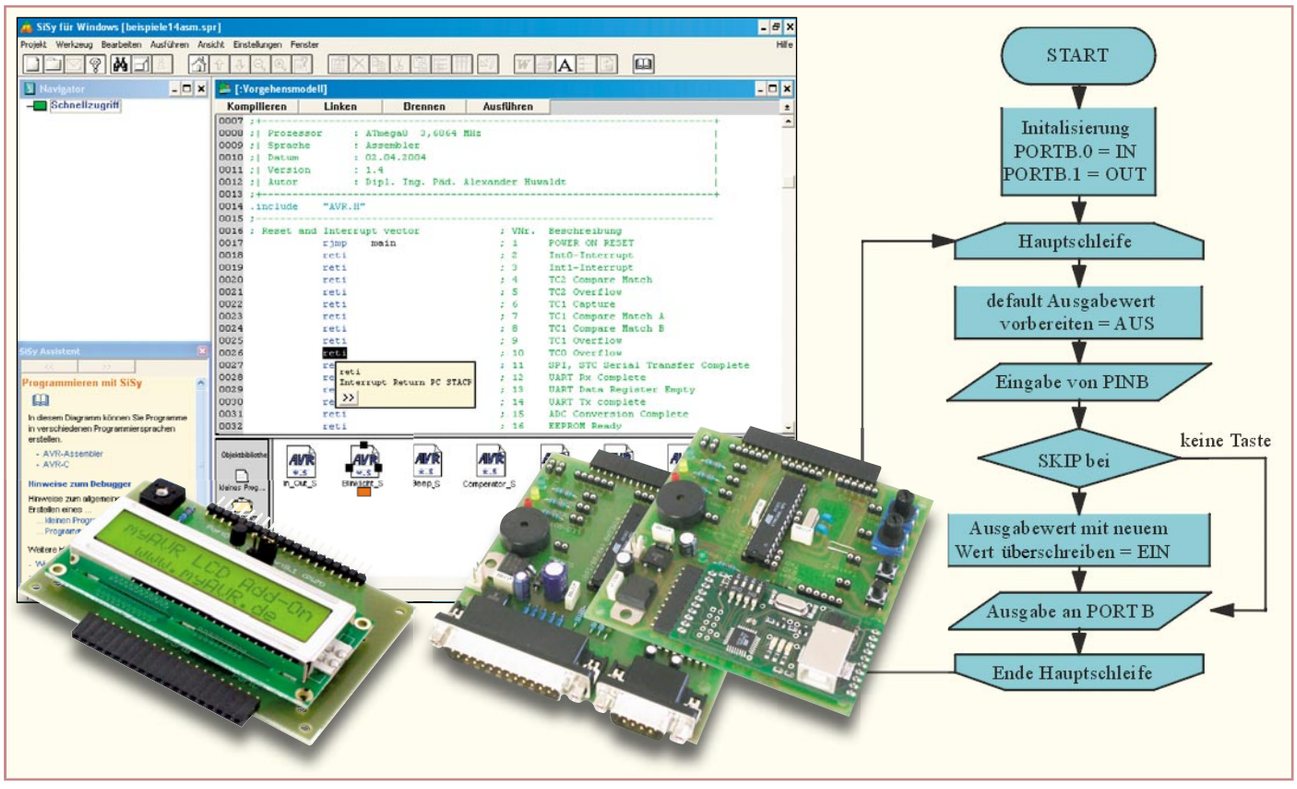

Mikrocontroller-Einstieg mit myAVR Teil 1

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, Software-Paket, Kabel, sämtliches Zubehör. Wir arbeiten uns im Verlaufe der Artikelserie anhand dieses Sets Schritt für Schritt in den Umgang mit dem AVR-Prozessor sowie den Programmiersprachen ein und zeigen anhand zahlreicher praktischer Anwendungen, dass es durchaus nicht kompliziert ist, in die Welt der Mikrocontroller einzusteigen. Der erste Teil stellt das Experimentiersystem vor und gibt einen Einblick in den Aufbau und die Funktion des behandelten Prozessors bis hin zu dessem Befehlssatz. Zusätzlich gewinnen wir einen ersten Einblick in die Entwicklungsumgebung „SiSy“.

ATMELs beliebte Problemlöser

Die AVR-8-Bit-RISC-Controller der Reihen ATtiny und ATmega von ATMEL erfreuen sich bei Elektronikern einer hohen Beliebtheit. Durch ihre kompakte und vielseitige Architektur erweisen sie sich als einfach zu handhabende Problemlöser für viele Aufgaben, zumal ein großer Teil der benötigten Hardware, wie A/D-D/A-Wandler, UART usw. bereits Bestandteil des dennoch kompakten Controllers sind. Programmierer schätzen die einfach zu handhabende und übersichtliche Art der Programmierung, eines der Hauptargumente vieler Programmierer, den AVR gegenüber dem ansonsten funktionell nahen Verwandten PIC vorzuziehen. Entsprechend findet man die AVR-Controller in zunehmend mehr Schaltungslösungen – sie sorgen für ein übersichtliches und ökonomisches Hardware-Design und sind auch von Programmier-Einsteigern einfach zu programmieren.

Bereits vor 5 Jahren haben wir uns ausführlich mit den Grundlagen der AVR-Architektur und -Programmierung beschäftigt. Seither hat sich viel getan. ATMEL hat die Angebotsbreite der beliebten Controller enorm erweitert, die Entwicklungs-Tools sind immer besser handhabbar, und man findet für jedes zu lösende Problem den passenden Controller.

