

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

Teil 01 - Einführung

1 Eine Einführung in C

1.1 Warum C ?

1.2 Wie entstand C ?

1.3 Der AVR-Mikrocontroller in einem eingebetteten System

Teil 02 - Erste Schritte

2 Grundausstattung und Installation der Werkzeuge

2.1 Testboard, Mikrocontroller, Programmer

2.1.1 Testboard und Mikrocontroller

2.2.2 ISP-Programmer

2.2 AVR Studio installieren

2.3 Versteht mich der Mikrocontroller ?

2.4 CodeVisionAVR C-Compiler installieren

2.5 Editieren von Quell-Dateien

Teil 03 - Aufbau eines C-Projektes

3 Was ist ein C-Projekt ?

3.1 Erzeugen eines C-Projektes

3.1.1 Ein neues Projekt beginnen

3.1.2 Ein C-Projekt generieren

3.2 Dateistruktur eines C-Projektes

3.3 Einbindung von AVR Studio in den CVAVR

3.4 AVR Studio Debugger

3.5 C-Compiler-Optionen

Teil 04 - Der Preprozessor

4 Preprozessor-Anweisungen

4.1 Struktur der C-Quell-Programme

4.2 #include-Anweisung

4.3 #define-Anweisung (Makro)

4.3.1 Makros ohne Parameter

4.3.2 Makros mit Parametern

4.4 #undef-Anweisung

4.5 #if-, #ifdef-, #ifndef-, #else- und #endif-Anweisungen

4.6 Andere Preprozessor-Anweisungen

Teil 05 - Syntax der C-Programme

5 Die Syntax der C-Programme

5.1 C-Quell-Programme

5.1.1 Kommentare

5.1.2 Deklarationen (Vereinbarungen)

5.1.3 Die Funktion main

5.1.4 Schlüsselwörter (Keywords) des CodeVisionAVR C-Compilers

5.2 Konstanten und Variablen

5.2.1 Zahlensysteme

5.2.2 Datentypen

5.2.3 Konstanten

5.2.4 Variablen

5.3 Operatoren

5.3.1 Arithmetische Operatoren

5.3.2 Relationale Operatoren

5.3.3 Logische und bitweise wirkende Operatoren

5.3.4 Andere Operatoren und Shortcuts

5.4 Komplexe Objekte in C

5.4.1 Funktionen

5.4.2 Funktions-Prototypen

5.4.3 Pointers und Arrays

5.4.3.1 Pointers

5.4.3.1.1 Pointers in Verbindung mit flash und eeprom

5.4.3.1.2 Pointers in Verbindung mit typedef

5.4.3.2 Arrays

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

- 5.4.3.2.1 Ein-dimensionale Arrays
- 5.4.3.2.2 Zwei-dimensionale Arrays
- 5.4.3.2.3 Drei-dimensionale Arrays
- 5.4.3.3 Benutzen der Array-Namen als Pointers
- 5.4.3.4 Arrays von Pointers
- 5.4.3.5 Pointers auf Funktionen (Funktionszeiger)
- 5.4.3.6 Funktionen in Verbindung mit `typedef`
- 5.4.4 Strukturen und Unionen
 - 5.4.4.1 Strukturen
 - 5.4.4.2 Unionen
- 5.4.5 Komplexe Typen (eine Zusammenfassung)
- 5.5 Steuerung des Programmablaufs
 - 5.5.1 Anweisungsblöcke { . . . }
 - 5.5.2 Die Anweisung `if`
 - 5.5.3 Die Anweisung `if` in Verbindung mit `else`
 - 5.5.4 Die Fallunterscheidung `switch`
 - 5.5.5 Die Schleife `for`
 - 5.5.6 Die Schleife `while`
 - 5.5.7 Die Schleife `do` in Verbindung mit `while`
- 5.6 Arbeiten mit den Ein-/Ausgabe-Ports

