimalen (161) und binären (1010 0001b) Zahlen.

Und nun viel Spaß und viel Erfolg!

Florian Schäffer, Mai 2008
(http://www.blafusel.de)

1. Einführung in die Mikrocontrollertechnik

Sollten Sie Ihre Elektronikkenntnisse schon vor einiger Zeit gewonnen haben, so besitzen Sie sicherlich auch ein bereits vorgeprägtes Bild von Mikrocontrollern (im Fachjargon auch als MC oder μC bezeichnet). Wie bei allen Entwicklungen im Computerbereich sind die Anfänge noch gar nicht so lange her, und wer sich mit dem neuen Thema auseinandersetzen wollte, brauchte nicht nur viel Fachwissen, sondern oft auch das notwendige Budget und viel Geduld bei der Beschaffung der seltenen Einzelteile. Belohnt wurden die tage- und wochenlangen Mühen dann mit schukartongroßen Kästen, die bis auf ein paar LEDs und Schalter nicht viel Kommunikation mit dem Bediener boten.
Legendär ist zum Beispiel der in der Fachzeitschrift Popular Electronics 1975 vorgestellte Altair 8800 oder der spätere IMSAI 8080.

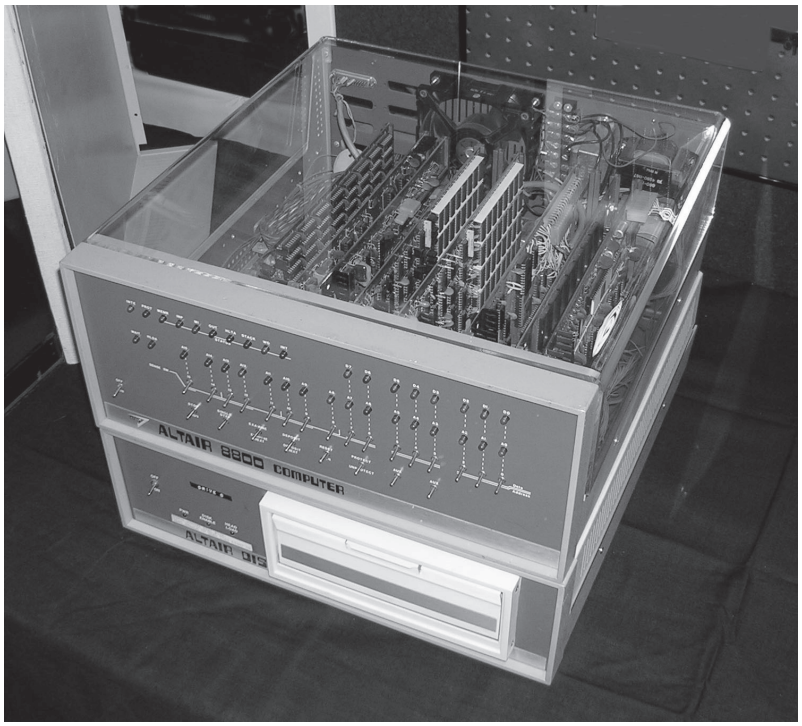

Abbildung 1.1: *Altair 8800 mit zusätzlichem Floppycontroller (Quelle: Wikipedia).*

Auch wenn der Lerneffekt beim Aufbau und Betrieb eines solchen Gerätes unbestreitbar hoch sein mag, so sind die Abmessungen, der aufwendige Aufbau und der fehlende Komfort bei Ein- und Ausgaben alles andere als alltagstauglich. Da ist es doch sehr erfreulich, dass heute ein einziger μC Chip in der Größe eines DIL IC-Gehäuses so gut wie alle Funktionen in sich integriert, die in den Anfangszeiten noch auf fünf Leiterplatten untergebracht werden

mussten. Zudem beträgt der Beschaffungspreis mit um die zwei Euro für den μC und vielleicht ca. zehn Euro für die Systemumgebung nur noch ein Bruchteil dessen, was ein Selbstbaugerät wie der Altair 8800 seinerzeit gekostet hat (397 US-$). Daneben bietet der Markt heute jede Menge Zubehör, das es dem Mikrocontroller ermöglicht, mit seiner Umgebung komfortabel zu kommunizieren und ihn effektiv und benutzerfreundlich jederzeit an neue Aufgaben anzupassen und ihn mit gängigen Programmierhochsprachen zu programmieren, statt endlos Kippschalter zu betätigen.

Haben Sie sich bisher also immer wieder von der vermeintlich aufwendigen Thematik der Mikrocontroller ferngehalten, dann bietet es sich an, die bisherigen Zweifel auszuräumen und zu sehen, wie einfach der Umgang mit eigenen Mikrocontrollern inzwischen geworden ist und welche vielfältigen Möglichkeiten ihr Einsatz bietet.

1.1 Einsatzgebiete für Mikrocontroller

Vielleicht haben Sie ja schon eine konkrete Aufgabe für Ihren ersten selbst aufgebauten μC. Eventuell hat Sie aber nur die Materie interessiert und Sie sind sich noch gar nicht so sicher, wozu Sie einen Mikrocontroller verwenden können und welche Aufgaben sich damit lösen lassen. Grundsätzlich kann man eigentlich sagen, dass in so gut wie jedem elektronischen Gerät heute ein μC steckt. Vorbei sind die Zeiten, in denen wahre IC Massengräber produziert wurden, um einfache Steueraufgaben zu verwirklichen. Und auch dort, wo bis vor einigen Jahren noch mechanische Systeme Steuerungen kontrollierten, werden inzwischen nur noch Mikrocontroller verwendet. Beispiele aus dem Alltag lassen sich in jeder Wohnung finden: Waschmaschinen, Geschirrspüler und Telefone waren vor gar nicht mal so langer Zeit noch hochmechanische Apparate. Ein von einem Elektromotor angetriebene mechanische Programmsteuerung regelte Wasserzufluss, Heizung, Trommelbewegung usw. Je nach eingestelltem Programm bewegten sich Zapfen oder ähnliche Steuerelemente auf einer Trommel und regelten die Wäsche – gut zu erkennen an dem typischen leisen Ticken der Maschinen.

