mitp

Herbert Bernstein

Mikrocontroller-programmierung

in Assembler und C

für die Mikrocontroller der 8051-Familie
Simulation unter Multisim

Inklusive CD-ROM

Herbert Bernstein

Mikrocontrollerprogrammierung in Assembler und C

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-8266-8313-8
1. Auflage 2013

E-Mail: kundenbetreuung@hjr-verlag.de

Telefon: +49 6221/489-555
Telefax: +49 6221/489-410

www.mitp.de

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Satz: III-satz, Husby, www.drei-satz.de
Coverabbildung: © Edelweiss – Fotolia.de

Inhaltsverzeichnis

Einleitung

Geräte wie PCs, Kraftfahrzeuge (Anti-Blockier-System, Anti-Schlupf-Regelung, Motormanagement nach Abgasnorm, Sicherungssysteme wie beispielsweise Airbags), Videorekorder, DVD-Player, Kopierer, Kommunikationsgeräte, Radios, Fernseher, Heizanlagen, Kranken-, Personal- und Firmenausweise usw. enthalten eine ständig wachsende Menge an Steuerungs- und Regelungssoftware in Verbindung mit Mikrocontrollern.

Aber was verbirgt sich hinter all diesen Anwendungen? Es sind mehr oder weniger leistungsfähige Mikrocontroller, die in den letzten Jahren aufgrund immer höherer Integrationsdichte und Leistungsfähigkeit sowie sinkender Preise in immer mehr Gebiete des täglichen Lebens Einzug halten. Durch die Simulationssoftware Multisim von National Instruments ist es möglich, Hardware und Software in Assembler und C zu erstellen. Für die Entwicklung der Hardware stehen alle erdenklichen Hilfsmittel (umfangreiche Messgeräte der analogen und digitalen Technik) zur Verfügung.

Eine reine Beschreibung der einzelnen Befehle und Peripherie-Funktionen alleine genügt nicht. Deshalb wird in diesem Buch in zahlreichen Applikationen das Zusammenspiel der einzelnen Steuerbits und -worte anhand praktischer Anwendungen dargestellt. Natürlich kann und soll dieses Buch keine fundamentierte Ausbildung ersetzen – es ist vielmehr als zusätzliches – deutschsprachiges – Hilfsmittel während und nach der Ausbildung gedacht. Besonders eignet sich wegen seiner Einfachheit, Maschinennähe und Schnelligkeit der Assembler C51.

Die Komplexität der Software erfordert neue Wege in der Programmierung der Geräte. Die immer größeren Programme verlangen die Verwendung von Hochsprachen. Mit der standardisierten Hochsprache C können auch Nichtprogrammierer ein Programm für einen Mikrocontroller entwickeln. Die Sprache Assembler wird mehr und mehr aus diesem Bereich der Programmierung verdrängt. Bedingt durch den Umstieg von Assembler auf C bzw. dem Neueinstieg in die Entwicklung mit C kommen vielfältige Probleme auf den Programmierer zu. Ziel ist es, diese Probleme zu lösen und eine Praxis-orientierte Beschreibung der wichtigen Schritte der C-Softwareentwicklung zu geben.

Über dieses Buch

Dieses Buch richtet sich an Einsteiger für die Programmierung des bekannten Mikrocontrollers 8051 in Assembler und C. Zuerst werden in Kapitel 1 die Simulation des Mikrocontrollers beschrieben und kleine Aufgaben von der Problemanalyse bis zur Programmerstellung mit dem simulierten Betriebssystem des Mikrocontrollers durchgearbeitet, denn schließlich haben der 8051 und seine Nachfolger einen Marktanteil von über 70 %. Anschließend wird an zahlreichen einfachen Beispielen gezeigt, wie mit Hilfe eines Cross-Assemblers die Programmerstellung und Programmdokumentation erleichtert wird. Die erforderlichen Kenntnisse auf dem Gebiet der Hardware und Software werden ineinandergreifend erklärt und dargestellt. Das Buch ist speziell für den Unterricht geschrieben, da man alles mit Multisim simulieren kann.

Kapitel 2 behandelt die Programmierung eines Mikrocontrollers in Assembler. Dabei helfen Ihnen die zahlreichen Möglichkeiten von Multisim, einem Programm von National Instruments. Sie bauen um den Mikrocontroller per Simulation die entsprechende Hardware auf. Sie lernen die Erweiterungen des Mikrocontrollers durch Schnittstellen und Speichereinheiten kennen und arbeiten mit mehreren Oszilloskopen und Logik-Analysatoren. Danach erstellen Sie das Assemblerprogramm. Auf der CD finden Sie einen entsprechenden Assembler.

Im Kapitel 3 finden Sie zahlreiche Programmierbeispiele des Assemblers und damit können Sie den Mikrocontroller programmieren. Alle Assemblerbeispiele im Buch sind auch auf der CD vorhanden, damit Sie eine Vorlage für die Kontrolle Ihres Programmes haben.

In Kapitel 4 sind die Programmierbeispiele für die Programmiersprache C vorhanden. Neben der Simulation können Sie auch die Hardware umkonfigurieren und weitere Programmierungen vornehmen.

Mnemonik des Mikrocontrollers

Sie lernen die Mnemonik des Mikrocontrollers kennen. Das vermeintliche Problem bei der Auswahl der Mnemonik, der Sprache der Mikrocontroller, besteht darin, dass nicht alle Befehle »offensichtliche« Namen besitzen. Bei einigen Befehlen ist das der Fall (z.B. `ADD`, `AND`, `NOR`), andere bestehen aus offensichtlichen Abkürzungen (z.B. `SUB` für Subtraktion, `XOR` für Exklusiv-ODER), während dies bei anderen überhaupt nicht der Fall ist. Die meisten Hersteller verwenden brauchbare Namen, zum Teil aber auch sehr unglückliche Bezeichnungen. Anwender, die sich ihre eigenen Mnemoniks ausdenken, werden wahrscheinlich kaum bessere Bezeichnungen als der Hersteller finden.

