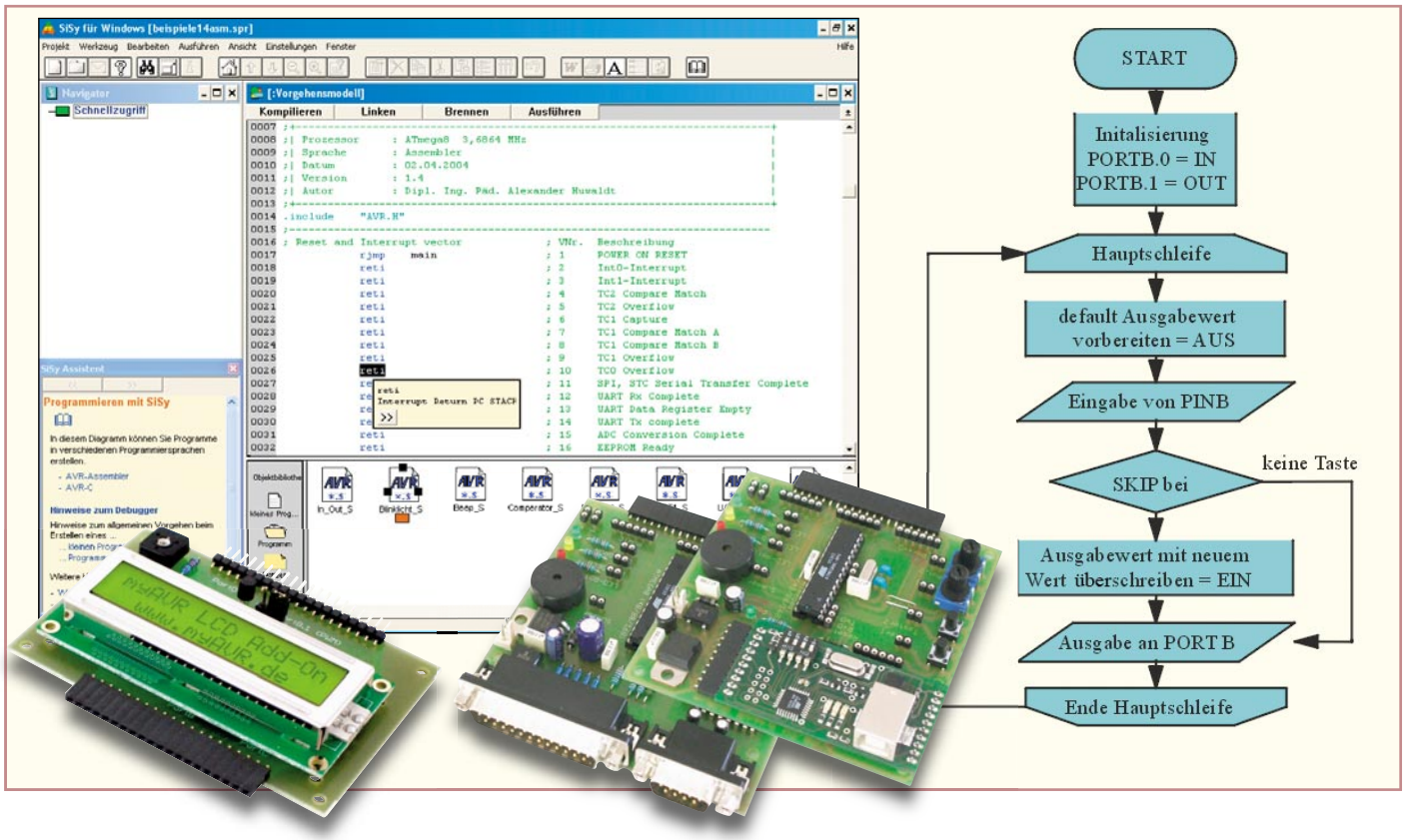


Mikrocontroller-Einstieg mit myAVR Teil 7

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 siebtenTeil unserer Serie zur Programmierung des AVR besprechen wir die Grundlagen der Programmierung zur Ansteuerung von LC-Displays.