In den Zeiten, in denen Telefone noch mit einer Wählscheibe ausgestattet waren, wurden die mechanischen Impulse der sich zurückstellenden Wählscheibe in elektrische Signale umgewandelt und dann in der Vermittlungsstelle über so genannte Hebdrehwähler zur Leitungsvermittlung genutzt. Bis auf wenige Ausnahmen übernehmen derartige Aufgaben inzwischen Mikrocontroller mit spezieller Programmierung, denn sie sind preiswerter, weniger fehleranfällig und leicht umzuprogrammieren, um Softwareupdates umzusetzen oder zusätzliche Funktionen zu integrieren.

Nun werden Sie vermutlich nicht eine Waschmaschine im Eigenbau herstellen wollen, wie sieht es aber mit einer Modelleisenbahn aus? Auch hier haben Mikrocontroller Einzug gehalten. Vorbei die Zeiten, in denen ein Trafo und ein paar „intelligente“ Weichen zur Zugsteuerung ausreichten. Jede Lok und sogar die Beleuchtung in den Passagierwagen verfügt über kleine Prozessoren, die über ein in die Versorgungsspannung der Gleise eingekoppeltes Signal gesteuert werden können, um Geschwindigkeit, Geräusche und auch Signalanlagen zu steuern. Und auch neben dem Gleis kann der μC Aufgaben übernehmen, für die bisher aufwendige Schaltungen notwendig waren: Die gesamte Beleuchtung einer Modellbahn-

anlage kann mit nur wenigen Bauteilen gesteuert werden. Auch wenn es etwas übertrieben erscheinen mag, einen ganzen Mikrocontroller damit zu beauftragen, die Leuchtreklame an einer Modellhauswand zu steuern: es ist vom Bauteilaufwand wesentlich einfacher als beispielsweise eine Schieberegisterschaltung (siehe auch ab Seite 127) und bietet zudem noch viel mehr Abwechslung bei den Lichteffekten bei vergleichbaren oder sogar geringeren Kosten und wesentlich weniger Schaltungsaufwand.

Mit Mikrocontrollern können Projekte oft billiger und einfacher (effektiver) realisiert werden, wodurch die Hemmschwelle vor der Umsetzung sinkt und der Weg für eigene Kreationen frei wird. Aufgrund der oft simplen Grundbeschaltung (Hardware) von Mikrocontrollern sinkt die Fehlerwahrscheinlichkeit und das Risiko, eine entworfene und produzierte Schaltung modifizieren oder entsorgen zu müssen. Da die Arbeitsweise des μC durch die Programmierung (Software) festgelegt wird, kann diese jederzeit modifiziert werden, ohne zwangsläufig auch Änderungen an der Hardware vornehmen zu müssen.

Für Mikrocontroller lassen sich zahlreiche Einsatzmöglichkeiten finden. Das wichtigste Aufgabengebiet wird dabei der Bereich Messen-Steuern-Regeln (MSR) sein: Mit dem μC wird eine Umgebungsbedingung gemessen und daraufhin wird ein Aggregat gesteuert, wodurch eine Regelung bewirkt wird, wenn sich der Vorgang wiederholt.

Bei allen Tätigkeiten ist ein Mikrocontroller dabei nicht auf die digitalen Anwendungen beschränkt, auch wenn intern lediglich Nullen und Einsen verarbeitet werden. Ebenso gut können analoge Daten eingelesen und ausgegeben werden. Solange man sich dabei nicht in den Bereich der Hochfrequenztechnik begibt, hat der Einsatz von Mikrocontrollern einen immensen Vorteil: Der Schaltungsaufwand ist übersichtlich und in der Regel sehr einfach zu verstehen, da meistens auf trickreiche Klimmzüge mit Transistoren, Kondensatoren usw. verzichtet werden kann (die vor allem bei Elektronikanfängern häufig zur Abschreckung führen) und bereits einfache Elektronikkenntnisse zu anspruchsvollen Ergebnissen führen.

1.2 Was ist ein Mikrocontroller?

Grundsätzlich stellt ein Mikrocontroller nichts anderes dar, als einen sehr kleinen Computer, der ebenfalls nach dem althergebrachten Funktionsprinzip Eingabe-Verarbeitung-Ausgabe (EVA) arbeitet. Der μC kann also Eingaben entgegennehmen, diese Daten verarbeiten und dann in irgendeiner Art und Weise darauf reagieren und etwas ausgeben. Die Eingabemöglichkeiten können vielschichtig sein: Beispielsweise Tastendrücke, Messwerte oder (Funk-) Signale. Je nach dem, welche Aufgaben das Programm vorsieht, können dann nach der Verarbeitung Reaktionen generiert werden, wie zum Beispiel: Leuchtanzeigen (LEDs, Textdisplays etc.), Kontrolle von Stellgliedern oder gesendete Signalpegel.

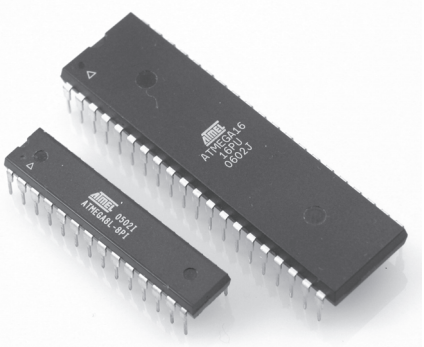

Abbildung 1.2:
μC ATmega8 (links) und ATmega16.

Die Unterscheidung zwischen Mikrocontrollern, wie sie hier im Buch verwendet werden, und den (Mikro -) Prozessoren (auch bezeichnet als CPU: central processing unit) in einem modernen PC ist nicht ganz trivial. Vereinfachend kann man sagen, dass ein Mikrocontroller bereits alle wichtigen Eigenschaften integriert hat, während eine CPU externe Peripherie benötigt. Je nach Modell kann ein μC beispielsweise bereits Analog/Digital-Wandler, serielle Schnittstellen oder verschiedene dauerhafte und flüchtige Speicher (für den Programmcode und als Arbeitsspeicher) integriert haben. Ein Computerprozessor benötigt hingegen hierfür weitere Hardware (beispielsweise RAM-Module als flüchtigen Speicher). Dafür bietet in der Regel eine moderne CPU wesentlich mehr effektive Rechenleistung als ein Mikrocontroller – auch wenn diese Unterscheidung nicht immer ganz zutreffend ist, so kann eine CPU zum Beispiel oft Arbeitsschritte parallel ausführen, wozu nur sehr wenige Mikrocontroller in der Lage sind, die hier im Buch keine weitere Beachtung finden werden.