Zusammen mit den Befehls-Mnemoniks wird ein Hersteller gewöhnlich auch den CPU-Registern entsprechende Namen zuweisen. Ebenso wie bei den Befehlsnamen besitzen einige Register offensichtliche Bezeichnungen (z.B. A oder ACC für Akkumulator), während andere mehr historische Bedeutung haben.

Assemblersprache und Assemblerprogramm

Wie bekommt man nun das Programm in Assemblersprache in den Computer? Man muss es übersetzen, entweder in Hexadezimal- oder in Binärzahlen. Man kann ein Programm in Assemblersprache manuell übersetzen, Befehl für Befehl. Die meisten Mikrocontroller machen die Aufgaben noch komplizierter, indem sie verschiedene Befehle mit unterschiedlichen Wortlängen besitzen. Manche Befehle sind nur ein Wort lang, während andere eine Länge von zwei oder drei Worten besitzen. Einige Befehle benötigen Daten im zweiten und dritten Wort. Andere benötigen Speicheradressen, Registernummern oder andere Informationen. All das wird anhand von simulierten Beispielen, die sich auf der CD befinden, ausführlich erklärt.

Die Assemblierung ist eine weitere routinemäßige Aufgabe, die man einem Mikrocontroller überlassen kann. Der Mikrocontroller führt keine Fehler beim Übersetzen von Codes aus. Er weiß immer, wie viele Worte und welches Format jeder Befehl benötigt und die Simulation zeigt direkt den Speicherbedarf und Speicherplatz an. Ein Programm, das eine derartige Aufgabe ausführt, wird als Assembler definiert. Das Assemblerprogramm übersetzt ein Anwender- oder Quellenprogramm (auch als Quellprogramm bezeichnet), das mit Mnemoniks geschrieben wurde, in ein Programm in Maschinensprache (oder Objektprogramm), das der Mikrocontroller ausführen kann. Die Eingabe für den Assembler ist ein Quellenprogramm und er gibt ein Objektprogramm aus. Bei Benützung des Wortes »Assembler« ist jeweils darauf zu achten, ob nur die Assemblersprache oder das Assemblerprogramm gemeint ist.

Assembler besitzen eigene Regeln, die man erlernen muss. Diese enthalten die Verwendung bestimmter Markierungen oder Kennzeichen (wie Zwischenräume, Kommata, Strichpunkte oder Doppelpunkte) an den entsprechenden Stellen, korrekte Aussprache, die richtige Steuerinformation und vielleicht auch die ordnungsgemäße Platzierung von Namen und Zahlen. Diese Regeln stellen nur ein kleines Hindernis dar, das man leicht überwinden kann.

Frühere Assemblerprogramme leisteten wenig mehr als die Übersetzung der Mnemoniks der Befehle und Register in ihr binäres Äquivalent. Die meisten neueren Assemblerprogramme besitzen zusätzliche Eigenschaften:

- Sie gestatten dem Anwender die Zuweisung von Namen zu Speicherplätzen, Eingabe- und Ausgabebausteinen und sogar vollständigen Befehlssequenzen.

- Sie wandeln Daten oder Adressen von verschiedenen Zahlensystemen (z.B. dezimal oder hexadezimal) in binäre und Zeichen in ihre ASCII- oder EBCDIC-Binärcodes um.
- Sie führen einige arithmetische Operationen als Teil des Assembliervorgangs aus.
- Sie teilen dem Laderprogramm mit, wohin Teile des Programmes oder Daten in den Speicher platziert werden sollen.
- Sie gestatten dem Anwender die Zuweisung von Speicherbereichen für zeitweilige Datenspeicherung und die Platzierung fester Daten in bestimmte Bereiche des Programmspeichers.
- Sie liefern Informationen, die erforderlich sind, um Standardprogramme aus einer Programmbibliothek oder Programme, die zu einer anderen Zeit geschrieben wurden, in das momentane Programm aufzunehmen.
- Sie gestatten dem Anwender die Steuerung des Formats der Programmauflistung und der verwendeten Eingabe- und Ausgabebausteine.

Alle diese Eigenschaften bedingen natürlich zusätzliche Kosten und Speicher. Mikrocontroller haben im Allgemeinen wesentlich einfachere Assembler als große Computersysteme mit Mikroprozessoren, tendieren jedoch dazu, immer größer zu werden. Man besitzt häufig eine Auswahl an verschiedenen Assemblerprogrammen. Das wichtigste Kriterium hierbei ist nicht, wie viele ausgefallene Eigenschaften der Assembler besitzt, sondern wie bequem er bei der Anwendung in der Praxis ist. Mit dem auf der CD vorhandenen Assembler kann man ohne Probleme im Unterricht oder im Selbststudium arbeiten.

Der Assembler löst keinesfalls die gesamten Probleme des Programmierens, genauso wenig wie der Hexadezimal-Lader. Ein Problem besteht in der gewaltigen Lücke zwischen dem Befehlssatz des Mikrocontrollers und den Aufgaben, die der Mikrocontroller auszuführen hat. Computerbefehle führen meist Dinge aus, wie das Addieren des Inhalts zweier Register, Verschieben des Inhalts des Akkumulators um ein Bit oder das Platzieren eines neuen Wertes in den Befehlszähler. Auf der anderen Seite möchte der Anwender eines Mikrocontrollers diesem aber auch überlassen, zu prüfen, ob eine analoge Ablesung einen bestimmten Wert überschritten hat, nach einem bestimmten Kommando eines Druckers Ausschau zu halten und darauf zu reagieren oder zu checken, ob ein Relais zur richtigen Zeit aktiviert ist. Ein Programmierer, der die Assemblersprache verwendet, muss derartige Aufgaben in eine Folge (oder Sequenz) einfacher Mikrocontrollerbefehle umwandeln. Diese Umwandlung kann eine schwierige und zeitraubende Aufgabe darstellen, aber nicht mit dieser Simulation.