Der Hersteller bietet für den professionellen Entwickler mehrere Entwicklungs-Kits an, großer Beliebtheit erfreut sich z. B. das AVR-Starter-Kit STK 500.

Wie gesagt, derartige Werkzeuge wenden sich jedoch an den Profi-Entwickler, also an den, der etwa schon Programmiersprachen wie Assembler oder C beherrscht und außer einer passenden Entwicklungs-

AVR kl-MB.indd 76 15.03.2006 10:05:59 Uhr

umgebung kaum weitere Mittel zur Programmierung benötigt.

Anders hingegen stellt sich eine solche professionelle Entwicklungsumgebung dem Einsteiger, dem Auszubildenden, Studierenden oder ganz einfach nur dem bisherigen „Hardware"-Elektroniker, der auch gern in die Welt des Mikrocontrollers einsteigen möchte, dar – es wird schlicht alles benötigte Wissen und viel Erfahrung vorausgesetzt. Und manche Werkzeuge sprengen auch den finanziellen Rahmen, den diese Klientel sich setzen kann bzw. will. Ergo kapitulieren viele potentielle Anwender vor der Hürde, ähnlich, wie es vor Jahren bei den PICs zu beobachten war.

SiSy® und myAVR – Lösung für Einsteiger

Dieses Problems hat sich die sächsische Firma Laser & Co. Solutions GmbH angenommen. Das Credo: einfach zu verstehende Software entwickeln, was sich im Markenzeichen „SiSy" manifestiert: „Simple-System". Diesem Credo verpflichtet, bietet Laser & Co. nun das „myAVR"-System an, ein Mikrocontroller-Experimentier- und Lernsystem für Hobby, Lehre, Studium und Beruf. Es besteht aus einer Reihe von einzelnen Hardware-Komponenten, der komplett deutschen Entwicklungsumgebung „SiSy AVR" und umfangreichem Lehrmaterial, das Schritt für Schritt, nur sehr wenige Grundkenntnisse voraussetzend, in die Programmierung der AVR-Controller einführt. Jeder Schritt kann praktisch am passenden myAVR-Board nachvollzogen werden, so dass man sehr schnell zu einem Erfolgserlebnis kommt, das zur Lösung der gestellten Übungsaufgaben anspornt. Einschränkend ist hier allenfalls zu vermerken, dass einige wenige Begriffe der Programmierung bekannt sein müssen, eine frühere Begegnung etwa mit BASIC macht den Einstieg leichter und man muss nicht erst Begriffe wie z. B. „Variable" lernen. Wird das System in der Lehre verwendet, wird diese Einschränkung obsolet, da hier ohnehin eine Einführung in Grundsatzbegriffe und -techniken erfolgt. Der Selbst-Studierende wird ebenfalls über geeignetes Lehr- bzw. Fachliteratur-Material verfügen, um diesen minimalen Voraussetzungen gerecht zu werden.

Bild 1: Das AVR-Starter-Kit enthält alles, was man zum Start in die Welt des Mikrocontrollers benötigt.

Dem, der bereits in Assembler, C oder BASCOM programmieren kann, wird hier ein besonders einfach zu handhabendes Werkzeug in die Hand gegeben, um schnell zum erwarteten Ergebnis, einem lauffähigen und direkt an der Hardware erprobten Programm, zu gelangen.

Das AVR-Starter-Kit (Abbildung 1) enthält faktisch alles, was man zum Einstieg benötigt, bis hin zum kompletten Kabelsatz. Der recht günstige Preis spricht für sich, immerhin erhält man hierfür ein komplett deutschsprachiges Entwicklungs-Kit und sämtliche erforderliche Hardware dazu bis hin zu einem ATmega-Controller.

Hauptbestandteile des Kits sind die Entwicklungsumgebung „SiSy AVR" auf CD-ROM (mit deutschem Benutzerhandbuch), das auf die Software und das AVR-Board zugeschnittene AVR-Lehrbuch mit ca. 200 Seiten und schließlich das „myAVR-Board" (Abbildung 2), das neben einem ATmega8-Controller eine Reihe für alle denkbaren Versuche nützlicher Peripherie enthält.

Dazu ergänzen Kabel, Batterie, Arbeitsblätter, Schnelleinstieg usw. den Lieferumfang – es kann sofort losgehen, nur ein PC, je nach Kit mit LPT- oder USB-Schnittstelle, wird noch benötigt.

Das Plus-Set der Reihe enthält zusätzlich einen direkt an das AVR-Board ansteckbaren Bausatz für die Programmierung der beliebten ATtiny-Controller (inklusive einem ATtiny12) und als Highlight ein komplettes, ebenfalls an das AVR-Board ansteckbares LCD-Board mit einem zweizeiligen, beleuchtbaren LC-Display (Abbildung 3). Dazu gibt es ein umfangreiches Lehrheft zur LCD-Programmierung, das auf ca. 50 Seiten wirklich alles zur LCD-Programmierung, auch in verschiedenen

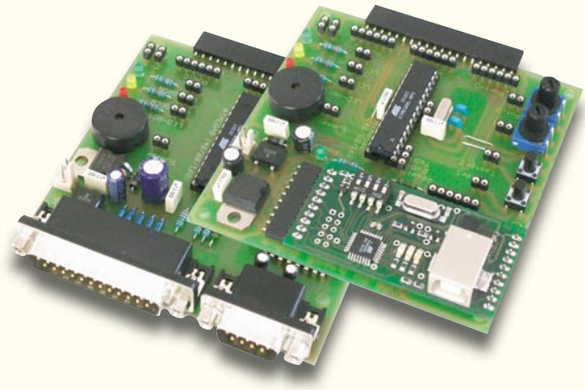

Bild 2: Das myAVR-Board ist als Seriell-/Parallelport-Version und mit USB-Schnittstelle erhältlich.