Die Befehlsbearbeitung moderner Mikroprozessoren folgt dem Von-Neumann-Zyklus. Die wichtigsten Phasen sind dabei das Laden des Befehls (fetch), seine Dekodierung (decode) und seine Ausführung (execute). Für diese Schritte werden (je nach Controllerarchitektur) ein oder mehrere Taktzyklen benötigt, die als Impulse durch einen (internen oder externen) Taktgenerator ausgelöst werden. Die auszuführenden Befehle liegen als Maschinencode im Programmspeicher des Mikrocontroller und wurden durch den Programmierer erstellt und dort abgespeichert.

1.2.1 Mikrocontroller-Speicher

Abgesehen von den Prozessoren, in denen der Programmcode während der Herstellung mit eingebaut wird, befinden sich die Anweisungen normalerweise in einem lösch- und wiederbeschreibbaren Speicher. Festprogrammierte ROMs (Read Only Memory: Nur-Lese-Speicher) sind zwar in der Massenproduktion günstig aber für eigene Entwicklungen und Kleinserien aufgrund der mangelnden Flexibilität und aufwendigen Produktion ungeeignet. Mitte der 1970er Jahre wurden primär EPROMs (Erasable Programmable Read Only Memory: löschbarer, programmierbarer Nur-Lese-Speicher) für Eigenentwicklungen genutzt. Zum Programmieren (dem Abspeichern der Programmbefehle) wird ein (meist externes) Programmiergerät verwendet. Zum Löschen wird der Chip durch das eingelassene Fenster mit UV Licht bestrahlt. Um versehentliches Löschen des Speichers zu verhindern, wird das Fenster im Regelbetrieb mit UV-Dichter Folie abgeklebt. Etwa 100…200 Schreib-/Löschvorgänge kann ein EPROM verkraften bevor es unzuverlässig wird. Die Anzahl der Lesevorgänge ist hingegen nicht begrenzt. Eine Weiterentwicklung stellen EEPROMs (Electrically Erasable PROM: elektrisch löschbarer, programmierbarer Nur-Lese-Speicher) dar. Statt UV-Licht genügt eine (oft deutlich über der normalen Betriebsspannung liegende) Löschspannung, um den Speicher zu leeren.

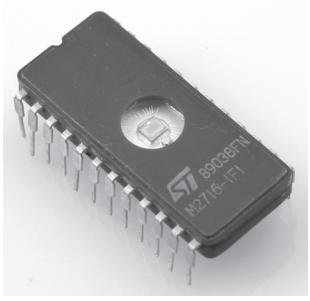

***Abbildung 1.3:** EPROM.*

Noch einen Schritt weiter geht der preiswertere Flash-Speicher (auch als Flash-EEPROM bezeichnet), der ähnlich wie ein EEPROM funktioniert. Allerdings genügt hier eine geringe externe Spannung (aus der dann intern die benötigte höhere

Spannung generiert wird) auch für den Löschvorgang, was den Schaltungsaufwand vereinfacht. Dadurch benötigen Mikrocontroller mit Flash-Speicher oft auch kein zusätzliches Programmier- und Löschgerät, sondern können direkt in der Schaltung programmiert werden. Zudem sind Speicherzugriffe in wesentlich kürzerer Zeit als beim (E)EPROM möglich. Eine typische Anwendung für Flash sind (USB-) Memory Sticks wie sie heute wohl jeder Computernutzer kennt. Nachteilig gegenüber einem EEPROM ist bei Flash allerdings, dass die Speicherzellen (Bytes) nicht einzeln, sondern nur blockweise gelöscht werden können. In der Mikrocontroller-Praxis ist dies aber kaum von Bedeutung. Wie alle wiederbeschreibbaren Speicher können auch Flash-Speicher (ähnlich wie EEPROMs) nur etwa 100.000 bis 1.000.000 Mal beschrieben werden.

Die heute eingesetzten Mikrocontroller vereinen meist eine Mischung aus diesen Speichertechniken: Für den Programmcode wird oft Flash genutzt, da dieser nicht flüchtig ist und so das Programm auch ohne Spannung im μC erhalten bleibt. Berechnungen werden im unbegrenzt oft wiederbeschreibbaren (S)RAM (Static Random Access Memory: statischer Speicher mit wahlfreiem Zugriff) abgelegt und bleiben dort so lange erhalten, wie der μC mit Spannung versorgt ist. Zusätzlich kann es ein internes EEPROM geben, in dem der Programmierer Werte dauerhaft speichern kann (beispielsweise Konfigurationsparameter des Benutzers).

1.2.2 RISC-Architektur

Der Atmel AVR, mit dem sich dieses Buch hier beschäftigt, gehört zu einer 8 Bit RISC-Prozessor-Familie. RISC steht für Reduced Instruction Set Computing und bedeutet rechnen mit reduziertem Befehlssatz, was sich im ersten Moment als Nachteil gegenüber dem Complex Instruction Set Computing (CISC: rechnen mit komplexem Befehlssatz) anhört. Im Vergleich zum RISC-Befehlssatz zeichnet sich ein CISC-Befehlssatz durch verhältnismäßig leistungsfähige Einzelbefehle aus, wohingegen RISC zugunsten einer hohen Ausführungsgeschwindigkeit und eines niedrigeren Decodierungsaufwands auf Seiten der CPU versucht, auf komplexe Befehle zu verzichten. Zu den bekanntesten CISC Vertretern gehören die Prozessoren der Motorola 68000er Reihe (üblich in Apple Macintosh Computern) und die Intel x86er (Pentium & Co.). RISC-Prozessoren gelten allgemein als schneller bei der Programmausführung, bereiten aber bei der Programmierung vor allem in Assembler mehr Aufwand, da die fehlenden komplexen CISC-Befehle durch mehrere einfache RISC-Befehle ersetzt werden müssen. Wenn Sie die Programme für Ihren Mikrocontroller in einer Hochsprache wie C erstellen, werden Sie den Mehraufwand nicht bemerken, da der Compiler die Arbeit für Sie übernimmt.

Zudem wurde der AVR explizit für die Programmierung mit C entworfen, so dass diese Compiler sehr effizienten Maschinencode generieren können, was sie auch einigen speziellen Befehlen verdanken, die dann doch aus dem CISC-Umfeld stammen. Während ein echter RISC-Befehl in nur einem Programmzyklus ausgeführt werden kann, gibt es auch Befehle, die zwei bis vier Takte benötigen. Allerdings muss man als Entwickler auf derartige Gegebenheiten nur bei sehr zeitkritischen Anwendungen achten.