Ferner muss man beim Programmieren in Assemblersprache eine sehr detaillierte Kenntnis des verwendeten speziellen Mikrocontrollers besitzen. Man muss wissen, welche Register und Befehle der Mikrocontroller hat, wie die Befehle die

verschiedenen Register genau beeinflussen, welche Adressierverfahren der Computer verwendet und eine Unmenge weiterer Informationen. Keine dieser Informationen ist für die Aufgabe, die der Mikrocontroller letztlich ausführen muss, relevant.

Programme in Assemblersprache sind nicht übertragbar. Jeder Mikrocontroller eines Herstellers besitzt seine eigene Assemblersprache, die durch seine Architektur bestimmt wird. Ein Programm in Assemblersprache, das für den 8051 geschrieben wurde, lässt sich nicht auf einen anderen Mikrocontroller eines anderen Herstellers übertragen. Da die Programme nicht übertragbar sind, bedeutet dies, dass man das Programm in Assemblersprache nicht auf einem anderen Mikrocontroller verwenden kann. Das bedeutet auch weiter, dass man nicht imstande ist, irgendein Programm zu verwenden, das nicht für Mikrocontroller speziell geschrieben wurde. Daraus ergibt sich häufig, dass man auf sich selbst angewiesen ist. Wenn man ein Programm für die Ausführung einer speziellen Aufgabe benötigt, wird man dies wahrscheinlich nicht in den kleinen Programmbibliotheken der Hersteller finden. Man wird es auch wahrscheinlich kaum in einem Archiv, Zeitungsartikel oder einer alten Programmbibliothek finden und so muss man sich sein Programm selbst schreiben.

Vor- und Nachteile »höherer« oder »Problem-orientierter« Sprachen

Die Lösung für viele der mit Assemblersprachen-Programmen verbundenen Schwierigkeiten ist die Verwendung von »höheren« oder »Problem-orientierten« Sprachen. Derartige Sprachen gestatten die Beschreibung von Aufgaben in einer Art und Weise, die eher Problem-orientiert als Computer-orientiert ist. Jede Anweisung in einer höheren Programmiersprache führt eine erkennbare Funktion aus. Sie wird im Allgemeinen mehreren Befehlen in Assemblersprache entsprechen. Ein Programm, das Compilierer (oder Compiler) genannt wird, übersetzt ein Quellenprogramm in eine höhere Programmiersprache in Befehle im Objektcode oder Maschinensprache.

Es existieren mehrere unterschiedliche höhere Sprachen für verschiedene Arten von Aufgaben. Offensichtlich erleichtern höhere Sprachen das Programmieren und beschleunigen die Erstellung eines Programms. Man kann etwa sagen, dass ein Programmierer ein Programm zehnmal schneller in einer höheren Sprache als in der Assemblersprache schreiben kann. Dies betrifft nur das Schreiben des Programms. Es enthält nicht die Definition des Problems, Entwicklung des Programms, Fehlersuche, Testen oder Dokumentation, was ebenfalls einfacher und schneller wird. Das Programm in der höheren Sprache ersetzt beispielsweise zum Teil eine entsprechende Dokumentation.

Höhere Sprachen lösen viele andere Probleme, die mit der Programmierung in Assemblersprache verknüpft sind. Die höhere Sprache besitzt ihre eigene Syntax (gewöhnlich definiert durch einen nationalen oder internationalen Standard). In der Sprache werden Befehlssatz, die Register oder andere Eigenschaften eines speziellen Computers nicht erwähnt. Auf alle derartige Details muss der Compiler achten. Die Programmierer können sich auf ihre eigentliche Aufgabe konzentrieren. Sie brauchen die verwendete CPU-Architektur nicht im Detail zu verstehen, das heißt, sie müssen nicht einmal etwas über den Computer wissen, den sie programmieren.

Programme, die in einer höheren Sprache geschrieben wurden, sind übertragbar; zumindest in der Theorie.Sie werden auf jedem beliebigen Computer laufen, der einen Standardcompiler für diese Sprache besitzt.Dadurch sind aber gleichzeitig alle früher in einer höheren Sprache geschriebenen Programme für bestimmte Computer auch verfügbar, wenn man ein neues System mit Mikrocontroller programmieren muss. Dies kann im Falle der früher gebräuchlichsten Sprachen wie FORTRAN oder BASIC bedeuten, dass man Tausende von Programmen zur Verfügung hat.

Wenn nun aber alle erwähnten Vorteile, die bei höheren Programmiersprachen aufgezählt wurden, wahr sind, wenn man Programme schneller schreiben kann und diese außerdem übertragbar sind, warum quält man sich dann überhaupt noch mit Assemblersprachen? Wer würde sich noch mit Registern, Befehlscodes, Mnemoniks und all diesem Ballast herumschlagen? Wie gewöhnlich steht allen Vorteilen meist auch eine entsprechende Anzahl von Nachteilen gegenüber.

Ein offensichtliches Problem liegt darin, dass man die »Regeln« oder die »Syntax« jeder höheren Programmiersprache, die man verwenden will, erlernen muss. Eine höhere Sprache besitzt einen ziemlich komplizierten Satz von Regeln. Man wird finden, dass man viel Zeit benötigt, ein Programm zu erhalten, das syntaktisch korrekt ist, und sogar dann wird es wahrscheinlich noch nicht das tun, was man will. Eine höhere Computersprache ist wie eine Fremdsprache. Wenn man ein wenig Talent besitzt, wird man rasch mit den Regeln vertraut sein und imstande sein, Programme zu liefern, die der Compiler annimmt. Aber mit dem Lernen der Regeln und dem Versuch, das Programm für den Compiler annehmbar zu gestalten, ist die Aufgabe noch lange nicht gelöst.

Ein weiteres Problem liegt darin, dass man einen Compiler zur Übersetzung der in einer höheren Sprache geschriebenen Programme in Maschinensprache benötigt. Compiler sind aufwendig und benötigen große Speicher. Während die meisten Assembler für Mikrocontroller einen Speicher bis maximal 2 Kbyte belegen (1 Kbyte = 1024 Bits), benötigt ein Compiler gewöhnlich wesentlich mehr Speicherplatz. Daher liegen die Kosten bei der Verwendung eines Compilers wesentlich höher.