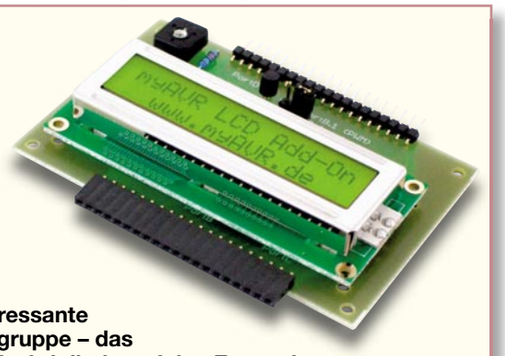

Bild 3: Interessante Zusatzbaugruppe – das LCD-Board wird direkt auf den Expansionsport des myAVR gesteckt. Das Board ist nebst LCD-Programmierlehrgang in der Plus-Version des AVR-Starter-Kits enthalten.

AVR kl-MB.indd 77 15.03.2006 10:06:03 Uhr

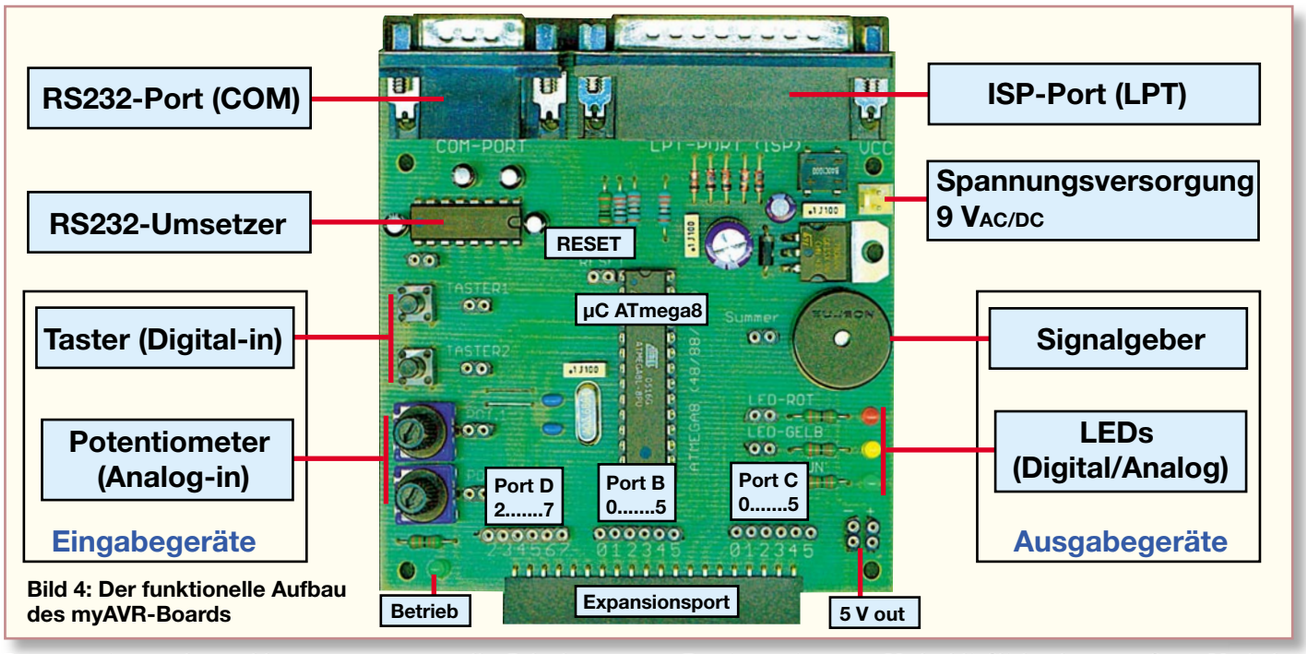

Bild 4: Der funktionelle Aufbau des myAVR-Boards

Programmiersprachen, erklärt.

Für weitergehende Experimente und Lösungen sind eine Reihe von Hardware-Komponenten verfügbar, so etwa direkt ansteckbare Laborkarten oder ein Metalldetektor-Projekt.

Mit Hilfe der zahlreichen Beispielprogramme ist es dem, der es ganz schnell wissen will, möglich, in wenigen Schritten sein erstes Programm zu laden, zu kompilieren, auf den Controller zu übertragen und zu testen. Anschließend entsteht automatisch der Wunsch, zu verstehen, wie es funktioniert, wie man es anpassen, verbessern und erweitern kann.

Besonders für Lehrpersonal ist diese Komplettlösung ideal, kann man doch kostengünstig und unter für jeden Auszubildenden völlig gleicher Hard- und Software-Ausstattung den Einstieg in die Mikrocontroller-Programmierung lehren. Für Lehrer sind übrigens speziell zusammengestellte Paketlösungen, z. B. mit erweiterter Hardware-Ausstattung, verfügbar.