1.3 Der passende Mikrocontroller-Typ

Beginnen Sie aus eigenem Interesse, als Hobby oder zur Realisierung eines bestimmten Projektes sich mit Mikrocontrollern zu befassen, dann stehen Sie anfangs immer vor der Frage, welcher μC denn nun der beste ist. Grundsätzlich kann man schon mal nie von *dem Besten* reden, denn jeder Mikrocontroller hat Vor- und Nachteile. Zudem haben Sie sich mit dem Lesen dieses Buches schon auf einen bestimmten Typ festgelegt: Atmel AVRs. Dennoch sollen hier einige der Entscheidungskriterien kurz beleuchtet werden. Zu den wichtigsten Fragen bei der Controllerauswahl gehören u. a. diese Punkte:

- Funktionsumfang Befehlssatz
- Geschwindigkeit
- Funktionsumfang integrierter Hardware
- Preis und Beschaffbarkeit
- Bauform
- Speichergröße
- Qualität der Dokumentation
- Akzeptanz bei anderen Nutzern
- Verfügbarkeit an Entwicklungstools

Wie Sie sehen, betreffen die meisten Fragen gar nicht den Mikrocontroller direkt, sondern Rahmenbedingungen, die teilweise viel wichtiger sein können. Als problematisch bei der Prozessorwahl stellen sich oft die sehr eingeschworenen Fangruppen heraus. Jeder Prozessor hat eine feste Gruppe von Anwendern, die oft nicht bereit sind, andere Produkte wertfrei zu würdigen und nur *ihren* Prozessor als den einzig richtigen gelten lassen – Ähnliches wird Ihnen bei der Wahl der passenden Programmiersprache erneut begegnen. Lassen Sie sich davon nicht abschrecken. Wichtig ist es, überhaupt erst einmal einen Einstieg zu finden. Wenn Sie dann später feststellen, dass der zuerst gewählte Mikrocontroller doch nicht der zu Ihrem Projekt optimal passende war, haben Sie trotzdem eine Menge gelernt und können Ihr Wissen schnell transferieren. Auch der Autor dieses Buches hat mehrere Anläufe benötigt und sich vor vielen Jahren mit Prozessoren wie dem 8085 und dem 8052 herumgeschlagen, bevor er dann nach langer Pause mit der C-Control einen neuen Einstieg fand und nun derzeit AVRs einsetzt. Immerhin schreitet auch die technische Entwicklung immer weiter voran und Ihre Anforderungen werden sich ebenso ändern.

1.3.1 Funktionsumfang Befehlssatz

Im Grunde kann es Ihnen mehr oder weniger egal sein, wie viele Befehle ein Prozessor intern kennt. Wie gezeigt, verfügen RISC-Prozessoren über weniger als bei einer CISC-Architektur. Die AVRs von Atmel besitzen etwa 60…130 so genannter Opcodes . Erst wenn Sie sich mit der Assemblerprogrammierung befassen, werden diese Befehle im Einzelnen bedeutungsvoll, da Sie dann eventuell fehlende CISC-Befehle durch mehrere RISC-Befehle nachbauen müssen. Nutzen Sie eine Hochsprache wie Basic oder C, dann übernimmt der Compiler diese Arbeit für Sie.

Auch wenn Funktionen wie Timer, Counter und Interrupts nicht unmittelbar zum Befehlssatz zu zählen sind, so sind dies Funktionen, bei denen Sie sich unbedingt vorher überlegen müssen, in wie weit sie benötigt werden. Im weiteren Verlauf des Buches werden Sie die Einsatzgebiete dieser Funktionen noch detailliert kennen lernen. Vorhanden sind die Funktionen zwar oft, doch es kann durchaus sein, dass zum Beispiel zu wenig Timer zur Lösung Ihres speziellen Problems verfügbar sind.

1.3.2 Geschwindigkeit

Es kommt ganz auf Ihre Anwendung an, ob die Prozessorgeschwindigkeit tatsächlich eine Rolle spielt. Für eine Heizungssteuerung ist es sicher unerheblich, ob die Temperaturregelung Mikrosekundengenau ausgeführt wird. Anders mag es aussehen, wenn Sie sehr kurze Zeiten messen wollen (beispielsweise die Laufzeit von Ultraschall zur Entfernungsmessung Ihres Roboters) oder eine große Anzahl von LEDs im Multiplexbetrieb (siehe Seite 121) ansteuern. Wie lange eine Abfolge von Befehlen dauert, können Sie dem Datenblatt zum Prozessor entnehmen. Dort steht dann beispielsweise unter dem Stichwort *Instruction Set Summary* für jeden einzelnen Maschinenbefehl, wie viele Taktzyklen er benötigt. Doch ehrlich gesagt: davon haben Sie herzlich wenig, denn so lange Sie nicht schon den fertigen Maschinencode Ihres Programms haben, wissen Sie nicht, welche und wie viele Anweisungen Ihr Compiler erzeugen wird.

Können Sie bei Ihrem Prozessor die Taktrate konfigurieren (beispielsweise mit Hilfe eines externen Schwingquarzes), dann haben Sie ein wenig Einfluss auf die unmittelbare Zeit, die ein Taktzyklus benötigt. Beherrscht Ihr Mikrocontroller also ein großes Spektrum als Taktfrequenz (beispielsweise bis zu 20 MHz bei ATmegas), haben Sie einen Vorteil gegenüber Prozessoren, die nur langsamer getaktet werden können. Trotzdem wird man oft nur eine mittlere Taktrate wählen und nicht die maximal mögliche ausreizen, da dies für die meisten Anwendungen gar nicht nötig ist. Vor allem beim Einsatz interner Timer (mehr dazu ab Seite 180) können hohe Taktraten sogar eher störend sein, da sich dann das Programm um die zu oft auftretenden Überläufe etc. kümmern muss.

Eine geringe Ausführungsgeschwindigkeit wirkt sich aber auch sehr störend aus, wenn beispielsweise Zeichen über die serielle Schnittstelle gesendet und empfangen werden sollen und der Prozessor nicht in der Lage ist, höhere (übliche) Übertragungsraten zu bewerkstelligen und dann eventuell empfangene Zeichen gar verloren gehen, was aber auch mit dem zur Verfügung stehenden Speicher zusammenhängen kann.