Ferner machen nicht alle Compiler unsere Aufgabe tatsächlich leichter. Wenn man jedoch die Aufgabe durchzuführen hat, einen Drucker zu steuern, eine Kette von Zeichen zu editieren oder ein Alarmsystem zu überwachen, kann die Aufgabe nicht ohne Weiteres in algebraischer Schreibweise dargestellt werden. In der Tat kann die Formulierung der Lösung in algebraischer Schreibweise wesentlich mühsamer und schwieriger sein als ihre Formulierung in Assemblersprache. Die naheliegende Antwort wäre natürlich die Verwendung einer besser geeigneten höheren Sprache.

Höhere Sprachen ergeben häufig wenig effiziente Programme in Maschinensprache. Der wesentliche Grund hierfür ist, dass die Compilierung einen automatischen Vorgang darstellt, der voll mit Kompromissen für die Ausführung zahlreicher Möglichkeiten ist. Der Compiler arbeitet ähnlich einem computergesteuerten Sprachübersetzer, bei dem die Worte manchmal stimmen, aber der Klang und der Satzbau wirkt äußerst unbeholfen.

Ein einfacher Compiler kann nicht wissen, wann eine Variable nicht länger verwendet wird und gelöscht werden kann, wann ein Register besser anstelle eines Speicherplatzes verwendet werden soll oder wann Variablen einfache Beziehungen untereinander besitzen. Der erfahrene Programmierer kann die Vorteile von Abkürzungen verwenden, um die Ausführungszeit zu verringern oder die Verwendung des Speichers zu reduzieren. Einige wenige Compiler (bekannt als optimierende Compiler) können dies ebenfalls, aber derartige Compiler sind wesentlich größer und langsamer als gewöhnliche Compiler.

Die allgemeinen Vor- und Nachteile höherer Programmiersprachen sind:

Vorteile:

- Bequemer für die Beschreibung von Aufgaben
- Größere Produktivität des Programmierers
- Einfachere Dokumentation
- Standardisierte Syntax
- Unabhängigkeit von der Struktur des speziellen Computers
- Übertragbarkeit
- Verfügbarkeit von Bibliotheken und anderen Programmen

Nachteile:

- Spezielle Regeln
- Weitgehende Hardware- und Softwareunterstützung erforderlich
- Orientierung der gebräuchlichen Sprachen auf algebraische oder kaufmännische Probleme
- Ineffiziente Programme

- Schwierigkeit bei der Optimierung des Codes für zeitliche und Speicheranforderungen
- Unmöglichkeit der bequemen Verwendung spezieller Eigenschaften eines Mikrocontrollers

Die Anwender von Mikrocontrollern werden auf verschiedene spezielle Schwierigkeiten stoßen, wenn sie höhere Programmiersprachen verwenden. Diese sind unter anderem:

- Es existieren wenige höhere Programmiersprachen für Mikrocontroller.
- Es sind keine Standardsprachen allgemein verfügbar.
- Wenige Compiler laufen tatsächlich auf Mikrocontrollern. Die es tun, benötigen oft sehr großen Speicherraum.
- Die meisten Mikrocontroller-Anwendungen sind nicht besonders gut für höhere Programmiersprachen geeignet.
- Speicherkosten sind in Mikrocontroller-Anwendungen nicht mehr kritisch.

Weil die wenigen existierenden höheren Sprachen häufig nicht den anerkannten Standards entsprechen, kann der Mikrocontroller-Anwender nicht erwarten, dass viele Programme übertragbar sind, er Zugriff zu Programmbibliotheken erhält oder frühere Erfahrungen oder Programme verwenden kann. Die verbleibenden Hauptvorteile liegen daher in der Verringerung des Programmieraufwandes und des geringeren erforderlichen detaillierten Verständnisses der Computerarchitektur.

Die mit der Verwendung höherer Programmiersprachen bei Mikrocontrollern verbundenen Kosten sind beträchtlich. Mikrocontroller sind für Steuerungen und andere Anwendungen besser geeignet als für Zeichenmanipulationen und Sprachanalyse bei der Compilierung. Daher werden die meisten Compiler für Mikrocontroller nicht auf einem System laufen, das auf einem Mikrocontroller basiert. Sie erfordern stattdessen einen wesentlich größeren Computer, das heißt, sie sind eher Cross-Compiler, die nicht auf dem Mikrocontroller laufen, für den das zu compilierende Programm geschrieben ist, als Selbst-Compiler. Ein Anwender muss daher nicht nur die Aufwendungen für ein größeres Mikrocontrollersystem tragen, er muss auch das Programm physisch vom Entwicklungssystem zum Mikrocontroller übertragen. Alles das übernimmt die Simulation.

Es sind einige wenige Selbst-Compiler verfügbar. Diese Compiler laufen auf dem Mikrocontroller, für den sie den Objektcode erzeugen. Unglücklicherweise benötigen sie jedoch sehr viel Speicherplatz (16 Kbyte oder mehr) sowie spezielle unterstützende Hard- und Software.

Höhere Programmiersprachen sind auch im Allgemeinen für Mikrocontroller-Anwendungen nicht gut geeignet. Die meisten der bekannten Sprachen sind entweder zur Lösung wissenschaftlicher Probleme oder für die Handhabung großer

Datenmengen in der kommerziellen Informationsverarbeitung vorgesehen. Wenige Mikrocontroller-Anwendungen fallen in diese Bereiche. Die meisten Mikrocontroller-Anwendungen beinhalten das Senden von Daten und Steuerinformationen zu Ausgabebausteinen und das Empfangen von Daten und Statusinformationen von Eingabebausteinen. Häufig bestehen die Steuer- und Statusinformationen aus einigen wenigen binären Ziffern mit sehr genauen Bedeutungen bezüglich der Hardware. Würde man versuchen, ein typisches Steuerprogramm in einer höheren Sprache zu schreiben, würde man sich häufig so fühlen, als versuchte man, eine Suppe mit Stäbchen zu essen. Für Aufgaben wie etwa Testgeräte, Terminals, Navigationssysteme und Bürogeräte arbeiten die höheren Sprachen wesentlich besser, als dies bei Anwendungen in der Messtechnik, Kommunikation, Steuerung von peripheren Bausteinen und im Automobil der Fall wäre.