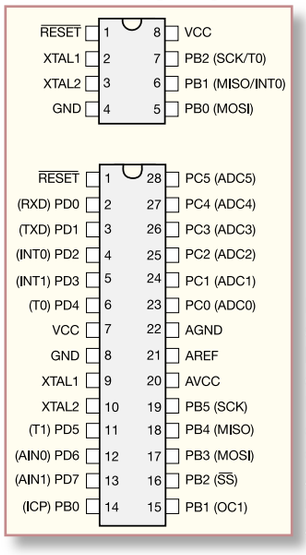

Bild 5: Anschlussbelegung der AVR-Mikrocontroller, oben ATtiny, unten ATmega8

Werfen wir zunächst einen Blick auf die Experimentier-Hardware, wozu auch das Wichtigste zur eingesetzten Controllerfamilie gehört, bevor wir uns der Entwicklungsumgebung und ersten Software-Projekten widmen. Dabei wollen wir allerdings nicht das umfangreiche Handbuch zum System ersetzen, dessen Inhalt führt über das Anliegen unserer Artikelserie weit hinaus, sowohl von der Grundlagenseite her als auch von der Projektausführung. Auch überlassen wir den ausführlichen Assemblerkurs dem Lehrbuch. Vielmehr unternehmen wir einen Exkurs durch das Thema mit aktuellen Bezügen zur Entwicklung des Systems.

Das Experimentier-Board „myAVR“

Das „myAVR“ ist in zwei Versionen verfügbar, die sich in der Art des Anschlusses an den PC unterscheiden. Während die eine Version den Kontakt über eine parallele und serielle Schnittstelle herstellt, erfüllt die zweite Version diese Aufgabe per USB-Port.

Zentraler Part des Boards (Abbildung 4) ist natürlich der auf einer Steckfassung untergebrachte ATmega8-Controller, der dank seiner In-System-Programmierbarkeit (Kürzel: ISP) direkt hier programmierbar ist.

Die Spannungsversorgung erfolgt über eine 9-V-Spannungsquelle. Eine Spannungsregler-Baugruppe sorgt für eine stabile 5-V-Betriebsspannung auf dem Board. Eine Pegelwandler-Baugruppe mit dem MAX 232 realisiert die Umsetzung zwischen der seriellen Schnittstelle des Mikrocontrollers und der RS232-Schnittstelle. Und schließlich gehört zur „festen Verdrahtung“ des Boards noch die Takterzeugung via 3,686411-MHz-Quarz.

Jeweils über die beigelegten Drahtbrücken und Buchsenleisten werden die Ports des Mikrocontrollers entsprechend der zu lösenden Aufgabe mit den Leuchtdioden, dem Piezo-Signalgeber, Tasten und Potentiometern auf der Platine verbunden, die als E/A-Bauteile dienen. All diese Bauteile reichen, um den 200-seitigen Mikrocontroller-Lernkurs ohne weitere Hardware zu bewältigen. Alle Ports des Controllers sind über eine Buchsenleiste herausgeführt, an die weitere Baugruppen, z. B. die LCD-Baugruppe oder eine eigene Applikation, einfach ansteckbar sind.

Der Mikrocontroller

Da man für die jeweils anstehende Aufgabenlösung nicht ohne Grundkenntnisse über den Aufbau und die Arbeitsweise des Mikrocontrollers auskommt, wollen wir uns zunächst diesem zuwenden. Der hier betrachtete ATmega8-Controller ist in der grundsätzlichen Architektur all seinen AVR-Verwandten ähnlich. Die Controllerbaureihe unterscheidet sich im Wesentlichen nur durch die Anzahl der I/O-Ports und die Speicherausstattung von seinen „großen“ Verwandten. Auch der ATtiny-Mini-Controller passt in dieses Schema. Die Anschlussbelegung der beiden Controller, die auf dem Board vorhanden bzw. im ATtiny-Programmierbausatz enthalten sind, ist in Abbildung 5 zu sehen.

Zum Grundverständnis des Controller-Aufbaus werfen wir einen Blick auf Abbildung 6. Hier sind die Hauptbaugruppen eines Mikrocontrollers zu sehen. Der entscheidende Unterschied zu einer reinen

AVR kl-MB.indd 78 15.03.2006 10:06:06 Uhr

Zentraleinheit (CPU, Prozessor), wie wir sie etwa vom PC her kennen, ist die Ausstattung mit weiteren Bausteinen wie Speicher, analogen und digitalen I/O-Baugruppen, Schnittstellencontrollern, Timern usw. Natürlich darf man die Rechenleistung und den Befehlsumfang der kleinen Zentraleinheit eines solchen Controllers nicht mit der einer PC-CPU vergleichen, für die Ausführung der für diese Controller spezifischen Aufgaben sind Rechenleistung und Befehlsumfang (hieraus leitet sich auch der Begriff RISC-Prozessor ab: Reduced Instruction Set Computer – eingeschränkter Befehlssatz) jedoch ausreichend. Nur durch diese Architektur sind Mikrocontroller-Baugruppen extrem kompakt (und stromsparend) aufzubauen, man benötigt kaum Peripherie. Da der Befehlsaufbau mit nur wenigen, einfach strukturierten Befehlen (Tabelle 1 am Ende des Artikels zeigt eine Kurzübersicht aller von den AVR-Mikrocontrollern unterstützten Befehle, hierauf kommen wir im Verlauf der Serie zurück) erfolgt, konnte man den Rechnerkern auf hohe Rechengeschwindigkeit optimieren, so sind die Controller auch bis hin zu Echtzeitaufgaben einsetzbar. Dazu kommt eine auf die Controller-Architektur speziell zugeschnittene Programm-Übersetzung und -Bearbeitung, die die Abarbeitungsgeschwindigkeit von Befehlen nochmals erhöht – RISC-Prozessoren sind in der Welt der Ein-Chip-Rechner die schnellsten Prozessoren!

