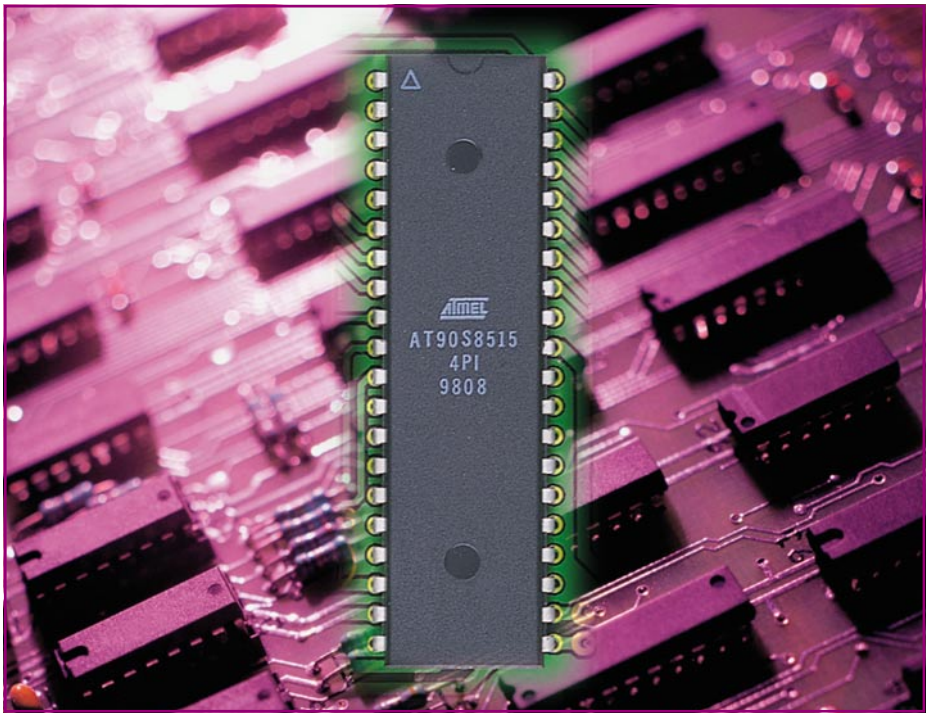

AVR-Grundlagen

Teil 3

Im dritten Teil der Serie stellen wir die Entwicklungsumgebung der AVR-Mikrocontroller vor. Von der Installation auf dem Rechner bis zur Erstellung eines kleinen Projektes anhand eines einfachen, implementierten Beispielprogramms werden die Grundfunktionen des „AVR - Studios" aufgezeigt.

AVR-Studio

Das AVR-Studio stellt die komplette Entwicklungsumgebung für die Mikrocontroller der AVR-Familie dar. Es beinhaltet neben dem Quelltexteditor und dem Compiler auch einen Simulator, mit dem man kleinere Programme testen kann.

Des Weiteren verfügt das AVR-Studio über die Möglichkeit der Steuerung der AVR-Emulatoren und AVR-Starter-Kits. Mit der entsprechenden Hardware (z. B. dem Starter-Kit Atmel STK 500) lassen sich die mit dem AVR-Studio programmierten und getesteten Programme auch in den Mikrocontroller übertragen.

In den folgenden Beschreibungen der Entwicklungsumgebung werden wir noch zahlreiche weitere Funktionen des AVR-Studios darstellen. Die Software findet sich sowohl auf der mit dem Starter-Kit mitgelieferten CD-ROM als auch im Internet bei Atmel.

Installation

Zur Installation auf dem Rechner wird nach dem Kopieren des Programmverzeichnisses die Datei „setup.exe" im Quellverzeichnis gestartet. Daraufhin öffnen sich nacheinander mehrere Dialogfenster, welche jeweils mit dem Button „Next" zu bestätigen sind. Nachdem man das letzte Fenster mit „Finish" geschlossen hat, ist das AVR-Studio für ein erstes Testprojekt einsatzbereit.

Das erste Projekt

Unser erstes Projekt soll eine an Port A.0 eines AT90S8515 angeschlossene LED zum Blinken bringen - die Grundübung zum Herantasten an einen neuen Controller.

Um mit der Programmierung beginnen zu können, sind zunächst einige Einstellungen vorzunehmen. Über das Menü „Project → New" wird ein neues Projekt erzeugt, worauf sich ein Dialogfenster (Abbildung 1) öffnet. Hier gibt man den Projektnamen (Project name), das Verzeichnis, in dem das Projekt angelegt werden soll (Location) und den Typ des Projektes (Project type) an.

