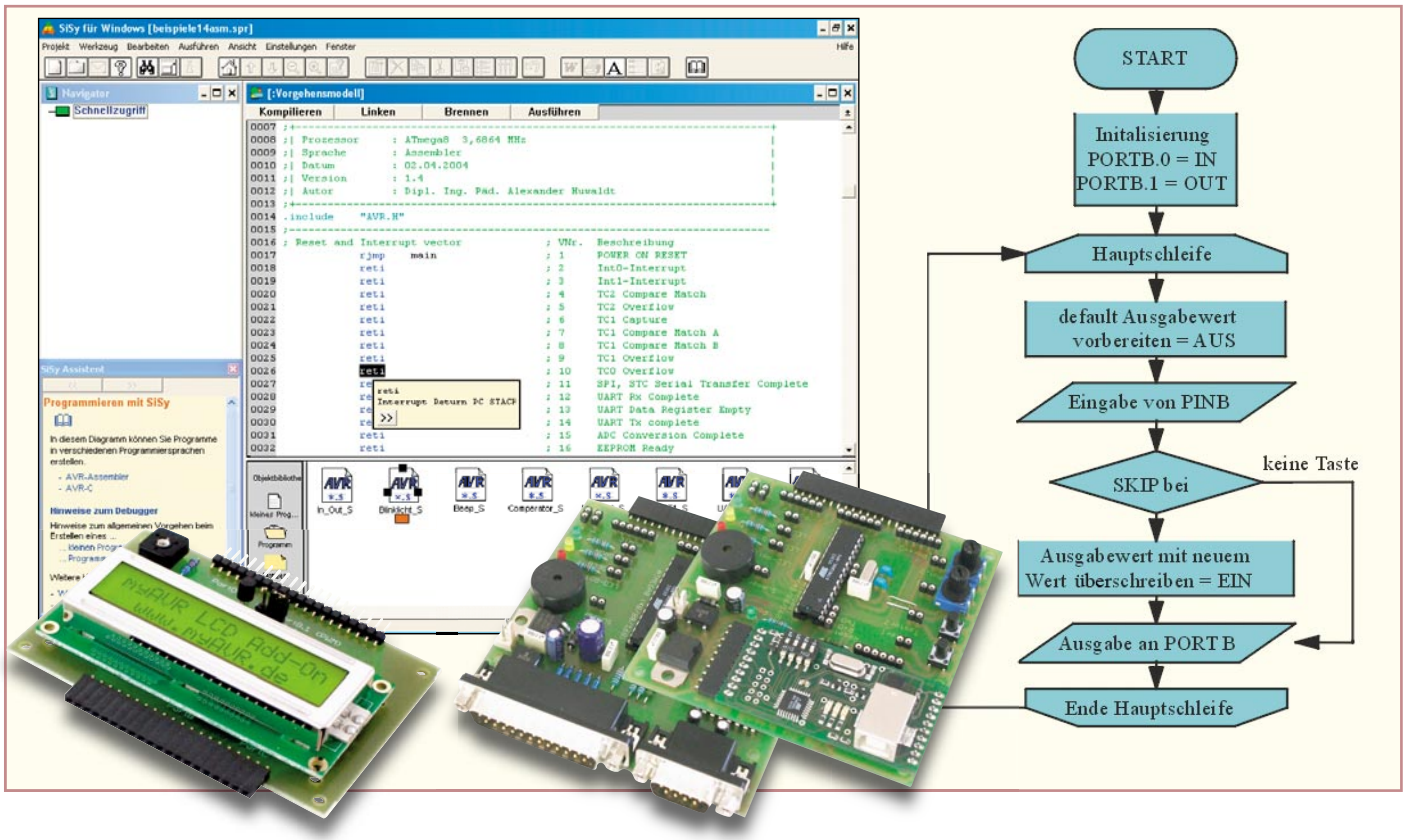

Mikrocontroller-Einstieg mit myAVR Teil 6

Keine Angst vor dem Einstieg in die Welt der Mikrocontroller-Programmierung! Die myAVR-Sets enthalten alles Nötige für den schnellen und fundierten Beginn der Programmierer-Karriere – Experimentier-Board mit ATMEL-Controller, Lehrbuch, Softwarepaket, Kabel, sämtliches Zubehör. Im sechsten Teil unserer Serie zur Programmierung des AVR geben wir eine kurze Einführung in die AVR-Programmierung über die Hochsprachen C und BASCOM.