Abbildung 7 zeigt bereits ein komplexeres Bild – den Aufbau unseres verwendeten AVR ATmega8. Hier finden sich alle besprochenen Baugruppen in detaillierter Aufteilung wieder.

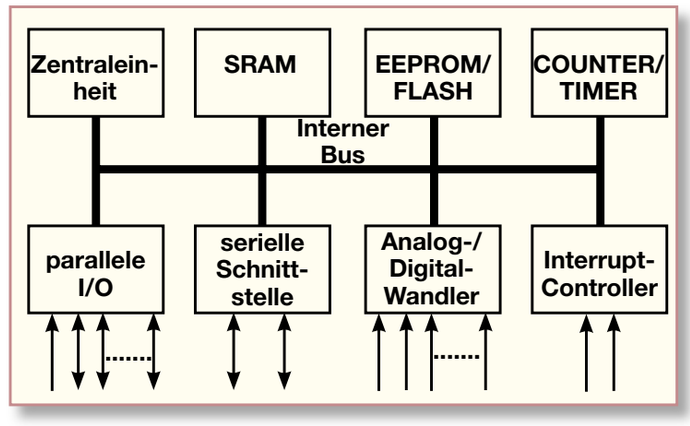

Bild 6: Grundaufbau eines Mikrocontrollers

Register, RAM, ROM, Flash …

Wichtig für das Verständnis bei der späteren Programmierung ist hier noch der etwas detailliertere Blick in die Speicherausstattung. Mikrocontroller verfügen über mehrere Speicherarten, die jeweils unterschiedliche Aufgaben haben.

Direkt im Kontakt mit dem Prozessorkern, dem Rechenwerk (ALU), verbunden sind die Register. Diese dienen der Verwaltung des Befehlscodes, sie sind die Operanden und Ergebnisspeicher für die direkten Maschinenbefehle des Rechenwerks. Die einzelnen Register haben jeweils bestimmte Aufgaben, wie wir noch im Verlaufe der Programmentwürfe sehen werden. Wichtig zu wissen ist die Tatsache, dass die Speicherinformationen der Register beim Abschalten der Betriebsspannung gelöscht sind!

Das sind auch die des SRAM-Datenspeichers. Der dient zur Speicherung der bei der Abarbeitung von Programmen anfallenden Programmdaten sowie aller Informationen, die nicht fest im eigentlichen Programm des Controllers verankert sind, etwa auch von außen erfasster bzw. eingegebener oder zwischengespeicherter Daten. Seine Lese- und Schreibzeiten sind sehr kurz, so dass der SRAM der schnellste Speicher des hier besprochenen Verbunds ist.

Die dritte Klasse von Speichern sind die nicht flüchtigen Speicher. Diese kommen als ROM oder (E)EPROM vor. Sie behalten grundsätzlich einmal hier gespeicherte Informationen, auch bei Abschalten der Betriebsspannung, und eignen sich so z. B. für die Speicherung des Betriebsprogramms eines Controllers, das beim Wiedereinschalten sofort zur Verfügung steht.