Punkt für Punkt – LC-Displays

LC-Displays gibt es in zahlreichen Formen, hauptsächlich sind dies die anwendungskonfigurierten Displays, die einem festen Verwendungszweck zugeordnet sind und nur bedingt anderweitig nutzbar sind, die Punktmatrix-Displays, die auf unterschiedlichen Zeilenzahlen und Zeilenlängen meist Zeichen in einer 5x7- bzw. 5x10-Punktmatrix darstellen können. Die dritte große Klasse sind die frei programmierbaren Grafik-Displays, die in horizontaler und vertikaler Richtung jeweils eine bestimmte Bildpunktzahl darstellen können. Dies kann einfarbig (z. B. bei vielen Messgeräten) oder mehrfarbig (z. B. bei TFT-Farbbildschirmen) erfolgen. Für einfache Text- und Pseudografik-Anzeigen genügen jedoch die beschriebenen alphanumerischen Punktmatrix-Displays. Hier können Texte, Ziffern und Pseudografiken angezeigt werden, die in einem Zeichen-ROM des LCD-Controllers abgelegt sind. Zusätzlich verfügt der LCD-Controller über einen kleinen RAM-Bereich (CGRAM), in den selbst kreierte Zeichen ladbar und von dort abrufbar sind.

LCD-Controller? Ja, jedes Display verfügt über einen eigenen Controller, der dem Steuerrechner, in unserem Fall dem AVR, viel Arbeit abnimmt. Er empfängt Steuerbefehle und Daten und steuert darauf basierend die entsprechende Ausgabe auf der Anzeige aus. Das spart u. a. unzählige Leitungen, denn je Anzeigezeile sind 8 Zeilenleitungen und bis zu 80 Segmentleitungen zu verwalten. Für die Initialisierung und den Ablauf der Anzeige sind einige Zeitfaktoren einzuhalten, auch das erledigt zum Teil der LCD-Controller, der auf einer Platine unmittelbar hinter dem eigentlichen LCD-Panel sitzt. Nach außen hin agiert dieser Controller mit einer standardisierten Schnittstelle (Abbildung 1), die bei allen Displays dieser Art gleich ist. Hier taucht spätestens der Begriff „HD44780-kompatibel“ auf. Das bezeichnet den am häufigsten eingesetzten LCD-Controller HD44780 von Hitachi, der neben dem Samsung KS00xx quasi den Industriestandard bildet. Er kann Displays mit bis zu 2 Zeilen und jeweils 8 Zeichen ansteuern. Für größere Displays erfolgt eine Kaskadierung mehrerer Controller bzw. der Einsatz komplexerer Controller anderer Familien, die sich aber nach außen über genau die gleiche Schnittstelle zeigen wie ein einzelner Display-Controller.

Wie man Abbildung 1 entnehmen kann, genügen 11 Leitungen, um die Kommunikation zwischen Steuer- und Anzeige-