Besser geeignete Anwendungen für höhere Sprachen sind jene, in denen große Speicher erforderlich sind. Wenn die Kosten eines einzelnen Speicherchips wesentlich sind, wie dies zum Beispiel in einem Steuergerät, einem elektronischen Spiel-, Haushalts- oder kleineren Messgerät der Fall ist, dann ist die Ineffizienz der höheren Sprache nicht mehr tragbar. Wenn andererseits das System mehrere tausend Bytes eines Speichers besitzt, wie dies in einem größeren Mess- oder Testgerät der Fall ist, dann ist die Ineffizienz der höheren Sprachen nicht so wesentlich. Natürlich sind die Größe des Programms und die produzierte Stückzahl des betreffenden Gerätes ebenfalls wesentliche Faktoren. Ein großes Programm wird die Vorteile der höheren Sprachen entsprechend nutzen. Andererseits werden bei einer Anwendung mit hohen Stückzahlen die festen Software-Entwicklungskosten nicht so wesentlich sein wie die Speicherkosten, die ein Teil jedes Systems sind.

Welches Niveau man verwenden sollte, hängt von der speziellen Anwendung ab. Es sollen kurz die Faktoren zusammengefasst werden, die für ein spezielles Programmierniveau den Ausschlag geben:

- **Maschinensprache:** Ein Programm in Maschinensprache ist tatsächlich völlig unwirtschaftlich. Bei den niedrigen Kosten eines Assemblers ist seine Anwendung nicht gerechtfertigt.
- **Assemblersprache** für kleine bis mittlere Programme (bis 2 Kbyte) und Anwendungen, bei denen die Speicherkosten ein wesentlicher Faktor sind:
  - Echtzeit-Steueranwendungen
  - Begrenzte Datenverarbeitung
  - Anwendungen mit hohen Stückzahlen
- **Höhere Sprachen**, wie C:
  - für große Programme

- für Anwendungen mit kleiner Stückzahl, die jedoch lange Programme benötigen
- für Anwendungen, die große Speicher erfordern
- für Anwendungen, in denen mehr Berechnungen als Eingaben/Ausgaben oder Steuerung vorliegen
- bieten Kompatibilität mit ähnlichen Anwendungen, die größere Computer verwenden
- bieten Verfügbarkeit von speziellen Programmen, die in der vorliegenden Anwendung verwendet werden können

Viele andere Faktoren sind ebenfalls wichtig, wie etwa die Verfügbarkeit eines größeren und speziellen Entwicklungssystems für die Hardwareentwicklung, Erfahrung mit speziellen Sprachen und Kompatibilität mit anderen Anwendungen.

Wenn schließlich die Hardware die wesentlichsten Kosten in unserer Anwendung darstellt oder die Geschwindigkeit kritisch ist, sollte auf jeden Fall die Assemblersprache bevorzugt werden. Man muss jedoch damit rechnen, dass man zusätzliche Zeit für die Entwicklung der Software aufwenden muss, um dafür niedrigere Speicherkosten und höhere Ausführungsgeschwindigkeit zu erhalten. Wenn die Software die größten Kosten in unserer Anwendung darstellt, so sollte man höhere Sprachen bevorzugen. Aber es sind zusätzliche Kosten für die Hard- und Software erforderlich.

Natürlich ist auch die Verwendung sowohl der Assemblersprache als auch höherer Programmiersprachen möglich. Man kann das Programm zunächst in einer höheren Sprache schreiben und dann einzelne Abschnitte durch Assemblersprache ersetzen. Das wird jedoch in der Praxis nicht sehr häufig durchgeführt, da hierdurch gewaltige Schwierigkeiten bei der Fehlersuche, dem Testen und der Dokumentation entstehen.

Es ist zu erwarten, dass in Zukunft eine Tendenz in Richtung höherer Programmiersprachen aus folgenden Gründen vorhanden sein wird:

- Programme scheinen immer aufwendiger und größer zu werden.
- Mikrocontroller, Schnittstellen und Speicher werden preiswerter.
- Software und Programmierer gewinnen an Erfahrung.
- Speicherchips bekommen mehr Kapazität bei niedrigeren Kosten »pro Bit«, so dass die tatsächlichen Einsparungen an Speicherkosten nicht mehr so stark ins Gewicht fallen.
- Es werden geeignetere und effizientere höhere Programmiersprachen entwickelt.
- Höhere Programmiersprachen werden weiter standardisiert.

Die Programmierung von Mikrocontrollern in Assemblersprache ist jedoch keine aussterbende Kunst, wie es derzeit bei größeren Computern und PCs der Fall ist. Jedoch werden längere Programme, billigere Speicher und teurere Programmierer einen immer größeren Anteil der Softwarekosten bei den meisten Applikationen zur Folge haben. Bei zahlreichen Anwendungen werden daher höhere Programmiersprachen Verwendung finden.

Wenn die Zukunft scheinbar den höheren Programmiersprachen gehört, wozu dient dann Kapitel 3 in diesem Buch über Programmierung in Assemblersprache? Die Gründe sind:

- Die meisten gegenwärtigen Anwender von Mikrocontrollern programmieren in Assemblersprache (mindestens 80 % gemäß einer kürzlich erfolgten Umfrage).
- Viele Anwender von Mikrocontrollern werden weiter in Assemblersprache programmieren, da sie deren detaillierte Steuerung benötigen.
- Bis jetzt ist noch keine allgemein verfügbare oder standardisierte höhere Programmiersprache entwickelt worden.
- Zahlreiche Anwendungen erfordern die Effizienz der Assemblersprache.
- Das gründliche Verständnis der Assemblersprache kann bei der Entwicklung höherer Sprachen helfen.