Als Testprojekt legen wir unser Projekt mit dem Namen „blinken" im Verzeichnis „C:\BLINKEN" an. Als Typ wird „AVR Assembler" ausgewählt, da wir die Software des AVR-Studios benutzen wollen. Nachdem man die Einstellungen über den OK-Button bestätigt hat, öffnet sich die im Moment noch leere Projektübersicht, die zwei Untergruppen enthält. Die Gruppe „Assembler Files" ist zur Aufnahme aller Quelltextdateien reserviert, die Gruppe „Other Files" enthält zusätzliche Dateien, z. B. Beschreibungen und Hintergrundinformationen.

Um das noch leere Projekt zu füllen, erstellen wir über das Menü „File → New text file" eine neue Textdatei, deren Name im darauf folgenden Dialog anzugeben ist. Dateien, die später einen Assembler-Quelltext enthalten sollen, müssen die Endung „.asm" tragen.

Die im Beispielprojekt erstellte Datei nennen wir „main.asm", da es sich um das Hauptmodul des Programmes handeln soll. Nachdem man den Namen über den OK-Button bestätigt hat, wird die Datei in die

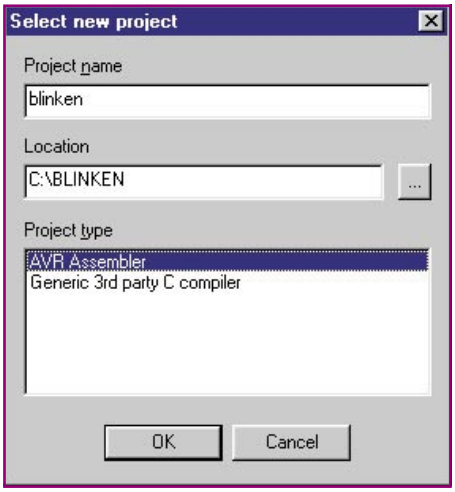

Bild 1: Die Erstellung eines neuen Projektes

Bild 2: Der Quellcode des Beispielprogramms (Fortsetzung s. nächste Seite)

```
;*****************************************************************************
;* ELV Elektronik AG                                                         *
;*                                                                           *
;* Titel       : Einfaches Testprogramm für AVR-Mikrocontroller              *
;* Version     : 1.0                                                         *
;* Datum       : 2001-08-21                                                  *
;*****************************************************************************
;* Mikrocontroller   : AT90S8515                                             *
;* Taktfrequenz      : 1 MHz                                                 *
;*****************************************************************************
;* Beschreibung:                                                             *
;* Das Programm wechselt in einem festen Intervall den logischen Zustand     *
;* von Port A.0 in den entgegengesetzen logischen Zustand.                   *
;* High -> Low  bzw. Low -> High                                             *
;*****************************************************************************


; Register- und Bit-Definitionen für den AT90S8515
.include "C:\Programme\Atmel\AVR Studio\Appnotes\8515def.inc"



.CSEG                       ; Beginn des Codesegments
.ORG  0x0000                ; Adresse 0x0000 im Programmspeicher

   rjmp  main               ; RESET: Sprung an den Anfang des
                            ; Hauptprogramms

.ORG  0x0007                ; Adresse 0x0007: Einsprungadresse im
                            ; Programmspeicher für Timer-Interrupt
   rjmp  timer_0            ; Aufrufen der Interrupt-Service-Routine


;*****************************************************************
;* Unterfunktionen                                               *
;*****************************************************************

;***********************************************************
;* Interrupt-Service-Routine (ISR) für Timer 0             *
;* Diese Funktion wird bei jedem Ansprechen von Timer 0 ge- *
;* startet.                                                *
;***********************************************************
timer_0:
   sbic  PORTA,0
   rjmp  T0_CLEAR
   sbi      PORTA,0
   rjmp  T0_END
T0_CLEAR:
   cbi      PORTA,0
T0_END:
   reti                     ; Rücksprung aus ISR muss mit
                            ; reti erfolgen
```

Projektübersicht eingebunden und geöffnet. Jetzt ist sie mit der Maus in die Gruppe „Assembler Files" zu schieben.

Nachdem alle bis jetzt beschriebenen Schritte beendet sind, kann man mit der Implementierung des Testprogramms (Abbildung 2) beginnen. Dazu erstellt man zunächst einen passenden Programmkopf, der alle wichtigen Informationen sowie eine Kurzbeschreibung der Programmfunktion enthält.