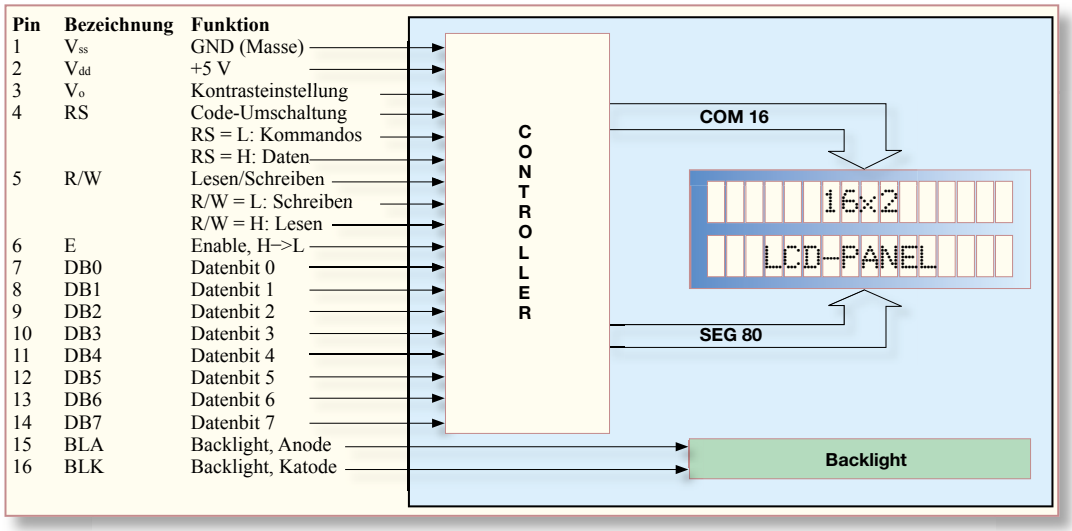


Bild 1: Standard-Anschlussbelegung der parallelen LCD-Schnittstelle und Aufbau des LCD-Moduls

controller zu regeln. Dazu kommen die Spannungsversorgung (V_{ss} [Masse], V_{dd} [Versorgungsspannung]), die Kontrasteinstellung des Displays (V_o) und die seit einigen Jahren zum Standard gehörende Displaybeleuchtung (BLA, BLK) mittels LEDs. Diese Art der Schnittstelle nennt man 8/4-Bit-Parallel-Interface, je nach Programmierung des Display-Controllers sind also die 8-Bit-Daten parallel als volles Byte (8 Bit) oder als zwei Halbbytes (je 4 Bit, Low- und High-Teil) übertragbar. Neben diesem Parallel-Interface gibt es noch mehrere Arten von seriellen Schnittstellen, die wir an dieser Stelle jedoch nicht näher betrachten wollen.

Wie die Schnittstelle in der Praxis aussieht, zeigt Abbildung 2. Hier sieht man die typischen 14- bis 16-poligen Anschlussreihen. Einige Displays erfordern den getrennten Anschluss der Hintergrundbeleuchtung, älteren Displays fehlt der ganz.

So einheitlich die Schnittstelle ist, so unterschiedlich fällt, Display für Display, die Zuordnung der Anzeigedaten (Display-Organisation) in dessen Speicher aus. Das stellt gerade den Einsteiger vor Probleme, wenn er für ein vorhandenes Programm ein abweichendes Display einsetzen möchte. Glücklicherweise liefert jeder Display-Hersteller ein Datenblatt, in dem die Speicherorganisation detailliert aufgeführt wird. So kann man das Beschreiben des Anzeige-RAMs im Display-Controller genau auf dessen Organisation ausrichten.

Zum Verständnis eines Displays samt seinem Controller gehört auch die zu jedem Display gehörende Befehls- und Timing-Tabelle (Tabelle 1 zeigt ein Beispiel). Hier findet man alle relevanten Display-Befehle, deren Erklärung und Ausführungszeiten. Letztere sind bei der Programmierung besonders wichtig, um das Zeitregime bei der Befehls- und Datenübermittlung einzuhalten. Denn der Display-Controller ist im Verhältnis zum Steuer-Controller, also unserem AVR, viel langsamer – darauf muss man sich einstellen!

In der Praxis diktiert die Abfolge von Steuerbefehlen und Daten das Geschehen an der Schnittstelle. In Abbildung 3 ist das vereinfachte Timing-Diagramm dazu gezeigt. Für die Arbeit mit dem AVR-Controller sind hier vier Dinge wichtig:

1. Es muss der Display-Controller unterscheiden können, ob jetzt ein Kommando oder Daten übertragen werden sollen. Dies erkennt er, wie wir in Tabelle 1 sehen, über den Zustand der Leitung RS.
2. Der Enable-Impuls, der dazu dient, das kommende Kommando bzw. die Daten in den Display-Controller-RAM zu übernehmen, muss zum richtigen Zeitpunkt gesendet werden. Mit dessen fallender Flanke erfolgt die Übernahme.
3. Für unsere AVR-Anwendungen ist kein Lesen aus dem Display-Controller nötig, deshalb liegt R/W immer auf „low“.
4. Da die meisten AVR-Controller keine Extra-Pins für den Anschluss eines Displays aufweisen (es gibt wenige Spezialtypen hierfür), nutzt man die Option des Display-Controllers, ihn im 4-Bit-Modus zu bedienen. Das erfordert dann auf der AVR-Seite nur wenige Controller-I/O-Pins für die Ausgabe auf das Display.

Da der AVR vergleichsweise flott getaktet ist, kann man bei I/O-Pin-Mangel zu einem Trick greifen: Während der Display-Controller noch damit beschäftigt ist, die geschickten Daten auf das Display zu bringen, kann man die betreffenden I/O-Pins des AVR „schnell“ auf Eingabe umschalten und z. B. zum Abfragen von Tastenzuständen nutzen. So können geschickte Programmierer auch Displays an AVRs mit sehr wenigen I/O-Pins betreiben.

Das myAVR-LCD-Add-on

Damit kommen wir zum speziell für die Nutzung am myAVR-Board konzi-

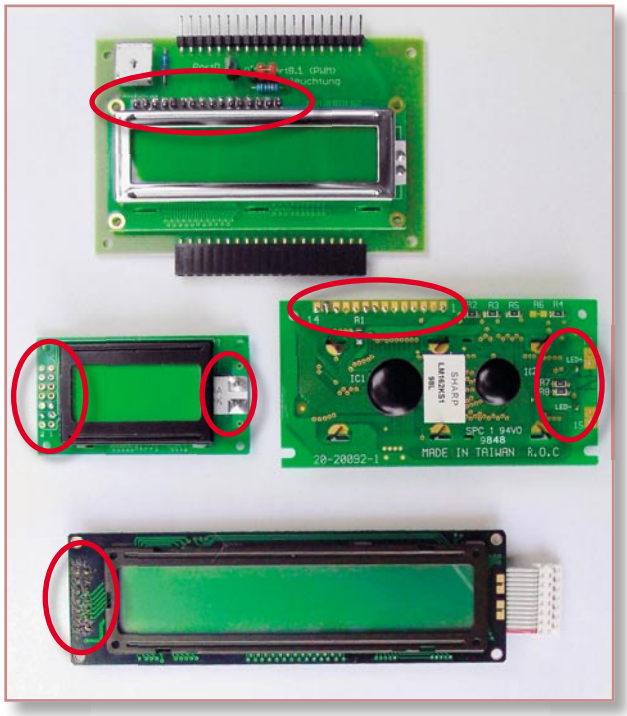


**Bild 2: Einige typische Displays mit ihren Anschlüssen.
Oben das myAVR-LCD-Add-on, dessen Display die typische einreihige, 16-polige Schnittstelle anbietet.
Darunter links ein 2x8-Display mit weit verbreiteter 14-poliger Schnittstelle und extra Backlight-Anschlüssen.
Rechts daneben ein Display mit einreihigem 14-poligen Anschluss und abgesetztem Backlight-Anschluss.
Unten ein älteres Display mit 14-poligem Anschluss, ohne Beleuchtung.**