Hochsprache oder nicht?

Bisher haben wir uns ganz der sehr Hardware-nahen Assembler-Programmierung gewidmet, die ob des erforderlichen Aufwandes – jeder Programmschritt wird ja quasi „zu Fuß“ eingegeben, obwohl uns das SiSy-Programm bereits viele Aufgaben, wie etwa das Schreiben der Vektortabelle, abnimmt – so manchen Programmierer abschreckt. Allerdings sind diese Hardware-Kenntnisse, z. B. der Register und der Arbeitsstruktur des Controllers, auch in anderen Programmiersprachen unbedingt erforderlich, weshalb Assembler-Kenntnisse eine nie verlorene Wissensbasis sind. Allerdings nehmen uns die komplexeren Programmierbefehle der Hochsprachen viel mühsame Detailarbeit ab, ganze Routinen werden dort durch einen einzigen Befehl ersetzt. Allerdings – auch das muss man wissen – sind manche Programmierdetails durch die Hochsprachen nicht zu realisieren, allenfalls durch dann doch wieder aufwändigere Umwege. Und man muss oft weit abstrakter und weitsichtiger denken als bei der maschinennahen Assembler-Programmierung.

Die für den ATMEL-AVR-Controller am meisten angewandten Hochsprachen sind C/C++ und BASCOM. Während Erstere quasi den Industriestandard darstellt, den unzählige Programmierer genauso beherrschen wie ein normaler Computernutzer seinen Internet-Browser, stellt BASCOM die Alternative für alle dar, die in der Vergangenheit einmal BASIC gelernt haben oder aber eine relativ einfache Programmiersprache erlernen wollen. Während C/C++ noch weitgehende Ähnlichkeiten zur Assembler-Programmierung erkennen lässt, ist BASCOM schon relativ fern der Hardware, glänzt aber eben durch schnelle Erlernbarkeit und einfache Strukturen. Auf beide Programmiersprachen wollen wir in der Folge kurz eingehen.

AVR-Programmierung in C

Wer hier einen C-Programmierkurs erwartet, muss allerdings leider enttäuscht werden, dies überlassen wir dann doch Hochschullehrern oder entsprechend dicken Lehrbüchern – alle, die C/C++ gelernt haben, werden hier zustimmen. Wir wollen an dieser Stelle lediglich den C-Kundigen einige Hinweise zur AVR-C-Programmierung und Einsteigern einen ganz kleinen Einblick in die Programmierung in C ge-

```
//---------------------------------------------------------------------
// Titel     : Beispiel IN/OUT für SiSy AVR-Board
//---------------------------------------------------------------------
// Funktion  : LED über Taster einschalten
// Schaltung : Taster an PORTD.2, LED an PORTB.0
//---------------------------------------------------------------------
// Prozessor : ATmega8  3.6864 MHz
// Sprache   : C
// Datum     : 17.03.2004
// Version   : 1.2
// Autor     : Dipl. Ing. Päd. Alexander Huwaldt
//---------------------------------------------------------------------
#include <io.h>
#include <avr/wdt.h>
//---------------------------------------------------------------------
main()
{
    unsigned char i;
    //hier Init-Code eingeben
    DDRB = 0xFF;                    // PORTB = Ausgang
    DDRD = 0x00;                    // PORTD = Eingang
    PORTD = 0xFF;                   // PORTD = PULL-UP

    while (true)                    // Mainloop
    {
        wdt_reset ();               // Wachhund zurückpfeifen
        // eine einfache Warteschleife
        for (int k=0; k<1000; k++) ();

        if (!(PIND&0x04))           //Bit 3 = 0 dann ist Taster gedrückt
        PORTB = i;
        i++;
    }
}
//---------------------------------------------------------------------
```