Das eigentliche Programm beginnt mit dem Einbinden der Definitionsdatei für den entsprechenden Mikrocontroller, die sich im Applikationsverzeichnis des AVR-Studios befindet. Deren Daten werden beim späteren Assemblieren automatisch mit in den Programmcode eingebunden.

Dies erfolgt mit der „.include <*Pfad*>"-Anweisung, wobei als Parameter der Pfad zur Datei anzugeben ist. Die eingebundene Datei enthält die mit der Adresse verknüpften Bezeichner für die Register des Mikrocontrollers. Mittels dieser Bezeichner lässt sich eine größere Übersichtlichkeit des Quellcodes erreichen, man muss also nicht alle Beziehungen einzeln im Quellcode auflisten.

Im Anschluss daran werden die Einsprungadressen des Programms festgelegt. Der Programmablauf beginnt nach einem Reset immer ganz oben im Programmspeicher bei Adresse 0, sodass an dieser Stelle

```
;************************************************************
;* Initialisierung der Prozessorregister                    *
;************************************************************
init:
   cli                          ; alle Interrupts abschalten

   ldi   R24,  0b11111111
   out   DDRA, R24              ; PORT A als Ausgang
   out DDRB,   R24              ; PORT B als Ausgang
   out   DDRC, R24              ; PORT C als Ausgang

   ldi   R24,  0b11111111       ; PullUp-Widerstände für alle Ports
                                ; aktivieren
   out   PORTA,   R24
   out   PORTB,   R24
   out   PORTC,   R24
   out   PORTD,   R24

   ldi   R24,  0b00000101       ; Prescaler von Timer 0: CK/1024
   out   TCCR0,   R24

   ldi   R24,0b00000010         ; Timer 0 Interrupt freischalten
   out   TIMSK,R24
   sei                          ; Interrupts wieder einschalten
   ret

;****************************************************************
;* Hauptprogramm                                                *
;* Die Hauptprogrammschleife arbeitet als eine Endlosschleife   *
;* in der alle relevanten Befehle und Verzweigungen ausgeführt  *
;* werden.                                                      *
;****************************************************************
main:                           ; Beginn des Hauptprogramms:
                                ; Hier werden alle notwendigen
                                ; Initialisierungen vorgenommen.

   ldi   R24,LOW(RAMEND)        ; Stackpointer (Zeiger auf den Programm-
   out   SPL,R24                ; stack) auf die oberste Adresse im
   ldi   R24,HIGH(RAMEND)       ; Datenspeicher (RAM) setzen
   out   SPH,R24

   rcall init                   ; Aufruf der Unterfunktion zur
                                ; Initialsierung der SFR des
                                ; Mikrocontrollers

loop:                           ; Hauptprogrammschleife
   rjmp  loop
```

Bild 2: Fortsetzung des Quellcodes

eine Sprunganweisung zum eigentlichen Programmbeginn erfolgen muss.

Die nächsten Speicherstellen repräsentieren die Einsprungadressen für die Interrupt-Service-Routinen, d. h., bei einer gültigen Unterbrechungsanforderung verzweigt das Programm automatisch an dieser Stelle. Es muss also an diesem Punkt wieder eine Sprunganweisung zur entsprechenden Interrupt-Routine zu finden sein.

Diese Schritte wollen wir anhand unseres Beispielprogramms nachvollziehen. Hier wird zuerst über die „.CSEG"-Anweisung der Beginn des Codesegments des Programms definiert. Mittels „.ORG" legt man den Platz des Programms im ROM fest, sodass der Sprungbefehl zum Hauptprogramm „main" definiert an Adresse 0 liegt. Nachfolgend finden sich die Einsprungadressen der Interrupts, von denen wir hier aber nur die des Timers 0 nutzen (Adresse 7).

Nach dem Reset wird der Programmablauf also direkt zum Hauptprogramm (main, siehe Ende des Listings) verzweigt, an dessen Anfang als Erstes die Initialisierungen erfolgen müssen. Hier ist der erste Schritt das Setzen des Stackpointers auf die oberste Adresse im RAM, die durch die Konstante „RAMEND" festgelegt ist. Der Stack ist ein Stapelspeicher, in dem wichtige Daten beim Aufruf einer Unterfunktion oder Interrupt-Routine abgelegt werden. Man hat immer nur Zugriff auf das zuletzt eingespeicherte Element. Diese Daten beinhalten z. B. die Rücksprungadresse zur aufrufenden Funktion, die beim Beenden der Unterroutine durch „ret" bzw. „iret" genutzt wird. Der Stackpointer zeigt dabei immer auf das letzte Element, das im Stack gespeichert ist. Er wird bei einem Funktionsaufruf entsprechend erhöht. Aus diesem Grund darf man den Stackpointer nicht in einer Unterfunktion initialisieren, da schon beim Aufruf dieser Funktion Daten in den Stack gelegt und durch eine Veränderung des Stackpointers verloren gehen würden.