Letztendlich wird bei Aussagen zur real nutzbaren Geschwindigkeit die praktische Erfahrung eine Rolle spielen. Aktuelle Mikrocontroller stoßen aber eher selten an ihre Grenzen.

1.3.3 Funktionsumfang integrierter Hardware

Dieser Punkt kann ganz entscheidend sein, denn wenn der von Ihnen gewählte Mikrocontroller eine benötigte Funktion nicht besitzt, müssen Sie diese bei Bedarf durch externe Hardware oder Programmtechnisch nachbilden. Deshalb sollten Sie sich stets vor Projekt-

beginn fragen, was Sie benötigen und ob Sie dies bereits integriert haben möchten. Vor allem Schnittstellen gehören hierzu:

- Anzahl digitaler I/O-Ports (Input/Output: Eingabe/Ausgabe)
- Analog-Digital- (A/D) und/oder Digital-Analog-Wandler (D/A)
- Serielle Schnittstellen wie USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter), SPI (Serial Peripheral Interface) oder I^2C/TWI (Two-wire Serial Interface)
- CAN-Schnittstelle
- Taktgenerator
- Spannungsversorgung
- Programmierschnittstelle und Debugfunktionen

Die digitalen I/O-Ports sind vermutlich die Schnittstelle, mit der Sie an Anfang am meisten experimentieren werden, da man hier z. B. Schalter, LEDs usw. anschließen kann. Wenn viele Leitungen vorhanden sind, können natürlich entsprechend viele Aktuatoren und Sensoren direkt angeschlossen werden. Wie Sie noch sehen werden, gibt es aber auch zahlreiche Möglichkeiten, wie Sie sich bei zu wenig freien Ports behelfen können. Interessant kann aber auch sein, wie stark ein einzelner Port oder alle Ports zusammen strommäßig belastet werden können. Nicht alle Mikrocontroller sind in der Lage, Standard-LEDs mit einer Leistungsaufnahme von ca. 20 mA direkt anzusteuern. In so einem Fall wird dann externe Hardware zusätzlich erforderlich, was zum Beispiel bei Platzmangel störend sein kann.

Besitzt der Mikrocontroller einen internen Oszillator zur Taktgenerierung, dann kann auf eine externe Beschaltung verzichtet werden, solange keine hohe Genauigkeit benötigt wird, wodurch der Beschaltungsaufwand weiter sinkt. Ebenso verhält es sich bei der Spannungsversorgung: Mikrocontroller mit internem Speicher dürfen nicht an einer Spannungsquelle betrieben werden, deren Spannung an- oder absteigt beim Erreichen der nominellen Ausgangsspannung oder beim Ausschalten. Mittels Brownout -Erkennung wird dies verhindert, da es ansonsten zum Löschen oder Verändern von Speicherinhalten kommt. Bringt der μC eine solche Funktion bereits mit, können Sie wieder drei externe Bauteile einsparen.

Der Programmierschnittstelle kommt noch mal eine besondere Beachtung zu. Prozessoren wie die AVRs unterstützen das so genannte In-System Programming (ISP). Damit ist gemeint, dass der Speicher des μC jederzeit direkt in der fertig aufgebauten Schaltung neu programmiert werden kann (siehe Seite 31f). Um z. B. den Programmcode zu ändern wird also kein externes Programmiergerät benötigt, was Zeit und Geld spart. Zudem kann bei den größeren Modellen über die JTAG- (Joint Test Action Group) Schnittstelle der Programmcode direkt im Prozessor debugt werden, um Fehler bequemer aufzufinden (vgl. S. 44).

1.3.4 Preis und Beschaffbarkeit

Was nützt Ihnen der schickste Prozessor, wenn es so gut wie unmöglich ist, ihn als normaler Anwender zu beschaffen, da kein Einzelhändler ihn in seinem Programm führt? Es ist vorteilhaft, wenn Sie den Chip bei einem der großen Elektronikgeschäfte bekommen können, bei

denen Sie auch die übrigen Bauteile beziehen. Ein Chip, der vielleicht nur einige Euro kostet, verteuert sich sonst ungemein, wenn Sie ein mehrfaches an Versandkosten hinzurechnen müssen, weil Sie beim Distributor des Controllers nicht auch noch Ihren übrigen Bedarf decken können. Welche Schmerzgrenze Sie für den Kaufpreis eines einzelnen Chips setzen, hängt primär von Ihren Verhältnissen und Bedürfnissen ab. Einfache μC wie der ATmega8 sind aber schon für unter zwei Euro zu haben (Stand: 11/2007). Angesichts dessen sinkt die Hemmschwelle für eigene Experimente angenehm tief, denn selbst wenn mal etwas schief gehen sollte, schmerzt es kaum, zum Ersatz zu greifen; und sind die ersten Schritte und Hürden gemeistert, können Sie sich den teureren Versionen beruhigt zuwenden.

An dieser Stelle deshalb auch noch ein Tipp: Alle Hersteller von Halbleitern bieten auf ihren Webseiten die Möglichkeit, Muster einzelner Produkte kostenlos zu bestellen. Sollten Sie also mal ein exotisches Bauteil ausprobieren wollen, können Sie dort ein Exemplar bestellen und werden in der Regel zügig und problemfrei beliefert.

Bei Fragen zur Beschaffbarkeit lohnt sich aus einem anderen Grund aber auch immer ein Blick auf die Herstellerwebseite: Prüfen Sie, ob der Halbleiter (Mikrocontroller oder ein anderes Bauteil) nicht schon abgekündigt (engl: deprecated) wurde. Vor allem Prozessoren haben oft nur einen kurzen Produktionszyklus und werden dann durch neue Modelle ersetzt. Eine Zeit lang bekommen Sie diese Bauteile dann zwar oft sehr günstig zu kaufen, laufen aber Gefahr, irgendwann keinen Nachschub mehr zu bekommen, was vor allem bei einer geplanten Serienproduktion sehr ärgerlich sein kann.