Definitionsdateien

Hauptprogramm

Bild 54: Ein Programmbeispiel in C – hier ist die Grundstruktur eines C-Programms deutlich zu erkennen.

ben, selbst ein einfacher Programmierkurs würde uns bequem ein bis zwei Jahre Folge für Folge beschäftigen.

Betrachtet man die Grundstruktur eines C-Programms (Abbildung 54) für den AVR-Controller, so findet man gewisse Ähnlichkeiten zum Assembler-Programm. Auch hier gibt es eine Definitionsdatei, die Datei „io.h“. Zusätzlich liefert C noch eine Reihe vorbereiteter Bibliotheksfunktionen, z. B. für den Watchdog-Timer, für die ebenfalls Definitionsdateien einzubinden sind. Wie bei Assembler gliedert sich ein C-Programm in die Initialisierungssequenz und eine Endlosschleife, in der die eigentliche Anwendung läuft.

SiSy-AVR ermöglicht es, unter der gleichen Programmierumgebung wie bei der Assembler-Programmierung C-Programme zu schreiben und diese in ausführbare Programme (.hex, .bin) zu kompilieren. Die in C üblichen Wertzuweisungen für die Realisierung von Ein- und Ausgaben werden vom Compiler automatisch in entsprechende Assembler-Anweisungen umgesetzt. Auch die Interruptvektoren werden vom Compiler automatisch erzeugt. Wenn man einen Interrupt programmieren möchte, benötigt man zusätzlich die Definitionsdateien „interrupt.h“ und „signal.h“. Dabei ist die Kennung des Interruptvektors als Parameter einzugeben. Das Makro erzeugt eine vollständige ISR und trägt den Interruptvektor entsprechend in die Tabelle ein. Ausführliche Erläuterungen und Programmierbeispiele zur AVR-Spezifik finden sich im AVR-Lehrbuch.

AVR-Programmierung in BASCOM

BASIC ist eine Programmiersprache, die sich durch ihre leistungsfähigen und leicht verständlichen Befehle schon immer großer Beliebtheit erfreute. Das ist heute nicht anders als in den 80er Jahren, als es darum ging, Z80- oder 68xxx-Systeme abseits der Maschinensprache zu programmieren. Darum ist BASIC auch heute noch vielfach vor allem für Einsteiger und Gelegenheitsprogrammierer das Mittel der Wahl, da man sich hier besonders schnell zurechtfindet und die Struktur der Programme (sofern man „sauber programmiert“) sehr übersichtlich ist. Zudem gibt es zahlreiche Publikationen, vor allem im Internet, mit vielen Anwendungsbeispielen, so dass hier ein Erfolgserlebnis nicht lange auf sich warten lässt.

Der derzeit wohl beliebteste BASIC-Compiler, der quasi Standard in der heutigen BASIC-Welt ist, ist BASCOM. Die Software ist von der niederländischen Firma MCS Electronics (www.mcselec.com) für knapp unter 80 Euro beziehbar und läuft auf allen PCs mit Windows-Betriebssystemen ab Version 95. Eine komplett lauffähige und nur in der Größe des Programmcodes begrenzte Demo-Version ist kostenlos beziehbar. Das ist sehr angenehm, man kann sich in Ruhe und ohne Zeitlimitierung einarbeiten und später entscheiden, ob man sich die Vollversion für größere Programme anschafft.