Im ROM werden Daten fest gespeichert,

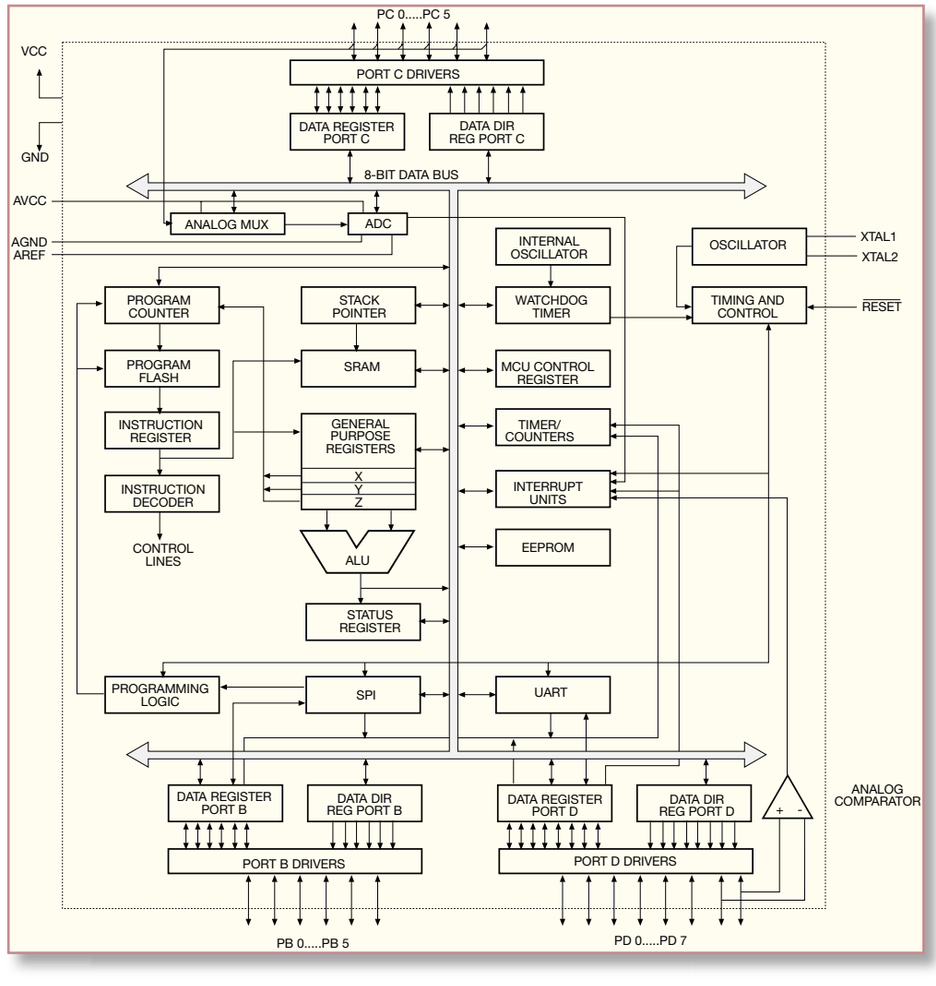

Bild 7: Blockschaltbild des ATmega8

AVR kl-MB.indd 79 15.03.2006 10:06:08 Uhr

Tabelle 1: Kurzübersicht aller von den AVR-Mikrocontrollern unterstützten Befehle

Arithmetik- und Logikbefehle

Mnemonics	Operands	Description
ADD	Rd, Rr	Add without Carry
ADC	Rd, Rr	Add with Carry
ADIW	Rd, K	Add Immediate to Word
SUB	Rd, Rr	Subtract without Carry
SUBI	Rd, K	Subtract Immediate
SBC	Rd, Rr	Subtract with Carry
SBCI	Rd, K	Subtract Immediate with Carry
SBIW	Rd, K	Subtract Immediate from Word
AND	Rd, Rr	Logical AND
ANDI	Rd, K	Logical AND with Immediate
OR	Rd, Rr	Logical OR
ORI	Rd, K	Logical OR with Immediate
EOR	Rd, Rr	Exclusive OR
COM	Rd	One's Complement
NEG	Rd	Two's Complement
SBR	Rd,K	Set Bit(s) in Register
CBR	Rd,K	Clear Bit(s) in Register
INC	Rd	Increment
DEC	Rd	Decrement
TST	Rd	Test for Zero or Minus
CLR	Rd	Clear Register
SER	Rd	Set Register
MUL	Rd,Rr	Multiply Unsigned
MULS	Rd,Rr	Multiply Signed
MULSU	Rd,Rr	Multiply Signed with Unsigned
FMUL	Rd,Rr	Fractional Multiply Unsigned
FMULS	Rd,Rr	Fractional Multiply Signed
FMULSU	Rd,Rr	Fractional Multiply Signed with Unsigned

Befehle zum Datentransfer

Mnemonics	Operands	Description
MOV	Rd, Rr	Copy Register
MOVW	Rd, Rr	Copy Register Pair
LDI	Rd, K	Load Immediate
LDS	Rd, k	Load Direct from data space
LD	Rd, X	Load Indirect
LD	Rd, X+	Load Indirect and Post-Increment
LD	Rd, -X	Load Indirect and Pre-Decrement
LD	Rd, Y	Load Indirect
LD	Rd, Y+	Load Indirect and Post-Increment
LD	Rd, -Y	Load Indirect and Pre-Decrement
LDD	Rd,Y+q	Load Indirect with Displacement
LD	Rd, Z	Load Indirect
LD	Rd, Z+	Load Indirect and Post-Increment
LD	Rd, -Z	Load Indirect and Pre-Decrement
LDD	Rd, Z+q	Load Indirect with Displacement
STS	k, Rr	Store Direct to data space
ST	X, Rr	Store Indirect
ST	X+, Rr	Store Indirect and Post-Increment
ST	-X, Rr	Store Indirect and Pre-Decrement
ST	Y, Rr	Store Indirect
ST	Y+, Rr	Store Indirect and Post-Increment
ST	-Y, Rr	Store Indirect and Pre-Decrement
STD	Y+q,Rr	Store Indirect with Displacement
ST	Z, Rr	Store Indirect
ST	Z+, Rr	Store Indirect and Post-Increment
ST	-Z, Rr	Store Indirect and Pre-Decrement
STD	Z+q,Rr	Store Indirect with Displacement
LPM		Load Program Memory
LPM	Rd, Z	Load Program Memory
LPM	Rd, Z+	Load Program Memory and Post-Increment
ELPM		Extended Load Program Memory
ELPM	Rd, Z	Extended Load Program Memory
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment
SPM		Store Program Memory
ESPM		Extended Store Program Memory
IN	Rd, A	In From I/O Location
OUT	A, Rr	Out To I/O Location I/O
PUSH	Rr	Push Register on Stack
POP	Rd	Pop Register from Stack