Teil 06 - Modularer Aufbau der AVR-Projekte

6 Modularer Aufbau

- 6.1 Das Konzept
- 6.2 Nomenklatur
- 6.3 Die globalen Header-Dateien
 - 6.3.1 Die globale Header-Datei `typedefs.h` (Typ- und Bit-Definitionen)
 - 6.3.2 Die globale Header-Datei `iomx.h` (Definitionen aller Register-Bits)
 - 6.3.3 Die globale Header-Datei `macros.h` (Definitionen von Makros)
 - 6.3.4 Die globale Header-Datei `switches.h` (Definitionen von Schalter-Makros)
- 6.4 Die Module
- 6.5 Anwendung der Module
 - 6.5.1 Das Modul `application` (Anwendungs-Steuerung)
 - 6.5.2 Das Modul `lcd_2wire` (Ausgabe auf LCD-20x4)
 - 6.5.3 Das Modul `num_conversion` (Typ-Konvertierung nach ASCII)
 - 6.5.4 Das Modul `adc_ref-1-1` (ADC mit interner Referenz 1,1 V)
 - 6.5.5 Das Modul `rc5_decoder` (RC5-IR-Fernsteuerung-Dekoder)
 - 6.5.6 Das Modul `rc5_encoder` (RC5-IR-Fernsteuerung-Enkoder)
 - 6.5.7 Das Modul `usart` (USART-Steuerung)
 - 6.5.8 Das Modul `twi_master` (I2C- bzw. TWI-Steuerung)
 - 6.5.9 Das Modul `timer0_pwm` (TIMER0-Steuerung)

Teil 07 - Anhang

7 Anhang

- 7.1 Begriffe und Definitionen
- 7.2 Eigene Bibliothek
- 7.3 AVR-Projekte (Programmbeschreibungen)
 - 7.3.1 AVR-Projekt PB_LED
 - 7.3.2 AVR-Projekt 2_Draht_LCD
 - 7.3.3 AVR-Projekt IRDMS

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

Vorbemerkung

Nichts ist vollkommen - und nichts ist endgültig! So auch nicht dieses Tutorial! Deshalb bitte immer erst nach dem neuesten Datum schauen. Vielleicht gibt es wieder etwas Neues oder eine Fehlerbereinigung oder eine etwas bessere Erklärung. Wer Fehler findet oder Verbesserungen vorzuschlagen hat, bitte melden (info@alenck.de).

Immer nach dem Motto: Das Bessere ist Feind des Guten und nichts ist so gut, dass es nicht noch verbessert werden könnte.

Gravierende Änderungen gegenüber der Vorversion

1. Neuer Abschnitt **5.6** in der Inhaltsangabe

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

1 Eine Einführung in C

Grundlage für dieses Tutorial waren folgende Quellen:

- "Programmieren in C" von Kerninghan/Ritchie (wobei Dennis Ritchie eher der Erfinder der Sprache C und Brian W. Kerninghan eher der Buchautor ist)
- "Embedded C Programming and the Atmel AVR" von Barnett/Cox/O'Cull
- "C-Booklet" von Reinhard Weber (eine Elektor-Publikation primär für den Mikrocontroller R8C/Tiny-Familie des Herstellers RENESAS)
- C-Programme für das ATM18-Projekt von Udo Jürß
- Beiträge im ATM18-Projekt-Forum des ComputerClub 2
- Elektor-Veröffentlichungen über einzelne ATM18-Projekte

Diesem Tutorial sind schon einige Beiträge vorausgegangen, die dazu führten, schon mal richtig loslegen zu können. Das ist auch ein Grund, weshalb das Tutorial zunächst eine andere Struktur hatte. Die höhere Priorität wurde als erstes auf **Teil 05 - Syntax der C-Programme** unter Einsatz des Compilers **CodeVisionAVR C-Compiler** gelegt. Mit diesen Beiträgen konnten Fortgeschrittene sich schon sehr gut behelfen. Nachdem einiges zusammengetragen wurde, hat sich der Autor entschlossen, das Tutorial mit einigen Ergänzungen in eine zusammengefasste allgemeine Form - sozusagen aus einem "Guss" - neu zu gestalten.

Es soll die Sprache C unter Berücksichtigung der Besonderheiten der **AVR-8-bit-Mikrocontroller** der Firma **Atmel** behandelt werden. Da die hier angesprochenen und bearbeiteten **AVR-Projekte** den **CodeVisionAVR C-Compiler**, der speziell für diese Mikrocontroller-Familie entwickelt wurde, einsetzen, beziehen sich die Beispiele in diesem Tutorial ohne weitere Hinweise auf gewisse Controller-Besonderheiten auf diesen Compiler (nachfolgend wird für den CodeVisionAVR C-Compiler die Abkürzung **CVAVR** verwendet).