Das AVR-Lehrbuch widmet der Programmierung mit BASCOM ein umfangreiches Kapitel, das alle relevanten Besonderheiten der AVR-Programmierung systematisch abarbeitet. Allerdings sollte man auch hier schon einmal Berührung mit einer Programmier-Hochsprache gehabt haben, mindestens sollte dies BASIC sein, aber auch C-Kundige werden sich hier sofort zurechtfinden.

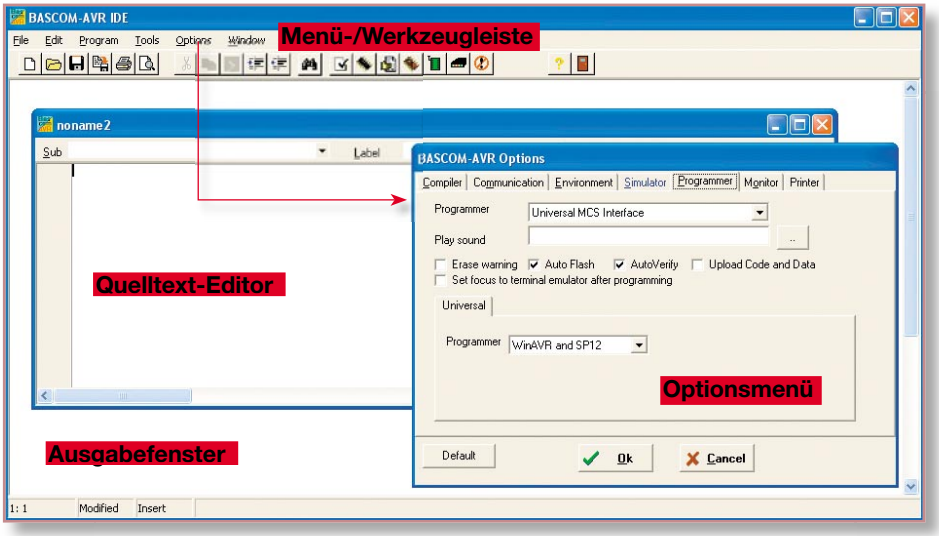

Bild 55: Die Oberfläche von BASCOM, hier mit geöffnetem „Options“-Fenster mit den myAVR-Einstellungen

Das Einrichten des Compilers

Um den Compiler, dessen Oberfläche in Abbildung 55 zu sehen ist, auf das myAVR-Board einzurichten, ist unter „Options“ → „Programmer“ im Fenster „Programmer“ als Programmer „Universal MCS Interface“ und „WINAVR and SP12“ anzuwählen, wie es in Abbildung 55 dargestellt ist.

Über „Communication“ kann die serielle Schnittstelle eingestellt werden. Hier ist der entsprechende Port, eine Baudrate von 9600, Parity „None“, Databits „8“, Stoppbits „1“, Handshake „None“ und Emulation „None“ einzustellen. Dann kann BASCOM über ein Nullmodemkabel via integriertem Terminal und Debugger/Simulator mit dem Controller auf dem myAVR-Board kommunizieren.

Vor Beginn der Programmierarbeit ist auch der jeweils verwendete Controller auszuwählen. Für unseren ATmega 8 ist unter „Compiler“ und „Chip“ der Chip „m8def.dat“ auszuwählen. Geht man die restlichen Optionen im „Options“-Menü durch, erschließen sich zahlreiche Einstell- und Beeinflussungsmöglichkeiten, die die spätere Programmierarbeit deutlich erleichtern, sind doch alle diese Einstellungen in der späteren Definitionsdatei „verpackt“.

```
' myAVR.bas Vorlage für myAVR Board
' Grundstruktur eines µC-Programms
'-----------------------------------------------------
$regfile = "m8def.dat"
$crystal = 3686400
' hier Initialisierung durchführen
Do
' hier Verarbeitungsaufgabe durchführen
Loop
End
```

Bild 56: Die Grundstruktur eines BASCOM-Programms

So muss man sich etwa, wenn man unter „LCD“ alle gewünschten Kriterien bis hin zur Port-Belegung vordefiniert hat, später im Programm nicht im Einzelnen um die Definition der LCD-Schnittstelle kümmern, sondern lediglich auf die Definitionsdatei verweisen.