Sprung-Anweisungen

Mnemonics	Operands	Description
RJMP	k	Relative Jump
IJMP		Indirect Jump to (Z)
EIJMP		Extended Indirect Jump to (Z)
JMP	k	Jump
RCALL	k	Relative Call Subroutine
ICALL		Indirect Call to (Z)
EICALL		Extended Indirect Call to (Z)
CALL	k	Call Subroutine
RET		Subroutine Return
RETI		Interrupt Return
CPSE	Rd,Rr	Compare, Skip if Equal
CP	Rd,Rr	Compare
CPC	Rd,Rr	Compare with Carry
CPI	Rd,K	Compare with Immediate
SBRC	Rr, b	Skip if Bit in Register Cleared
SBRS	Rr, b	Skip if Bit in Register Set
SBIC	A, b	Skip if Bit in I/O Register Cleared
SBIS	A, b	Skip if Bit in I/O Register Set
BRBS	s, k	Branch if Status Flag Set
BRBC	s, k	Branch if Status Flag Cleared
BREQ	k	Branch if Equal
BRNE	k	Branch if Not Equal
BRCS	k	Branch if Carry Set
BRCC	k	Branch if Carry Cleared
BRSH	k	Branch if Same or Higher
BRLO	k	Branch if Lower
BRMI	k	Branch if Minus
BRPL	k	Branch if Plus
BRGE	k	Branch if Greater or Equal, Signed
BRLT	k	Branch if Less Than, Signed
BRHS	k	Branch if Half Carry Flag Set
BRHC	k	Branch if Half Carry Flag Cleared
BRTS	k	Branch if T Flag Set
BRTC	k	Branch if T Flag Cleared
BRVS	k	Branch if Overflow Flag is Set
BRVC	k	Branch if Overflow Flag is Cleared
BRIE	k	Branch if Interrupt Enabled
BRID	k	Branch if Interrupt Disabled

Bit- und Bit-Test-Befehle

Mnemonics	Operands	Description
LSL	Rd	Logical Shift Left
LSR	Rd	Logical Shift Right
ROL	Rd	Rotate Left Through Carry
ROR	Rd	Rotate Right Through Carry
ASR	Rd	Arithmetic Shift Right
SWAP	Rd	Swap Nibbles
BSET	s	Flag Set
BCLR	s	Flag Clear
SBI	A, b	Set Bit in I/O Register
CBI	A, b	Clear Bit in I/O Register
BST	Rr, b	Bit Store from Register to T
BLD	Rd, b	Bit load from T to Register
SEC		Set Carry
CLC		Clear Carry
SEN		Set Negative Flag
CLN		Clear Negative Flag
SEZ		Set Zero Flag
CLZ		Clear Zero Flag
SEI		Global Interrupt Enable
CLI		Global Interrupt Disable
SES		Set Signed Test Flag
CLS		Clear Signed Test Flag
SEV		Set Two's Complement Overflow
CLV		Clear Two's Complement Over-flow
SET		Set T in SREG
CLT		Clear T in SREG
SEH		Set Half Carry Flag in SREG
CLH		Clear Half Carry Flag in SREG
NOP		No Operation
SLEEP		Sleep
WDR		Watchdog Reset