Kapitel 3 dieses Buches befasst sich ausschließlich mit Assemblern und Programmierung in Assemblersprache. Es ist jedoch wünschenswert, dass die Leser wissen, dass die Assemblersprache nicht die einzige Alternative ist. Man sollte sorgfältig neue Entwicklungen beobachten, die möglicherweise eine wesentliche Reduzierung der Programmierkosten bewirken, falls derartige Kosten ein wesentlicher Faktor für die jeweilige Anwendung sind.

Programmiersprache C

Kapitel 4 dieses Buches wird C gewidmet. Hinsichtlich des Umfangs und der sprachlichen Möglichkeiten ist C eine relativ »kleine« Sprache. Beispielsweise sind weder E/A-Anweisungen Bestandteil von C noch existieren Mittel zur Zeichenkettenverarbeitung. In den meisten C-Programmierumgebungen stehen jedoch viele dieser Dinge in Form von Bibliotheks- bzw. Systemfunktionen bereit. Diese Beschränkung im Sprachumfang ist eine wesentliche Ursache dafür, dass C mit relativ geringem Aufwand implementiert werden kann, insbesondere auch auf Mikrosystemen mit Mikrocontroller.

In gewissem Sinne stellt C einen Kompromiss zwischen einer typischen höheren Programmiersprache (wie beispielsweise PASCAL) und der Assemblersprache der jeweiligen Mikrocontroller dar. Auf der einen Seite wird die Entwicklung gut struk-

turierter und portabler Software unterstützt, andererseits bietet C eine leistungsfähige Schnittstelle zur Hardware sowie vielfältige und äußerst flexibel verwendbare Mittel zur Entwicklung kompakter sowie code- und laufzeiteffizienter Programme. Während die Lösung insbesondere systemnaher Aufgabenstellungen in anderen Programmiersprachen zum Teil erhebliche »Klimmzüge« erfordert, um durch die Sprache vorgegebene Restriktionen zu umgehen, ist C durch Freizügigkeit bei der Wahl algorithmischer Strukturen und beim Zugriff auf Datenobjekte gekennzeichnet. Diese Entwurfsstrategie bedingt aber auch, dass dem Programmierer ein hohes Maß an Verantwortung überlassen bleibt. Vor allem dem Neuling wird empfohlen, die zur Verfügung stehenden sprachlichen Mittel sehr sorgsam und gezielt einzusetzen. Ein disziplinierter Programmierstil und Erfahrungen im Umgang mit C sind notwendig, da andernfalls die Gefahr besteht, dass fehleranfällige und wenig übersichtliche Programme entstehen. Obwohl man die Sprache C relativ leicht erlernen kann, ist sie aus oben genannten Gründen nur bedingt für den Anfänger auf dem Gebiet der Programmierung geeignet.

Die Programmiersprache C befindet sich in einer ständigen Weiterentwicklung. Lange Zeit galt der beschriebene Sprachumfang als De-facto-Sprachstandard. Viele der verfügbaren Compiler berücksichtigen nachträgliche Erweiterungen, wie z.B. den Aufzählungstyp (`enum`) und die Strukturzuweisung. Grundlage für den in diesem Buch gewählten Sprachumfang ist die veröffentlichte C-Sprachdefinition.

Der erste Teil von Kapitel 4 gibt eine Einführung in die grundlegenden Sprachelemente von C. Behandelt werden die Grunddatentypen und ihre Realisierung als variable und konstante Werte, die einfachen arithmetischen Operatoren und Ausdrücke sowie die elementaren Steuerstrukturen, soweit sie mit einer vergleichbaren Semantik auch in anderen Programmiersprachen existieren. Ein Teil von Kapitel 4 enthält auch die Erläuterung der Grundlagen zu Funktionen und zur Programmstruktur. Auf dieser Basis werden einfache, aber vollständige Beispielprogramme gezeigt und somit alle notwendigen Kenntnisse vermittelt, um möglichst schnell selbst praktische Programmierübungen vornehmen zu können.

Ein Teil von Kapitel 4 behandelt die für C charakteristischen Sprachaspekte. Dabei werden die vielfältigen Operatoren erklärt, die strukturierten Datentypen und die Arbeit mit Zeigern erläutert sowie zur Programmlaufsteuerung vervollständigt. Der Abschnitt über Funktionen fasst alle damit im Zusammenhang stehenden Details zusammen. Die restlichen Abschnitte sind speziellen Fragen gewidmet, z.B. den Problemen der Speicherklasse von Variablen, Funktionen und Parametern, den Möglichkeiten der Initialisierung von Variablen sowie den Aspekten der Typkonvertierung. Die Diskussion dieser Probleme wird so lange wie möglich vertagt, um alle damit verbundenen Gesichtspunkte im Zusammenhang erläutern zu können. Sie erfolgt, wenn abschließend einige etwas komplexere Beispiele zu ausgewählten Aufgabenstellungen vorgestellt werden.

Zunächst werden die Konventionen vorgestellt, nach denen der Aufruf von C-Programmen erfolgt. Zu jedem C-Compilersystem gehört ein Präprozessor, der – unabhängig von der eigentlichen Sprache – u.a. Möglichkeiten zur Bildung von Makros und zur bedingten Compilierung bietet. Das betrifft z.B. die Anwendung der Systemfunktionen zur Ein- und Ausgabe und zur Prozesssteuerung. Es sollen verschiedene Aspekte der C-Programmierung verdeutlicht und so zur Beherrschung der sprachlichen Mittel beigetragen werden.

Für das Erlernen der Programmiersprache C wird empfohlen, die einzelnen Abschnitte bis einschließlich der Behandlung der Standardbibliothek sequenziell zu studieren und durch praktische Übungen zu ergänzen. Mit den Problemen der systemnahen Programmierung und den abschließenden Beispielen kann sich der Leser in beliebiger Reihenfolge und je nach Interesse auseinandersetzen.