Die Programmstruktur

Auch das BASCOM-Programm, das C- und auch Assembler-Kundigen in seiner Grundstruktur (Abbildung 56), bestehend aus Initialisierungssequenz, Hauptprogramm und Endlosschleife, gleich bekannt vorkommen dürfte, benötigt zunächst eine Definitionsdatei, eben jene, die wir beim Einrichten des Compilers ausgewählt haben: m8def.dat.

Der BASCOM-Befehlssatz ist sowohl über die Online-Hilfe als auch als äußerst umfangreiches und kaum eine Frage offen lassendes, in leicht verständlichem Englisch geschriebenes pdf-Handbuch (über Download bei MCS kostenlos beziehbar) beschrieben.

Kompilieren und Brennen

Das Kompilieren des fertigen Programms erfolgt ähnlich einfach, wie wir es im Assembler-Lehrgang bei SiSy-AVR kennengelernt haben.

Zuvor kann man ein paar sehr nützliche BASCOM-Werkzeuge einsetzen, um das entwickelte Programm zu testen: den Syntax-Check für den Quellcode und den AVR-Simulator (Abbildung 57). Besonders

Bild 58: Der fertige Quellcode wird kompiliert.

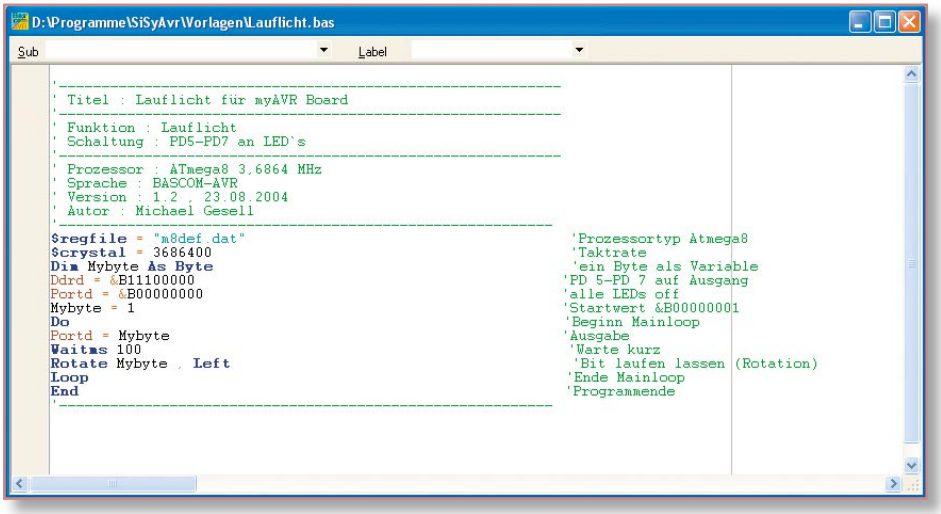

Bild 59: Unser kleines BASCOM-Beispielprogramm

Letzterer bietet bis hin zu visualisierten Port-Ausgaben und Hardware-Simulation, z. B. von LC-Displays, sowie der Übersicht über die Prozessor-Flags und -Stacks wirklich alle Möglichkeiten des Programmtests.