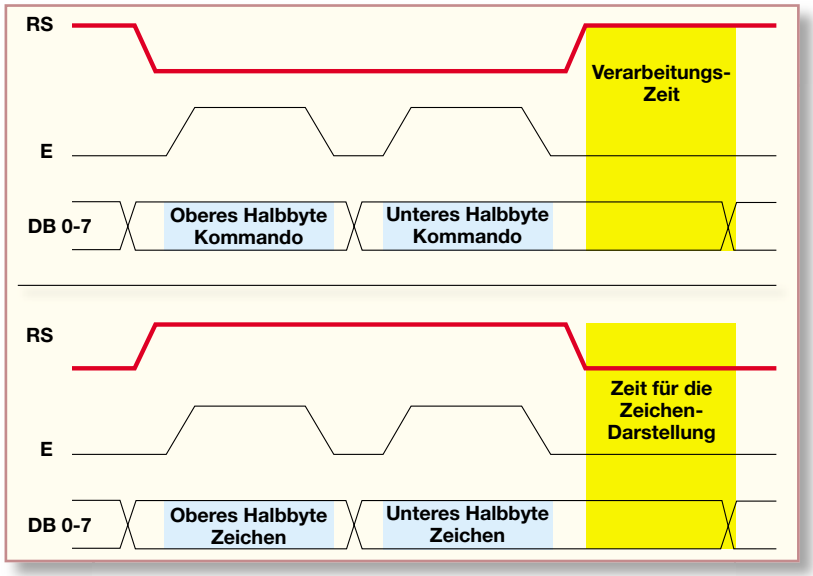
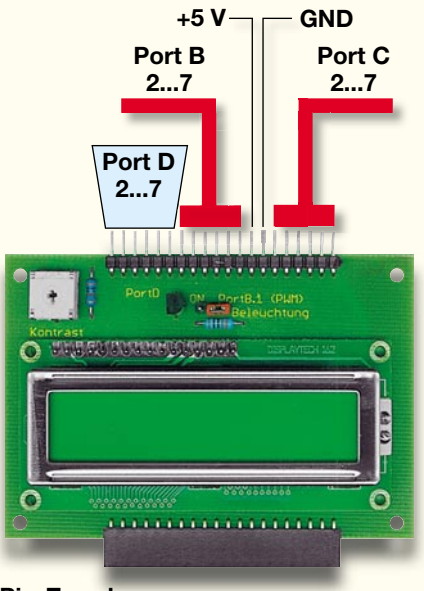


Bild 3: Die Zeitabläufe (Timing) beim Übertragen von Kommandos (oben, RS = 0) und Zeichen (unten, RS = 1). Bei jedem Enable-Impuls erfolgt die Übernahme eines Kommando- bzw. Zeichen-Halbbytes in den Display-Controller.

pierten myAVR-LCD-Add-on (Abbildung 4). Das kommt mit einigen der AVR-Kits oder ist preiswert einzeln als Bausatz oder betriebsfertige Platine erhältlich. Die trägt ein zweizeiliges, je 16-stelliges LC-Display mit Hintergrundbeleuchtung, das ein HD44780-kompatibles Interface besitzt. Der Kontrast ist über einen Trimmer auf der Platine einstellbar. Die Hintergrundbeleuchtung ist über einen Jumper wahlweise fest an +5 V gelegt oder, vom AVR gesteuert, schalt- bzw. dimmbar.



Pin-Zuordnung:

Port	Pin-Stiftleiste	LCD-Pin
D.2	1	4/RS
D.3	2	6/Enable
D.4	3	11/DB4
D.5	4	12/DB5
D.6	5	13/DB6
D.7	6	14/DB7
B.1	8	15/Bel. Anode
+5 V	13	1/V_{ss}
GND	14	2/V_{dd}

Bild 4: Das LCD-Board mit der Pin- und Port-Belegung für die LCD-Ansteuerung

Man erkennt sehr schnell, dass hier tatsächlich nur 4 Pins für die Daten- und Kommandoausgabe und 2 Pins zur Steuerung erforderlich sind, die Platine nutzt den Port D des AVR-Controllers. Will man die Beleuchtung vom AVR aus schalten oder dimmen, kommt noch Port B 1 hinzu. Somit stehen noch (fast) alle Pins der Ports B und C für andere Zwecke zur Verfügung. Die sind für weitere Add-on-Platinen zu einer Buchsenleiste durchgeschleift.

Die Folge ist, dass alle Programmierungen des LC-Displays im 4-Bit-Modus erfolgen müssen!