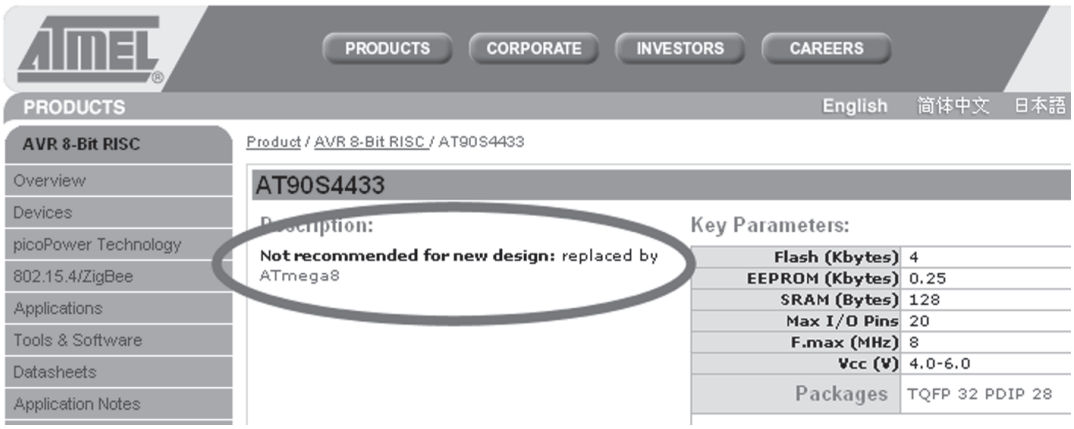

Abbildung 1.4: *Beispiel für Hinweis auf ein auslaufendes Modell (Quelle: www.atmel.com).*

1.3.5 Bauform

Nicht außer Acht lassen sollte man die Bauform, in der ein Mikrocontroller angeboten wird. Ein DIL (Dual In-Line) Gehäuse mit 28 Beinchen ist wesentlich handlicher als ein 64-poliges TQFP Gehäuse oder gar ein BGH (Ball Grid Array), welches sich mit Heimmitteln gar nicht mehr verarbeiten lässt. Dabei ist natürlich unter anderem ausschlaggebend, ob Sie den Chip

auf einem Experimentierboard oder einer auch noch von Laien lötbaren Platine einsetzen wollen, oder gleich an die Seriengroßproduktion bei minimalem Platzbedarf denken.

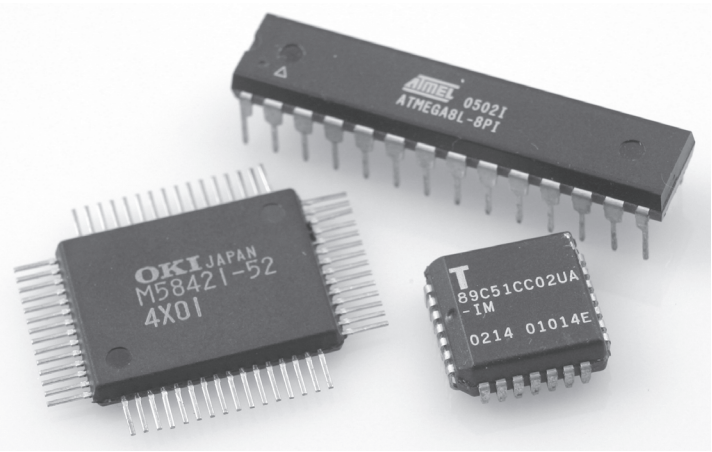

Abbildung 1.5: *Verschiedene Gehäuseformen: SMD, PLCC und DIL.*

Vorteilhaft kann es sein, wenn der gleiche Controller in verschiedenen Bauformen existiert. So können Sie während der Entwicklungsphase auf eine Form zurückgreifen, die sich gut löten lässt oder für die es passende Sockel zum einfachen Wechsel des Chips gibt und später bei der Serienfertigung zu den meist kleineren SMD-Varianten (Surface Mounted Device) und ähnlichem wechseln.

1.3.6 Speichergröße

Generell trifft es natürlich zu, dass man nie genug Speicher haben kann und viel immer gut ist. Da ein mehr an Speicher sich aber auch fast immer in der Gehäusegröße niederschlägt, lohnt es sich doch, sparsam zu sein. Wie schon im Kapitel Mikrocontroller-Speicher ab Seite 13 gezeigt, gibt es verschiedene Speicher in einem Mikrocontroller.

RAM ist meistens kein Problem, denn die Anzahl der verwendeten Variablen ist meistens gering und auch sonst werden bei Mikrocontroller-Anwendungen i. d. R. keine großen Datenmengen im Speicher bewegt (wie beispielsweise bei Datenbanken und deren Sortierung).

Speicher in Form von EEPROM kann schon interessant werden: Hier werden meistens Konfigurationsparameter abgelegt, die auch nach einem Spannungsabfall wieder verfügbar sein sollen (beispielsweise durch den Benutzer ausgewählte Sprache für Textausgaben oder Soll-Werte für in der Regelungstechnik). Bei den meisten Anwendungen beschränken sich die Daten aber trotzdem auf wenige Bytes. Interessant kann ein großer EEPROM-Speicher aber werden, wenn er als Ergänzung eines eventuell knappen Flash-Speichers herhalten soll.

Im Flash wird das Programm abgelegt. 8 kByte sind für viele Heimanwendungen oft schon mehr als genug. Der tatsächliche Verbrauch lässt sich zuvor kaum bestimmen und ist auch stark vom Compiler abhängig. Hier hilft wieder einzig die Erfahrung und da ist es dann na-

türlich wieder mal praktisch, wenn der gleiche Mikrocontroller in verschiedenen Speicherausbaugrößen verfügbar ist.

Es gibt aber Anwendungen, die schnell viel Speicher verbrauchen: Grafik und Text. Schon für einfache Textausgaben auf einem LCD wird pro Zeichen ein Byte RAM benötigt, da ja im Programmcode das auszugebende Zeichen abgelegt sein muss. Für eine kleine Menüführung und benutzerfreundliche Ausgaben wird dann oft schnell mehr Speicher belegt, als man vermutet. Grafikausgaben sind noch Speicherhungriger, denn pro Pixel wird ein Bit benötigt und bereits ein Bild auf einem kleinen Display mit 128x64 Pixel belegt 1.024 Byte – so viel Speicher wie bei vielen Typen gerade maximal zur Verfügung steht. Hier hilft man sich als Entwickler dann oft mit einem Trick und lagert den statischen Text oder die Grafik in den nicht ausgelasteten Flash-Speicher aus (dazu mehr ab Seite 199).