Über die Buch-CD:

Multisim können Sie über das Internet unter `www.ni.com/multisim/try/d/` erhalten. Beim Herunterladen fehlen jedoch die Tools vom Assembler und C. Sie befinden sich auf der CD und können von dort auf den PC überspielt werden.

Auf der CD befinden sich auch zahlreiche Beispiele in Assembler und in C:

Wenn bei einer Abbildung im Buch in Klammern z.B. *mc3.1* steht, können Sie unter dieser Bezeichnung das fertige Beispiel auf der CD aufrufen. Wenn Sie `mc3.1` öffnen, erscheint die Schaltung mit dem Mikrocontroller 8051, die Taste und die Leuchtdiode.

Wenn Sie den Ein/Ausschalter links im Bildschirm anklicken, starten Sie die Simulation, das heißt, Sie können mit der Taste »Leerzeichen« die Leuchtdiode ein- oder ausschalten.

Über `MCU` und `MCU8051` erhalten Sie den Code-Manager, die Ansicht vom Debugger und die Speicheransicht. Über den Code-Manager sehen Sie die Verwaltung des Programmbeispiels und über den Debugger kommen Sie direkt in das Programm und sehen den gesamten Aufbau mit den Speicherzellen und den Befehlen in Assembler. Es stehen Ihnen auch zahlreiche Möglichkeiten der Programmierung und der Fehlersuche zur Verfügung. Mit der Speicheransicht sehen Sie das Spezial-Funktionsregister, die Belegung des IRAM, IROM und XRAM.

Danksagung

Meiner Frau Brigitte danke ich für die Erstellung der Zeichnungen und ein besonderer Dank gilt Herrn Gerhard Zwilling.

München, Januar 2013, Herbert Bernstein

Mikrocontroller-8051-Familie und AT89C51

Was haben Analog Devices, Atmel, Cypress Semiconductor, Dallas Semiconductor, Goal, Hynix, Infineon, Intel, OKI, Philips, Silicon Labs, SMSC, STMicroelectronics, Synopsis, TDK, Temic, Texas Instruments und Winbond gemein? Sie alle bieten 8051-basierte Mikrocontrollerbausteine bzw. IP-Cores an!

Aufgrund der großen Verbreitung dieser Mikroprozessorfamilie und damit verbunden mit großen Softwarebibliotheken gibt es auch eine Vielzahl von synthetisierbaren Implementierungen. Diese sind als so genannte IP-Cores in einer Hardwarebeschreibungssprache wie beispielsweise VHDL frei und im Quelltext verfügbar. Sie eignen sich für den Einsatz in FPGAs und anwendungsspezifischen integrierten Schaltungen (ASICs).

1.1 Einführung

Seit der Einführung des 8-Bit-Mikrocontrollers 8048 im Jahre 1976 von Intel, des ersten Ein-Chip-Mikrocontrollers auf einem einzigen integrierten Baustein, ist diese so vielseitig einsetzbare Schaltung durch Entwicklung einer Reihe ähnlicher Bausteine mit unterschiedlichen Zielsetzungen entwickelt worden. Zum Beispiel wurde beim Mikrocontroller 8049 sowohl die Programm- als auch die Datenspeicherkapazität gegenüber dem 8048 (oder seiner EPROM-Version 8748) verdoppelt. Für Anwendungen, bei denen nur externe Programmspeicherkapazität erforderlich ist, stehen die Mikrocontroller 8035 und 8039 zur Verfügung. Der kostengünstige Mikrocontroller 8021 ist für Anwendungen vorgesehen, die mit einer geringeren Anzahl von Ein/Ausgabeleitungen auskommen und er arbeitet bei niedrigerer Geschwindigkeit mit einer Teilmenge des 8048-Befehlsvorrats.

Der Mikrocontroller 8051 und seine Nachfolger sind inzwischen eine industrielle Standardschaltung geworden, die von zahlreichen Halbleiterherstellern angeboten und weltweit eingesetzt werden. Von beiden Familien haben die Halbleiterhersteller zahlreiche Schaltkreise in der CMOS-Version zur Verfügung gestellt und entwickelt und CMOS-Versionen aller anderen Schaltungen sowie weitere Mikrocontroller, die sich durch zusätzliche anwenderorientierte Funktionen auszeichnen.

Bei allen Herstellern sind zahlreiche Mikrocontroller mit stark erhöhter Integrationsdichte verfügbar, die aufgrund ihrer hohen Leistungsfähigkeit neue Anwen-

dungsbereiche erschließt. Die Leistungsfähigkeit dieses Mikrocontrollers 8051 ist gegenüber dem 8048 beträchtlich gesteigert worden, indem ca. 60 000 Transistoren auf die Chipfläche aufgebracht werden. Der 8051 gehört zur Mikrocontroller-Familie 8051, die auch die Mikrocontroller 8031 (ohne ROM) und 8751 (mit EPROM) umfasst. Die Mitglieder dieser Familie sind die Mikrocontroller 8052 und 8032, die sich durch Verdopplung der RAM-Kapazität auf 256 Bytes sowie – beim 8052 – durch Verdopplung der ROM-Kapazität auf 8 Kbyte auszeichnen (der 8032 hat kein ROM). Außerdem sind diese beiden Mikrocontroller mit einem zusätzlichen Timer mit speziellen Eigenschaften ausgerüstet.

Ab 1980 kamen mehrere Einchip-Mikrocontroller aus der Familie 8051 auf den Markt. Der 83C152 und 83C252 von Intel, der 80C154 von OKI, der 83C552 von Philips/Valvo, der 82C451 von Philips/Signetics und 80512, 80515 und 80C517 von Siemens. Alle diese Mikrocontroller sind mit einem Watchdog-Timer, 8-Bit-Analog-/Digital-Wandler, Impulsweitenmodulation und einer seriellen Schnittstelle nach I^2C ausgerüstet.