Im nächsten Schritt sind die entsprechen-

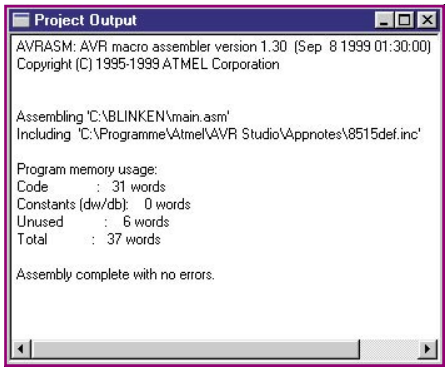

Bild 3: Das Ergebnisfenster

den Special-Function-Register des Mikrocontrollers auf die passenden Startwerte zu setzen. Da nach einem Reset alle Ports als Eingänge geschaltet sind und wir für die angestrebte Blinkfunktion nur einen Ausgang benötigen, konfiguriert man diesen Port (A) als Ausgang. Alle anderen werden auf den gleichen Zustand gesetzt, da unbenutzte und unbeschaltete Eingänge immer ein Störfaktor sind, weil es hier keinen definierten Zustand gibt. Im nächsten Schritt werden auch deshalb alle PullUp-Widerstände der Ports aktiviert, so liegen alle Ports auf einem definierten Pegel.

Anschließend erfolgt die Initialisierung der Register für Timer 0, sodass die zugehörige Interrupt-Service-Routine in einem festen Intervall aufgerufen wird. Dazu teilt der Prescaler (Vorteiler) den Prozessortakt durch 1024, damit man eine entsprechend der Aufgabe des Programms langsame Blinkzeit erreicht.

Jetzt ist noch der Interrupt freizuschalten, indem das passende Bit im „TIMSK"-Register gesetzt wird, und in der Interrupt-Routine noch der Quelltext zum Invertieren des Pins (die vier Zeilen nach „timer_0:") einzufügen - und das Programm ist fertig.

Übersetzen des Programms

Damit der Mikrocontroller das Programm versteht, ist es notwendig, die Befehle in ein entsprechendes Format zu übersetzen. Diese Aufgabe übernimmt der „Assembler", der die vorhandenen Befehle in die zugehörigen Codes im hexadezimalen Format übersetzt. Der Assembler wird über das Menü „Projekt → Assemble" oder durch das Drücken der F7-Taste gestartet. Sobald der Vorgang beendet ist, öffnet sich ein Fenster, in dem das Ergebnis dargestellt ist. Eventuelle Fehler werden übersichtlich dargestellt, sodass man sie einfach finden kann. Sind alle Fehler beseitigt, sieht das Ergebnis wie unter Abbildung 3 gezeigt aus. Hier sieht man die Ausnutzung des Programmspeichers, aufgeschlüsselt in vier Teile. Unter der Kennung „Code" ist die Anzahl der verwendeten Programmcodewörter aufgezeigt. Konstanten (Constants), die im Programmspeicher abgelegt werden, haben wir in diesem Beispiel nicht verwendet, sodass hier die Angabe „0 words" erscheint. Weiterhin sind ungenutzte Bereiche angegeben, die z. B. durch „.ORG"-Anweisungen auftreten können. Hier werden 6 ungenutzte Wörter angegeben, die im Bereich der Einsprungadressen zustande kommen, da sich zwischen der Sprunganweisung nach einem Reset und dem Sprungbefehl für die Interrupt-Routine noch weitere Einsprungstellen für Interrupts befinden. Diese Interrupts benötigen wir in diesem Programm aber nicht. Die vierte Angabe (Total) bezieht sich auf den insgesamt belegten Programmspeicherplatz, der hier mit 37 Worten angegeben ist.

Oben in der Auflistung findet man übrigens auch die Einbindung der Definitionsdatei des verwendeten Controllers.