1.3.7 Qualität der Dokumentation

Dieses Kriterium bei der Wahl des passenden Mikrocontrollers gehört (wie die folgenden Kriterien auch) zu den oft vernachlässigten, aber deshalb nicht weniger wichtigen. Der super-tolle Prozessor ist nichts Wert, wenn Sie nicht wissen, wie er beschaltet und bedient wird. Die meisten (freien) Dokumente werden Sie heute online im Internet finden. Deshalb kann es nicht schaden, mal auf der Herstellerwebseite vorbei zu schauen. Dort werden Ihnen meistens umfangreiche Dokumente als PDF (Portable Document Format) angeboten, die Sie dann mit dem kostenlosen Acrobat Reader (http://www.adobe.com) betrachten können. Angesichts des Umstandes, das sämtliche Hersteller von Mikrocontrollern amerikanische oder asiatische Firmen sind, können Sie sich aber darauf einstellen, hier stets nur englischsprachige Dokumente zu finden. Ohne entsprechende Sprachkenntnisse werden Sie grundsätzlich Probleme bekommen, denn der Markt an deutschsprachiger Sekundärliteratur ist oft dünn und vor allem vermutlich nie so umfassend wie die Datenblätter (Datasheets) der Hersteller.

Dafür bieten die Hersteller oft auch weiterführende Hilfen wie zum Beispiel *Application Notes* an, in denen oft interessante Einzelprojekte komplett vorgestellt werden. Diese sind immer wieder eine gute Hilfe, wenn es darum geht, vielleicht mal ein ähnliches Projekt umzusetzen, denn viele oft benötigte Anwendungen sind dort bereits gelöst worden, da die Hersteller ja auch daran interessiert sind, dass Ihre Mikrocontroller problemlos eingesetzt und somit verkauft werden.

In diesem Buch soll bis auf wenige Ausnahmen darauf verzichtet werden, Inhalte aus Datenblättern wiederzukauen. Bitte konsultieren Sie immer die Datenblätter, die Sie allesamt als PDF beim Hersteller der eingesetzten Schaltkreise herunterladen können. Die Dokumente zu den Atmel Prozessoren bekommen Sie auf der Webseite http://www. atmel.com/dyn/products/data sheets. asp? family_id=607.

1.3.8 Akzeptanz bei anderen Nutzern

Solange Sie nicht ein Entwicklerdasein als Einzelkämpfer fristen wollen, ist eine möglichst große Nutzergemeinschaft sehr hilfreich. Früher oder später wird es sich nicht vermeiden lassen und Sie werden ein schier unlösbares Problem mit Ihrem Mikrocontroller haben oder Sie suchen nach einem Gedankenanstoß zur Lösung. Dann ist es schön, wenn Sie (heutzutage wohl vor allem im Internet) andere Leidensgenossen treffen und sich mit ihnen austauschen können. Vielleicht hat sich schon mal jemand mit einem ähnlichen Problem befasst oder nimmt Ihnen mit seinem Rat das Brett vor dem Kopf weg, an dem Sie schon fast verzweifeln. Je beliebter ein Prozessor dann ist und je mehr Fans er hat, desto leichter wird es Ihnen fallen, Hilfe zu erhalten. Betrachten Sie aber nicht nur den Mikrocontroller losgelöst von seiner Umgebung. Bei der eigentlich beliebten C-Control fällt zum Beispiel auf, dass zwar viele Anwender diese in Basic programmieren, aber so gut wie niemand zu finden ist, der hier mit C arbeitet.

Atmels AVRs erfreuen sich auch in Deutschland so großer Beliebtheit, dass Sie in der Regel sogar einen deutschsprachigen Kontakt finden werden, was bei aller sprachlichen Gewandtheit meistens angenehmer ist.

Aus diesem Grunde hier eine kleine (unvollständige) Auswahl empfehlenswerter Anlaufstellen bei Elektronik- und Programmierfragen mit Schwerpunkt AVR:

- http://www.elektor.de/forum/ Allgemeines Elektronikforum mit μC Unterforum des Elektor-Verlages.
- http://www.blafusel.de/phpbb/ Forum des Autors. Vor allem bei Fragen direkt zu diesem Buch geeignet.
- http://www.mikrocontroller.net/forum/ Forum und Artikelsammlung rund um Mikrocontroller, schwerpunktmäßig AVRs.
- http://www.roboternetz.de/ Forum vor allem für Robotik aber auch deren Mikrocontroller-gestützte Programmierung.
- de.sci.electronics Usenet-Forum (bspw. über http://groups.google.de nutzbar) zum Thema Elektronik im Allgemeinen.
- http://www.avrfreaks.net/ Englischsprachiges Forum und Artikelsammlung zu AVRs. Teilweise Anmeldung erforderlich.

1.3.9 Verfügbarkeit an Entwicklungstools

Der beste Mikrocontroller ist ohne Entwicklungsumgebung ziemlich wertlos. Sie benötigen in der Regel wenigstens noch ein Gerät, um den Prozessor zu programmieren (bei der In-System Programmierung von AVRs genügen im Notfall ein paar Widerstände) und eine Programmiersprache. Sowohl für die Programmierung wie auch für die Softwareentwicklung benötigen Sie einen PC mit Betriebssystem. Die beste Verfügbarkeit haben sicherlich Programme, die auf Microsoft Windows laufen. Allerdings kann es sein, dass nicht alle Versionen unterstützt werden. Unix-Derivate werden schon weniger unterstützt und bei Apple Macintosh wird die Luft dünn. Ihr PC benötigt zudem eine Schnittstelle. Parallele oder se-

rielle Schnittstellen bereiten die wenigsten Probleme, sind aber oft vor allem bei Laptops nicht mehr vorhanden, so dass auf USB ausgewichen werden kann.

Software, um den fertigen Code in den Prozessor zu übertragen (aus Tradition spricht man hier noch immer vom *brennen*), gibt es wohl für jedes System. Simulatoren, also Software, die die Ausführung einer fertige Applikation auf dem PC nachbildet und Registerzustände anzeigt oder ganze Grafik-LCDs nachbildet, sind hilfreich aber trotzdem auch verzichtbar, so dass eine fehlende Unterstützung für das eigene Betriebssystem verwunden werden kann.