1.1 Warum C ?

Viele Elektronik-Fans haben sich erfolgreich mit Mikrocontrollern beschäftigt und auch schon tolle Programme in Assembler geschrieben. Die Behauptung: "Nur wenn man Assembler anwendet, lernt man den Aufbau eines Prozessors richtig kennen und kann ihn dadurch besser nutzen" entspringt der Erfahrung aus der Computer-Pionierzeit, in der man bei Hochsprachen (ALGOL, FORTRAN, COBOL usw.) immer dann Probleme hatte, wenn zeitkritische (Realtime-)Probleme zu lösen waren. Diese waren nur durch die maschinennahe, auf die individuelle Datenverarbeitungsanlage zugeschnittene Programmierung - der Anwendung eines Assemblers - zu lösen.

C nimmt nun eine Zwischenstellung zwischen Assembler und Hochsprache ein. Sie vereint zwei an sich widersprüchliche Eigenschaften:

- gute Anpassung an die **individuelle** Rechnerarchitektur (Hardware) und
- hohe Portabilität, d.h. Übertragbarkeit auf **verschiedene** Rechnerarchitekturen.

C-Compiler erzeugen sehr effizienten Code (sowohl bezüglich der Laufzeit als auch bezüglich der Programmgröße), daher kann für die meisten Anwendungsfälle auf den Einsatz eines Assemblers verzichtet werden. Die **einfachere Programmierung** bedingt kurze Entwicklungszeiten für die Software und die **hohe Portabilität** ergibt einfachere und vor allem schnellere Anpassungen an eine andere Hardware.

Auch wenn man sich in der Assembler-Programmierung gut auskennt, kommt doch hin und wieder der Wunsch nach einer effektiveren Programmierumgebung auf. Wer schon mal versucht hat, mathematische Funktionen wie z.B. `1/x` oder `sinus(x)` in einem Assembler-Programm zu implementieren, kennt die Probleme. Hier bietet die Hochsprache C, der Industriestandard im Bereich der Unix-Systeme, der Mikrocontroller und der Mikroprozessoren, entscheidende Vorteile. C-Programme sind portabel, d.h. eine einmal erstellte Programmstruktur lässt sich auf andere Controller-Typen übertragen. Lediglich die **Port-Zuordnungen** und die Einstellungen der so genannten **Special Function Register** müssen angepasst werden.

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

Gerade für die AVR-Mikrocontroller empfiehlt sich daher der Einsatz der Programmiersprache **C**. Der Befehlssatz und die Architektur dieser Mikrocontroller wurden speziell für die Programmiersprache **C** optimiert. **C** stellt Konstrukte für Datenstrukturen und Programmablaufsteuerung zur Verfügung und ermöglicht dennoch als Hochsprache durch die direkte Einbindung von Assemblerbefehlen eine hardwarenahe Programmierung in zeitkritischen Programmabschnitten. Mit dieser Methode können auch komplexe Anwendungen strukturiert entwickelt werden und der Rückgriff auf umfangreiche Bibliotheken macht die Einbindung beliebiger Funktionen möglich. Einige **C**-Compiler optimieren den Programmcode sehr effizient. Zu diesen gehört auch der **CodeVisionAVR C-Compiler**, auf dem die weitere Beschreibung zugeschnitten ist. Daneben bietet der Halbleiterhersteller **Atmel** mit seinem **AVR Studio** eine sehr effektive und kostenlose Entwicklungsumgebung an: Ein zusätzlicher Grund, um auf **C** um- bzw. einzusteigen! Die gerade bei der Fehlersuche nötige Transparenz unterstützt das Tool sehr wirkungsvoll.

Aber! Vor den Erfolg haben die Götter bekanntlich den Schweiß gesetzt. Man wird schon einige Zeit benötigen, um eigene Programme in **C** zu schreiben. Daneben muss man sich - genau wie bei anderen Programmiersprachen auch - noch ein paar Fachausdrücke der englischen Sprache aneignen. Aus Beiträgen und Anfragen in Mikrocontroller-Foren kann man entnehmen, dass dies für viele Elektroniker ein nicht zu unterschätzendes Problem darstellt - besonders wenn es darum geht, dicke Handbücher der Mikroprozessoren und Beschreibungen von Anwendungen (Applikationen) in englischer Sprache zu lesen.