In der Begleitdokumentation zum myAVR-LCD-Add-on sowie im myAVR-LCD-Lernheft sind alle Informationen zur Hardware, so u. a. auch die Speicherorganisation des verwendeten Displays, der Zeichensatz und weitere Infos ausführlich dargestellt, weshalb wir hier nicht weiter darauf eingehen wollen. Wenden wir uns also der Programmierpraxis zu!

Die Initialisierung des Displays

Wie der AVR erfordert auch der Display-Controller eine Reihe von Maßnahmen, um ihn definiert zu starten – das Display ist zu initialisieren. Dieser Part ist äußerst wichtig, hier gibt es die meisten „unerklärlichen" Fehler im späteren Betrieb. Darum sind die Startbedingungen, die im AVR-Lehrmaterial noch einmal detailliert erläutert sind, akribisch zu beachten!

Zur Initialisierung sind Display-Controller und AVR einige Mitteilungen zu machen, die diese in die Lage versetzen, miteinander zu kommunizieren:

1. Der AVR ist so zu initialisieren, dass Port D als Ausgang geschaltet ist.
2. Dem Display-Controller ist eine Wartezeit entsprechend Datenblatt zu gewähren, damit er hochfahren kann.
3. Der Display-Controller ist in den 4-Bit-Mode zu schalten, um das bereits besprochene 4-Bit-Interface nutzen zu können (Bit 4 „DL" in der Zeile „Function Set" der Kommando-Übersicht auf 0 setzen).
4. Dem Display-Controller ist mit dem Befehl „Function Set" mitzuteilen, wie viele Zeilen des Displays (Bit 3 „N" in der Zeile „Function Set" der Kommando-Übersicht; 0 für 1 Zeile, 1 für beide Zeilen setzen) und welche Zeichenmatrix (Bit 2 „F" in der Zeile „Function Set" der Kommando-Übersicht; 0 für 5 x 7,

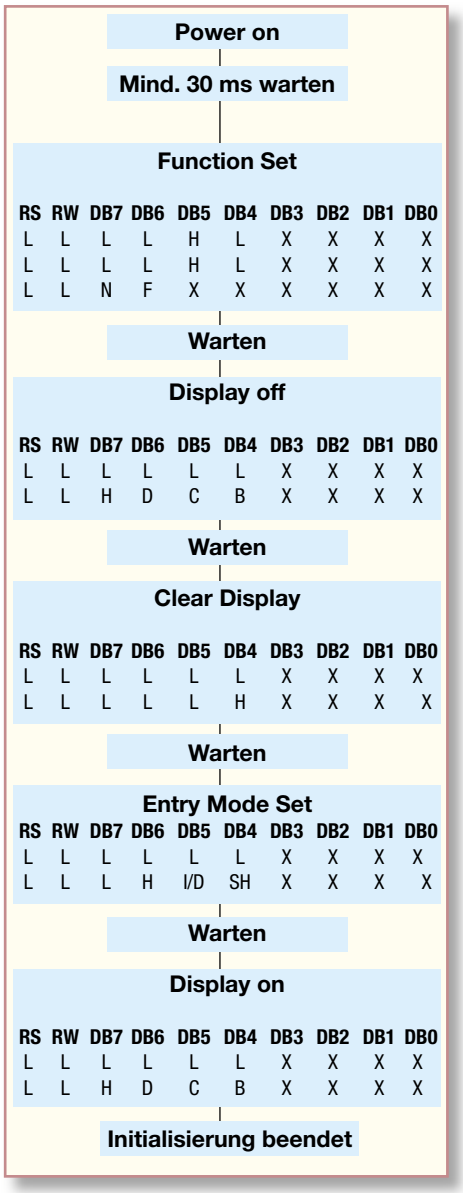


Bild 5: Der allgemeine Befehlsablauf bei der Display-Initialisierung (Befehle und Wartezeiten siehe Tabelle 1 und Assemblerlisting, zum Ablauf der Function-Set-Sequenz siehe Text)

1 für 5 x 10 setzen) verwendet werden soll. Wir werden beide Zeilen und die 5x7-Matrix verwenden.