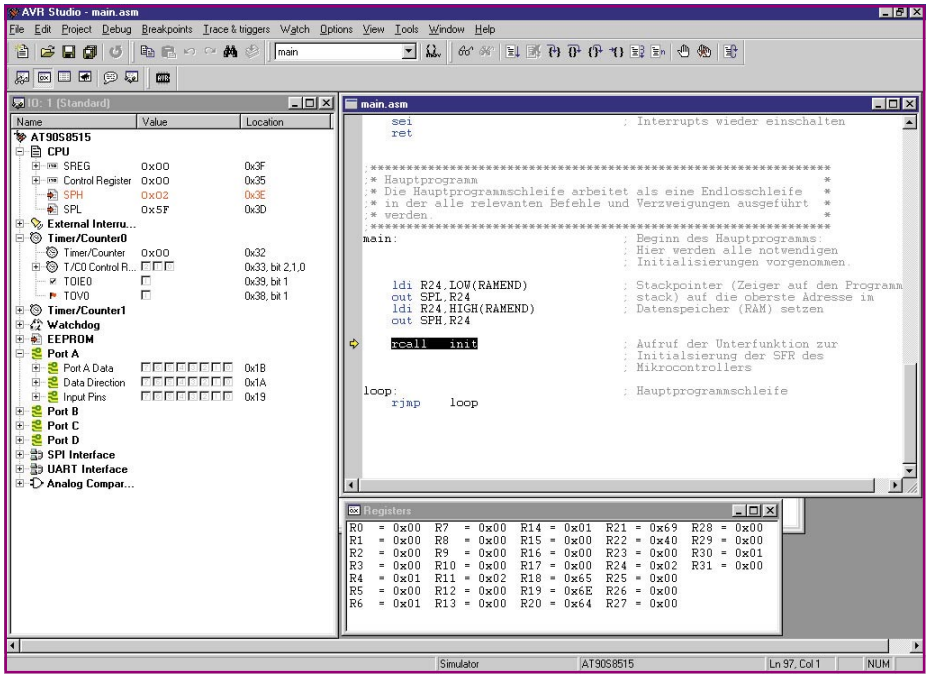

Bild 4: Der Software-Simulator des AVR-Studios

Testen des Programms

Der Test dieses Programms gestaltet sich, da die Hauptfunktion in einer Interrupt-Routine läuft, mit dem Software-Simulator nicht ganz einfach, so kann man nur die einzelnen Programmteile, nicht aber den gesamten zeitlichen Ablauf testen.

Der Start des Simulators erfolgt über das Menü „Debug → Go" oder mit der F5-Taste. Jetzt lassen sich über den Menüpunkt „View" verschiedene Ansichten darstellen, sodass man während der Simulation über jedes Prozessorregister, den gesamten Datenspeicher usw. den Überblick behält. In Abbildung 4 befindet sich auf der linken Seite eine Übersicht über die SFR (Special-Function-Register), in der jedes relevante Bit der integrierten Peripherie dargestellt ist. Unten rechts ist der Inhalt der Register zu sehen. Diese Anzeigen werden nach jedem ausgeführten Schritt aufgefrischt, nicht aber online im laufenden Betrieb. Im Menü „Debug" befinden sich alle Befehle zur Ablaufsteuerung des Simulators, wobei hier nur die beiden wichtigsten genannt werden sollen.

Der Befehl „Trace into" führt immer nur einen Befehl aus und stoppt den Simulator danach wieder automatisch, wobei auch ein Auffrischen der Anzeigen in den ausgewählten Übersichten erfolgt. Der richtige Aufruf von Interrupt-Service-Routinen lässt sich mit dieser Funktion nicht testen.

Der zweite wichtige Befehl lautet „Go". Er veranlasst, dass der Programmablauf so lange fortgeführt wird, bis er über den Befehl „Break" oder durch das Erreichen eines Haltepunktes (Breakpoint) beendet wird. Somit sind hier auch die Interrupt-Routinen erreichbar, indem man an ihren Anfang ein Breakpoint setzt. Ein solcher Haltepunkt wird mit dem Befehl „Toggle Breakpoints" im Menü „Breakpoints" gesetzt bzw. gelöscht, und zwar jeweils für die Zeile im Quelltext, in der sich der Cursor befindet. Erreicht das Programm bei der Ausführung einen Breakpoint, so wird der Programmlauf sofort beendet und in den Übersichten befinden sich die aktuellsten Daten. Hier sei jedoch vollständigkeitshalber noch einmal darauf hingewiesen, dass das Zeitverhalten bei der Simulation nicht exakt mit dem der späteren Schaltung übereinstimmt.

Im nächsten Teil der Serie testen wir unter Mithilfe des Starter-Kits STK 500 unser fertig erstelltes Programm in einem entsprechend über das STK 500 programmierten Controller.