Der etwas höhere Aufwand der Einarbeitung im Vergleich zu **Basic**-Dialekten lohnt sich aber schnell, da man die AVR-Mikrocontroller mit **C** gut strukturiert programmieren kann und dabei trotzdem die volle Kontrolle über die Hardware behält. Durch den Einsatz von Bibliotheken (Libraries) muss nicht jede Grundfunktion selbst entwickelt werden. Fertige und getestete Bibliotheken stehen zur Verfügung. Wenn komplexere Anwendungen entwickelt werden, zahlt sich der höhere Einarbeitungsaufwand aus, da **C** als hardwarenahe Hochsprache einen guten Kompromiss darstellt.

1.2 Wie entstand **C** ?

Die Antwort auf die Frage nach dem "Warum **C**?" beantwortet zum Teil auch schon die Frage nach dem "Wie entstand **C**?"

Motivation für die Entwicklung der Sprache **C** ging einher mit der Entwicklung des Betriebssystems UNIX auf der DEC PDP-11 von **Dennis Ritchie**, welches in der Syntax dieser Sprache erzeugt wurde. Es sollte erreicht werden, dass UNIX auch auf andere Prozessoren portiert werden kann, ohne jedes Mal das Betriebssystem vollständig in Maschinensprache bzw. dem individuellen Assembler des Prozessors neu schreiben zu müssen. Lediglich ca. 5 % des Übersetzers, der sich auf die spezifische Hardware bezog, sollte der individuellen Anpassung vorbehalten bleiben. Die Befehle des neuen Betriebssystems sollten ohne Neuprogrammierung erzeugt werden können - lediglich durch Kompilierung mittels eines **C**-Compilers.

So kommt es, dass UNIX-Systeme nicht ohne **C** denkbar sind, aber die Programmiersprache **C** sehr wohl angewendet wird ohne UNIX.

1.3 Der AVR-Mikrocontroller in einem eingebetteten System

Der Ausdruck **eingebettetes System (Embedded System)** bezeichnet einen elektronischen Prozessor oder Controller, der in einem technischen Kontext eingebunden (eingebettet) ist. Dabei hat er die Aufgabe, das System, in das er eingebettet ist, zu steuern, zu regeln und/oder zu überwachen. Ein solches Bauelement stellt der **AVR-Mikrocontroller** dar. Die **C**-Programmiersprache ist, um im "Bild" zu bleiben, die wärmende Bettdecke.

Eingebettete Systeme verrichten - weitestgehend unsichtbar für den Benutzer - ihre Aufgaben in einer Vielzahl von Anwendungsbereichen und Geräten sowohl im inner- und äußer-häuslichen Bereich (z.B. in Waschmaschinen, Kühlschränken, Fernsehern, DVD-Playern und Flugzeugen, Kraftfahrzeugen, Mobiltelefonen) wie auch im industriellen Bereich (insbesondere in der Robotertechnik). Im Fall von komplexen Gesamtsystemen handelt es sich dabei meist um eine Vernetzung einer Vielzahl von ansonsten autonomen, eingebetteten Systemen.

AVR-8-bit-Mikrocontroller

Arbeiten mit CodeVisionAVR C-Compiler

Teil 01 - Einführung

Oft werden eingebettete Systeme speziell an eine bestehende Aufgabe angepasst. Das bedeutet, dass eine optimierte, gemischte Hardware/Software-Implementierung gewählt wird. Dabei vereint ein solches Vorgehen die große Flexibilität von Software mit der Leistungsfähigkeit der Elektronik. Die Software dient dabei sowohl zur Steuerung des Systems selbst als auch ggf. zur Interaktion des Systems mit der Außenwelt über Sensoren, Stellgliedern und definierten Schnittstellen.