Die Einchip-Mikrocontroller-Familie hat sehr viele Mitglieder hervorgebracht, doch weisen alle mehr oder weniger gemeinsame Merkmale auf:

- 8-Bit-Prozessorkern mit einheitlichem Befehlssatz
- mindestens 128 Bytes internes RAM
- externes RAM und ROM
- einheitliches Adressierungsmodell für so genannte »Special Function Register« (SFR)
- Full-Duplex-UART
- fünf Interrupt-Quellen
- zwei Interrupt-Prioritäten
- diverse Timer

Aufgrund der unterschiedlichen Befehlslängen von einem bis zu drei Byte sowie den unterschiedlichen Ausführungszeiten für einen Befehl handelt es sich eindeutig um eine CISC-Architektur. Neben dem klassischen CISC-Mikrocontroller (complex instruction set computer), wie man diese bei den Mikroprozessoren in der PC-Technik findet, verwendet man immer mehr die modernen RISC-Mikrocontroller (reduced instruction set computer). Diese Mikrocontroller setzen einen reduzierten Befehlssatz ein, der wesentlich effizienter ist und dadurch auch erheblich schneller das Programm abarbeitet. Trotzdem hat die CISC-Architektur erhebliche Vorteile in der Ausbildung und im Studium.

Ein Befehlszyklus entspricht in der ursprünglich von Intel entwickelten Struktur einem bis drei Maschinenzyklen. Ein Maschinenzyklus entspricht zwölf Taktzyklen. Heute übliche Varianten des 8051 kommen hingegen meist mit zwei Taktzyklen pro Maschinenzyklus aus und damit ist bei gleicher Taktfrequenz eine deutlich höhere Befehlsabarbeitung möglich.

Eine Besonderheit des 8051 ist der Bitprozessor, der im bitadressierbaren Bereich eine schnelle und einfache Bitmanipulation erlaubt.

1.1.1 Unterschiede in der Familie 8051

Die in NMOS-Technologie (im Jahr 1976) gefertigten 8051 und 8031 von Intel und den anderen Halbleiterherstellern wurden ab diesem Zeitpunkt in CMOS-Technik hergestellt. Auf technische Unterschiede der NMOS- und CMOS-Versionen bezüglich der Oszillator-Beschaltung und des Betriebs bei reduzierter Leistung wird noch eingegangen. In Tabelle 1.1 sind die Mikrocontroller der Familie 8051 aufgelistet. Weitere Unterschiede der einzelnen Schaltungen sind in den folgenden Ausführungen beschrieben. Oft wird die Bezeichnung 8051 auch als Oberbegriff für diese Schaltungen verwendet.

NMOS-Varianten			
Typ	ROM intern	RAM intern	Bemerkung
8031	–	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen
8032	–	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen
8051	4096	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
8052	8192	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
8751	8192	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, EPROM als OTP oder mit UV-Fenster
CMOS-Varianten			
Typ	ROM intern	RAM intern	Bemerkung
80C31	–	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen
80C32	–	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen
80C51	4096	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
80C52	8192	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
80C54	16384	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
80C58	32768	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, maskenprogrammierbares ROM
87C51	8192	128	zwei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, EPROM als OTP oder mit UV-Fenster
87C52	8192	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, EPROM als OTP oder mit UV-Fenster
89C52	32768	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, Flash-Speicher

Tabelle 1.1: NMOS- und CMOS-Varianten der Familie 8051

Darüber hinaus gibt es von verschiedenen Firmen erweiterte 8051-Mikrocontroller, z. B. von Siemens (heute Infineon) entwickelt und in NMOS-Technologie hergestellt, wie Tabelle 1.2 zeigt.

Typ	ROM intern	RAM intern	Bemerkung
80515	8192	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, PWM, 8-fach A/D-Wandler, maskenprogrammierbares ROM
80535	–	256	drei 16-Bit-Timer, ein UART, zwei externe Interruptquellen, PWM, 8-fach A/D-Wandler

Tabelle 1.2: 8051-Familie von Siemens (heute Infineon)

Die CMOS-Varianten der Infineon-Serie sind funktionsgleich, allerdings nicht in allen Varianten pinkompatibel.

Weitere moderne 8051-kompatible Mikrocontroller, die in CMOS-Technologie mit Flash-Speicher hergestellt werden und eine schnellere Befehlsverarbeitung zulassen sind erhältlich. Tabelle 1.3 zeigt Typen von verschiedenen Herstellern.

Hersteller	Typ	Flash intern	RAM intern	Bemerkung
ATMEL	AT89C2051	2048	128	nur 20 Pins, kein externer Daten-/Adressbus, zwei 16-Bit-Timer, ein UART, ein Komparator, Flash-Speicher
ATMEL	AT89C4051	4096	128	nur 20 Pins, kein externer Daten-/Adressbus, zwei 16-Bit-Timer, ein UART, ein Komparator, Flash-Speicher
ATMEL	AT89C51ED2	16384	256	bis zu sechs Ports (je nach Gehäusegröße), 1792 Bytes internes MOVX-SRAM, RS232-Bootloader im ROM, 2048 Bytes, internes Daten-EEPROM, drei 16-Bit-Timer, ein UART, ein Komparator, zwei Stackpointer, PWM, SPI, Flash-Speicher
MAXIM/ DALLAS	DS89C430	16384	256	ein Taktzyklus pro Befehl, max. 33 MIPS bei 33 MHz, 1 Kbyte internes MOVX-SRAM, RS232-Bootloader im ROM, 2048 Bytes, internes Daten-EEPROM, drei 16-Bit-Timer, ein UART, ein Komparator, zwei Stackpointer, Flash-Speicher
MAXIM/ DALLAS	DS89C450	65536	256	ein Taktzyklus pro Befehl, max. 33 MIPS bei 33 MHz, 1 Kbyte internes MOVX-SRAM, RS232-Bootloader im ROM, 2048 Bytes, internes Daten-EEPROM, drei 16-Bit-Timer, ein UART, ein Komparator, zwei Stackpointer, Flash-Speicher

Tabelle 1.3: 8051-kompatible Mikrocontroller in CMOS-Technologie