5. Mit dem Befehl „LCD off“ (siehe Tabelle 1) wird das Display zunächst abgeschaltet, um keine undefinierten Ausgaben zu erhalten.
6. Mit dem Befehl „LCD clear“ (siehe Tabelle 1) erfolgt das Überschreiben des gesamten Display-Controller-RAMs mit Leerzeichen und das Rücksetzen des Display-Cursors auf das erste Zeichen der ersten Zeile. So ist der Start klar definiert.
7. Mit dem Kommando „Entry Mode Set“ (siehe Tabelle 1) wird der Einfügemodus für die Zeichen festgelegt.
8. Zuletzt wird mit dem Befehl „LCD on“ das Display eingeschaltet.

Der gesamte Ablauf ist in Abbildung 5 noch einmal zusammengefasst. Die einzelnen Programmteile in Assembler hierzu finden sich im Gesamtlisting als entsprechende Unterprogramme.

Zu beachten ist, dass der LCD-Controller zunächst ja auf den 8-Bit-Modus konfiguriert ist. Da wir aber nur 4 Datenleitungen nutzen und jedes Kommando, wie auch die Daten, in Form von zwei Halbbytes übertragen, geht „natürlich“ beim ersten Kommando „Function Set“ ein Halbbyte verloren. Deshalb ist das Kommando anschließend nochmals komplett zu übertragen und erscheint in Abbildung 5 so quasi zweieinhalb Mal. Wichtig sind auch die angegebenen Wartezeiten zwischen den Befehlen. Sie sind (aufgerundet und an die AVR-Teilerfaktoren angepasst) ebenfalls als Unterprogramme im Gesamtlisting untergebracht (siehe z. B. LCD-Enable-Interrupt und die Wait-Sequenzen, die immer wieder neu aufgerufen werden).

Das erste LCD-Projekt

Nachdem die Initialisierung geschafft ist, machen wir uns an das erste Programm, um einen kleinen Text auf dem Display anzeigen zu lassen.