Heutige eingebettete Systeme basieren häufig auf einer Controller-Plattform, die mit der PC-Welt wenig mehr gemeinsam haben. Im Bezug auf die Peripherie haben die AVR-Mikrocontroller einen sehr hohen Integrationsstand erreicht, der viele externe Bauelemente einsparen hilft und damit Kosten und Energieverbrauch sehr minimiert. Die Wiederverwendung von fertigen Programm-Modulen sowie die fehlende Notwendigkeit des vollständigen Re-Designs sind ebenfalls nicht zu unterschätzende Einsparungen bei den Programmierkosten neben den eingesparten Energie- und Materialkosten.

In einem eingebetteten System muss die Software vielen Anforderungen genügen und dabei oft Echtzeit(**realtime**)-Aufgaben erfüllen. Hardwareseitig existieren um den Controller herum nur stark reduzierte Ressourcen. Ein Flash-Speicher und ein EEPROM ersetzen mechanische Speicherkomponenten wie eine Festplatte und der stromsparende Controller kommt ohne Lüfter aus (bewegliche Teile bedeuten Verschleiß und Fehleranfälligkeit). Die Eingabe besteht meistens nur aus einem Tastenfeld und die Ausgabe wird - soweit vorgesehen - durch LED's oder ein LCD realisiert.

Die fertige Software auf dem Controller wird Firmware genannt. Sie wird in den Flash-Speicher des Controllers "programmiert" (geflasht). Dadurch besteht die Möglichkeit eines Firmware-Updates, ohne dass der Chip ausgewechselt werden muss (**ISP - In-System-Programmer**).

Die geschilderte Flexibilität bei gleichzeitig sehr hohem Integrationsgrad eines eingebetteten Systems setzt bei seiner System-Entwicklung/Programmierung die Einbeziehung der gesamten Umgebung (des Kontextes) voraus:

- Kenntnisse über den eingesetzten Chip mit seinen Registern, Ports, Timern usw.
- Kenntnisse über die einzusetzende Peripherie (Sensoren, Steuerungsmechanismen usw.)
- Kenntnisse der einzusetzenden Tools (AVD Studio, **C**-Compiler, Assembler und/oder Interpreter und Programmer) und letztlich
- Kenntnisse der eingesetzten Programmiersprache.

Das alles macht die Beschäftigung mit eingebetteten Systemen so interessant und die Einsatzmöglichkeiten scheinen fast grenzenlos zu sein.

Es ist sicherlich sehr interessant, auch die Architektur der Mikrocontroller zu ergründen: Welche Arten Speicher gibt es, wie sind sie aufgebaut und wie werden sie angesprochen? Welche Register und Ports gibt es und wie und was kann man über sie steuern? Zu vielen technischen Fragen wird der geneigte Leser von Mal zu Mal innerhalb der AVR-Projekte mehr und mehr erfahren, setzt aber - wenn man tiefer einsteigen will - die Kenntnis der Programmierung voraus.

Noch ein wichtiger Hinweis:

Fast alle Lehrbücher, Tutorials usw. über die **C**-Programmiersprache beginnen mit der "berühmten" Ausgabe **Hello, World!** auf einem Medium wie dem Bildschirm oder dem Drucker, das bei den AVR-Mikrocontrollern natürlich nicht zur Verfügung steht! Die zu Beginn der Einführung in eine Programmiersprache natürlich stark motivierende Wirkung einer sofort nachvollziehbaren Ein- und Ausgabe ("Da passiert schon etwas!") muss hier leider weit nach hinten geschoben werden. Die klassischen Funktionen `scanf()` für die Eingabe und `printf()` für die Ausgabe lassen sich bei den Mikrocontrollern ohne die entsprechenden Medien nicht anwenden. Lehrbücher, die diese Funktionen sogar - ohne ihre Syntax genauer zu erklären - bereits ab dem Anfang bei fast allen Beispielen anwenden, können leider nicht 1zu1 für die hier beschriebene Plattform verwendet werden. Es ist auch leider nicht möglich, die Beispiele ohne entsprechende Funktionen für andere Ein-/Ausgabe-Medien gleich zu Beginn der Lernphase auf andere Art und Weise zum Laufen zu bringen.

Mit anderen Worten: Zu Beginn der Einführung kommen - sofern nicht mit vollendeten Programmen gearbeitet wird - mehr "Gedanken"-Beispiele zur Anwendung!