Unumgänglich ist natürlich eine Programmiersprache. Für die meisten Mikrocontroller werden Assembler, Basic und C angeboten. Historische Sprachen für μC wie Fortran und Ada werden so gut wie gar nicht mehr angeboten. Je nach Geldbeutel und Ambition bietet der Markt ein breites Spektrum von kostenlosen bis hin zu teuren Kaufprodukten. Gelegentlich gibt es auch abgespeckte oder eingeschränkte Kaufversionen als freie Software. Beispielsweise wird für AVRs eine kostenlose Version von BASCOM -AVR angeboten, die aber auf 4 kByte ausführbaren Code beschränkt ist, womit also nur die Hälfte des Flash-Speichers eines ATmega8 ausgenutzt werden kann. Wie bei allem ist auch bei der Wahl der Programmiersprache darauf zu achten, dass eine hochwertige Dokumentation existiert und die Sprache in der Lage ist, die gewünschten Aufgaben zu bewältigen, denn nur weil in Basic üblicherweise nach einer Fallunterscheidung mehrere Anweisungen als Block zusammengefasst werden können, darf man daraus nicht schlussfolgern, dass jeder Basic-Dialekt dies auch beherrscht.

1.4 Entscheidungshilfe Atmel AVR

Auch wenn Sie sich schon durch die Lektüre dieses Buches quasi auf einen 8 Bit RISC Prozessor der AVR-Familie von Atmel festgelegt haben, so bleibt Ihnen immer noch die Qual der Wahl, welchen μC Sie tatsächlich verwenden. Schwerpunktmäßig wird zwar die Reihe ATmega betrachtet, aber auch die anderen AVRs können für Sie interessant sein und sind ebenso einfach anzuwenden.

Primäres Unterscheidungsmerkmal stellt wohl der jeweils vorhandene Speicher, die Anzahl an I/O-Leitungen und die maximal zulässige Taktrate dar. Die Reihe der ATtiny zeichnet sich vor allem durch kleine Gehäuse aus. Daraus resultierend stehen nur wenige (6-18) I/O-Pins zur Verfügung. Die älteren (und für neue Designs nicht mehr empfohlenen) Modelle können oft nur relativ langsam mit 2-6 MHz getaktet werden – im Gegensatz zu den neueren Vertretern mit bis zu 20 MHz. Auch ist der Speicherplatz sehr beschränkt. Mit 1-8 kByte (die meisten sogar nur 1-2 kByte) bieten sie nur für kleine Anwendungen Platz. Einzelne Modelle besitzen allerdings kein SRAM. Dafür sind sie preislich sehr attraktiv und schon für deutlich unter einem Euro erhältlich. Dadurch werden die ATtiny vor allem dort interessant, wo bisher umfangreiche Konstruktionen aus Standard-Logik ICs eingesetzt wurden. Aber auch einfache Schaltungen aus diskreten Bauteilen lassen sich oft Platz- und Geldsparend ersetzen. So kann aus einem einzelnen ATtiny, zwei LEDs und zwei Vorwiderständen bereits eine komplette Wechselblinkschaltung aufgebaut werden, wozu sonst ein NE555 oder eine astabile Kippstufe (Multivibrator) mit einer Handvoll zusätzlicher Bauteile

benötigt wird. Wird der ATtiny entsprechend seiner Verfügbarkeit an I/O-Pins um weitere LEDs ergänzt, kann er auch gleich noch ein Lauflicht erzeugen und je nach Programmierung sogar zwischen verschiedenen Darstellungen wechseln, wozu eine diskret aufgebaute Schaltung nicht oder nur mit immensem Teileaufwand in der Lage ist. Die meisten ATtinys gibt es in einem 8-poligen DIL oder SOIC-8 SMD Gehäuse.

Reicht der Speicher und/oder die Anzahl an Datenleitungen nicht mehr aus, schlägt die Stunde der ATmega -Reihe. Das kleinste Mitglied stellt der ATmega8 dar, der (wie sein Produktname schon andeuten soll) 8 kByte Flash besitzt. Gefolgt von den Modellen mega16, mega32, mega64 und mega128 mit jeweils 16, 32, 64 und 128 kByte bei teilweise ebenfalls zunehmendem RAM und EEPROM. Die kleineren ATmega werden in einem 40-poligen DIL oder 44-poligen TQFP Gehäuse geliefert. Bei den größeren gibt es dann teilweise nur noch SMD-Varianten. Den kleine ATmega8 gibt es in einem etwas ungewöhnlichen schmalen 28-poligen DIL Gehäuse (7,62 mm Abstand der Anschlussbeinchenreihen) und als TQFP-32. In der Reihe der ATmegas stellt der mega8 den einzigen μC dar, der über kein JTAG-Interface verfügt (vgl. S. 44).

Neben den auffälligen Eigenschaften der Standardtypen gibt es auch eine Reihe an Untertypen mit vielfältigen Variationen. Vor allem in Betracht auf den internen Funktionsumfang gilt es bei der Wahl des passenden Mikrocontrollers aufmerksam zu vergleichen. Auch steigt der Preis hier deutlicher zwischen den einzelnen Modellen von ca. zwei Euro bis hinauf zu acht Euro an. Zu berücksichtigen sind dann bei der Auswahl u. A. folgende Punkte, die auch bei ATtinys wichtig sind:

- maximal möglich Taktrate in Abhängigkeit der Versorgungsspannung
- Versorgungsspannung (es gibt Typen mit 2,7 - 5,5 V und welche mit 4,5 - 5,5 V bei ATmega und 1,8 - 5,5 V bzw. 2,7 - 5,5 V bei ATtiny)
- Anzahl der Timer etc.

Mit den ATmegas lassen sich sehr umfangreiche Projekte realisieren, da genügend Speicher auch für aufwendige Text- und Grafikausgaben vorhanden ist und über die zahlreichen Datenleitungen gut Kontakt zur Außenwelt hergestellt werden kann.

Grundsätzlich sind die einzelnen Modelle untereinander zwar sehr ähnlich, aber nicht immer Pinkompatibel und so gut wie nie Softwarekompatibel. Da sich vor allem die Register (siehe S. 94) jeweils ändern, bedarf es i. d. R. einer Anpassung und erneuten Compilierung des Quellcodes.

Für einige Spezialanwendungen gibt es zudem noch AVR Derivate, die dann bereits fertige Controller für LCD-Steuerungen, CAN oder Licht und Motoren beinhalten.

Die Familie der AT89er gehört trotz Namensähnlichkeit nicht zu den AVR Typen mit dem RISC Befehlssatz, sondern ist eine Intel 8051 kompatible 8-Bit μC Serie.