AVR kl-MB.indd 80 15.03.2006 10:06:09 Uhr

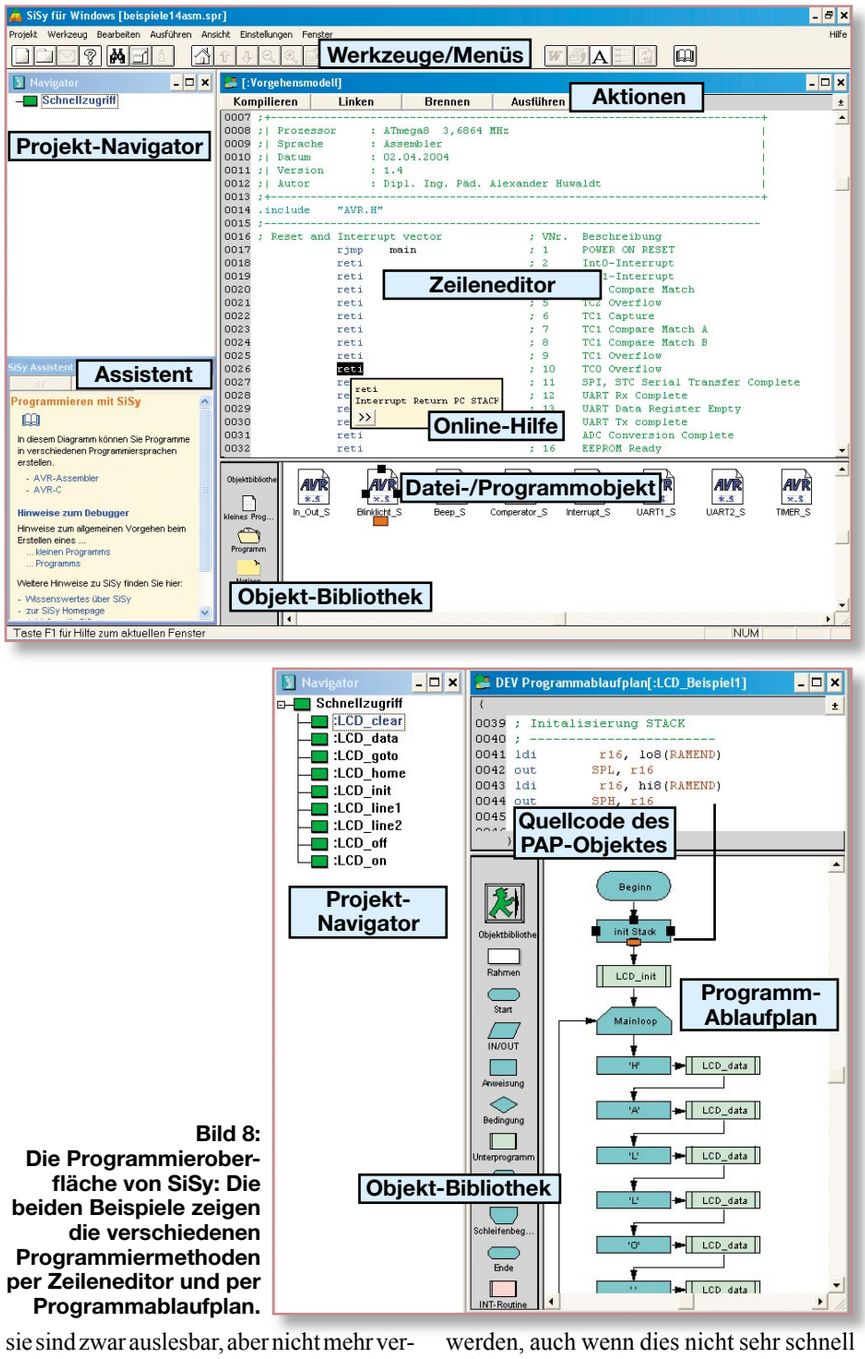

Bild 8: Die Programmieroberfläche von SiSy: Die beiden Beispiele zeigen die verschiedenen Programmiermethoden per Zeileneditor und per Programmablaufplan.

sie sind zwar auslesbar, aber nicht mehr veränderbar. Der ROM ist ein Einmal-Speicher – einmal mit Daten geladen, ist der Festspeicher nicht mehr löschbar.

Ganz im Unterschied zum EPROM. Dieses ist zwar ebenfalls ein Festwertspeicher wie das ROM, es ist jedoch durch Bestrahlung eines Löschfensters im Gehäuse per UV-Licht löschbar und daraufhin wieder neu beschreibbar. Die Zahl der Lösch-/Schreib-Vorgänge ist jedoch begrenzt.

Der moderne Ableger des EPROMs ist das elektrisch löschbare EEPROM. Es ist von außen mit einem elektrischen Impuls blockweise löschbar und bis zu hunderttausend Mal wieder neu beschreibbar. Somit kann es sehr einfach mehrfach beschrieben werden, auch wenn dies nicht sehr schnell erfolgt. Diese geringe Lese- und Schreibgeschwindigkeit ist bei der Konzipierung des Programms zu beachten.

Deutlich schneller arbeiten die modernsten nicht flüchtigen Speicher – die Flash-Speicher. Sie sind schnell, behalten ihre Daten auch beim Abschalten der Spannungsversorgung und sind immer wieder beschreibbar, wenn auch nicht so oft wie die EEPROMs. Über ihre serielle Schnittstelle eignen sie sich hervorragend für die Programmierung des Controllers, etwa von einem PC aus.

Bis auf das ROM finden wir alle besprochenen Speicherarten in unserem AVR ATmega8 wieder, wie Abbildung 7 beweist.

Das soll es zunächst in puncto Hardware gewesen sein, widmen wir uns nun der Software bzw. der Programm-Entwicklungsumgebung.

Die Entwicklungsumgebung SiSy

Bevor wir die eigentliche Entwicklungsumgebung betrachten, müssen wir uns den üblichen Gang der Programm-„Produktion“ vergegenwärtigen – die folgenden Schritte werden uns immer wieder begleiten. Zuerst wird der Quellcode in einem geeigneten (ASCII-Text-)Editor geschrieben. Hier erfolgt die Zusammenstellung der einzelnen Befehle, von der Initialisierung des Controllers über die Zusammenstellung der Unterprogramme bis zum Abschluss des Programms.

Dieser Quellcode muss anschließend in die so genannte Maschinensprache übersetzt werden, einen Code, der vom Rechenwerk des Controllers direkt „verstanden“ wird und durch diesen abarbeitbar ist. Diese Übersetzung übernimmt ein Compiler. Er prüft auch den Quellcode auf syntaktische Fehler. Um mehrere Objektdateien (Programmteile) zu einer ausführbaren Programmdatei zusammenzufügen, kommt ein so genannter Linker zum Einsatz.

Zur Programmierumgebung gehören auch Testwerkzeuge, die eine Programmlaufsimulation sowie eine systematische Fehlersuche ermöglichen, Simulatoren und Debugger.

Und schließlich gehören noch Werkzeuge für die Übertragung des Maschinencodes in das EEPROM oder, wie bei unserem ATmega8, in den Flash-Speicher des Controllers hierzu. Bei manchen Programmierumgebungen, wie auch bei „SiSy“, sind die Schritte Linken (Übersetzen der Objektdatei in eine für den Controller ausführbare Hexdatei) und Speichern auf den Flash-Speicher („Brennen“) getrennt, bei anderen hingegen ist dies nur ein Werkzeug, das lediglich Rückfragen vor dem Brennen stellt.

Kommen wir damit zu „SiSy“, einem so genannten Modellierungswerkzeug, das für die objektorientierte Programmierung von Branchenlösungen entwickelt wurde. Unsere hier genutzte Entwicklungsumgebung ist ein Add-on für dieses Modellierungswerkzeug, das speziell für die AVR-Programmierung entwickelt wurde. Es erlaubt sowohl die einfache Programmentwicklung über einen üblichen Programmeditor als auch die Generierung eines Programms über einen grafischen Programmablaufplan (PAP). Abbildung 8 vermittelt einen ersten Eindruck dieser beiden Vorgehensweisen anhand der Bedienoberfläche von SiSy.

Diese werden wir anhand erster Programmierschritte in der nächsten Ausgabe eingehender kennen lernen. ELV

AVR kl-MB.indd 81 15.03.2006 10:06:11 Uhr