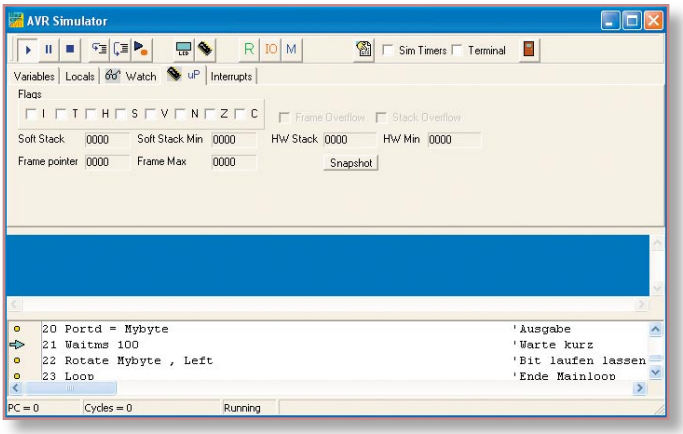

Bild 57: Mehrere Werkzeuge, hier der Simulator, ermöglichen den ausführlichen Programmtest.

Das Kompilieren selbst erfolgt unspektakulär, ein kleines Überwachungsfenster (Abbildung 58) zeigt den Fortschritt und eventuelle Fehler an.

In ganz ähnlicher Weise erfolgt das Brennen in den AVR über die Option „Send to Chip“ im Menü „Program“. Auch hier erfolgt die Überwachung durch kleine Kontrollfenster, die das Schreiben und Verifizieren dokumentieren und bei Fehlern entsprechend signalisieren. Zu beachten ist hier, dass das myAVR-Board unbedingt über ein separates Netzteil angeschlossen sein muss, da der BASCOM-Compiler nicht für die Betriebsspannung via paralleler Schnittstelle sorgt.

Wir wollen in unserer kurzen Einführung nicht detailliert auf die Programmierung eingehen, sowohl das AVR-Lehrbuch als auch das BASCOM-Handbuch tun dies erschöpfend. Allerdings wollen wir abschließend ein kleines Programm aufzeigen, das die drei LEDs auf dem myAVR-Board als Lauflicht einsetzt (Abbildung 59).

Nach der Eintragung der Definitionsdatei und der Taktrate des ATmega 8 wird zuerst die später im Hauptprogramm zu verarbeitende Variable über den Befehl DIM deklariert, dann, ähnlich wie beim Assembler, die Ports, der Ausgangszustand und ein Startwert für die Variable definiert.

Im Hauptprogramm sorgen nun die Befehle WAITMS (stoppt das Weiterlaufen in der Endlosschleife jeweils kurz für 100 ms, um ein sichtbares Neu-Durchlaufen des Lauflichts zu erzeugen) und ROTATE (schiebt in der Variablen Bit für Bit nach links durch) ganz einfach für die Erzeugung des kleinen Lauflichts. Vergleicht man dieses Programm mit seinem Assembler-Pendant, dann wird schnell klar, wie schnell man hier programmieren kann, zumal das Handbuch eine ausgezeichnete Hilfe mit jeweiligem Programmbeispiel bietet. In Abbildung 60 ist die für das Lauflicht-Programm vorzunehmende Verdrahtung auf dem myAVR-Board zu sehen.

Im nächsten und abschließenden Teil unserer Artikelserie krönen wir unseren Exkurs mit der Anbindung des optionalen LCD-Boards an das myAVR-Board und einer kompletten AVR-Anwendungslösung mit Hard- und Software.

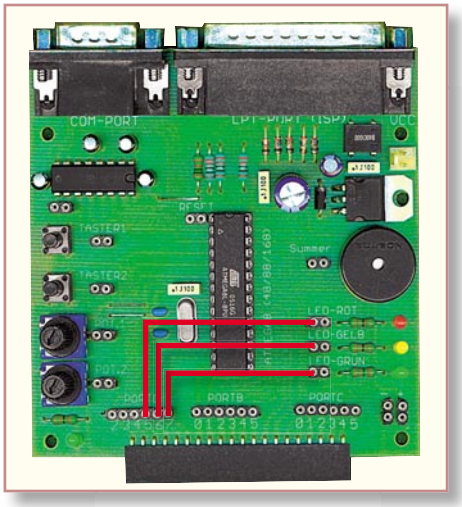

Bild 60: Die Verdrahtung des myAVR-Boards für das BASCOM-Lauflicht-Programm