Wenn wir das Gesamt-Listing betrachten, finden wir viele alte Bekannte, so zuerst die Grundstruktur des AVR-Assemblerprogramms. Für das erste Programm benötigen wir in der Interrupt-Vektortabelle nur den Power-on-Reset. Die folgende Start-Sequenz für den AVR-Controller ist ebenfalls bereits bekannt.

```
;-----------------------------------------------------------------------
; Title         : Programmbeispiel für myAVR-Board mit LCD Add-On
;-----------------------------------------------------------------------
; Funktion      : gibt die Zeichenkette "Hallo myAVR" auf dem myAVR LCD aus
; Schaltung     : PORT D2 = RS, PORT D3 = E, PORT D4-7 = D4-7
;-----------------------------------------------------------------------
; Prozessor     : ATmega8
; Takt          : 3,6864 MHz
; Sprache       : Assembler
; Datum         : 03.02.2005
; Version       : 1.6
; Autor         : Dipl. Ing. Päd. Alexander Huwaldt
;-----------------------------------------------------------------------
.include    "AVR.H"
;-----------------------------------------------------------------------
; Reset and Interrupt vector            ; VNr.  Beschreibung
            rjmp    main                ; 1     POWER ON RESET
            reti                        ; 2     Int0-Interrupt
            reti                        ; 3     Int1-Interrupt
            reti                        ; 4     TC2 Compare Match
            reti                        ; 5     TC2 Overflow
            reti                        ; 6     TC1 Capture
            reti                        ; 7     TC1 Compare Match A
            reti                        ; 8     TC1 Compare Match B
            reti                        ; 9     TC1 Overflow
            reti                        ; 10    TC0 Overflow
            reti                        ; 11    SPI, STC Serial Transfer Complete
            reti                        ; 12    UART Rx Complete
            reti                        ; 13    UART Data Register Empty
            reti                        ; 14    UART Tx complete
            reti                        ; 15    ADC Conversion Complete
            reti                        ; 16    EEPROM Ready
            reti                        ; 17    Analog Comperator
            reti                        ; 18    TWI (I²C) Serial Interface
            reti                        ; 19    Store Program Memory Ready
;-----------------------------------------------------------------------
; Start, Power ON, Reset
main:       ldi r16,lo8(RAMEND)
            out SPL,r16                 ; Init Stackpointer LO
            ldi r16,hi8(RAMEND)
            out SPH,r16                 ; Init Stackpointer HI
            rcall   LCD_init
;-----------------------------------------------------------------------
mainloop:   rcall   wait
            ldi     r16, 'H'
            rcall   LCD_data
            ldi     r16, 'a'
            rcall   LCD_data
            ldi     r16, 'l'
            rcall   LCD_data
            ldi     r16, 'l'
            rcall   LCD_data
            ldi     r16, 'o'
            rcall   LCD_data
            ldi     r16, ' '
            rcall   LCD_data
            ldi     r16, 'm'
            rcall   LCD_data
            ldi     r16, 'y'
            rcall   LCD_data
            ldi     r16, 'A'
            rcall   LCD_data
            ldi     r16, 'V'
            rcall   LCD_data
            ldi     r16, 'R'
            rcall   LCD_data
            ldi     r16, '!'
            rcall   LCD_data
            ldi     r16, ' '
            rcall   LCD_data
            rjmp    mainloop
;=======================================================================
;hier Unterprogramme und Interruptroutinen zufügen
;-----------------------------------------------------------------------
wait:       push    r24
            ldi     r24,0x13                ; hier delay einstellen 13 = ca. 20yS
w1:         subi    r24,0x01
            rcall   wait20ms
            brcc    w1
            pop     r24
            ret
;-----------------------------------------------------------------------
```

```
wait5ms:    ldi     r16, 255
            ldi     r17,26
w5ms:       dec     r16
            brne    w5ms
            dec     r17
            brne    w5ms
            ret
;-----------------------------------------------------------------------
wait20ms:   ldi     r16, 255
            ldi     r17,104
w20ms:      dec     r16
            brne    w20ms
            dec     r17
            brne    w20ms
            ret
;-----------------------------------------------------------------------
LCD_init:   sbi     DDRD,2                  ;LCD RS = OUT
            sbi     DDRD,3                  ;LCD E  = OUT
            sbi     DDRD,4                  ;LCD D4 = OUT
            sbi     DDRD,5                  ;LCD D5 = OUT
            sbi     DDRD,6                  ;LCD D6 = OUT
            sbi     DDRD,7                  ;LCD D7 = OUT
            cbi     PORTD,2                 ;LDC RS = Low
            ;warte bis PowerUp
            ldi     r18,20
powerup:    rcall   wait5ms
            dec     r18
            brne    powerup                 ; Power-Up Wartezyklus min 30 ms
            ; sende init 1
            ldi     r16, 0b00100000         ; 4 Bit Modus aktivieren
            out     PORTD, r16
            rcall   LCD_enable              ; Enable-Impuls
            rcall   wait5ms
            ldi     r16, 0b00101000
            rcall   LCD_cmd                 ; Function Set 4 Bit, 2 Zeilen, 5x7
            rcall   LCD_off
            rcall   LCD_clear
            ldi     r16, 0x06
            rcall   LCD_cmd                 ; Entry Mode Set, increase+shifted
            rcall   LCD_on
            ret
;-----------------------------------------------------------------------
LCD_data:   ldi     r18,0b0000100           ;RS = hi
            rjmp    LCD_out
;-----------------------------------------------------------------------
LCD_cmd:    ldi     r18,0b0000000           ;RS = lo
LCD_out:    mov     r17, r16
            swap    r17
            andi    r16, 0b11110000
            or      r16,r18
            andi    r17, 0b11110000
            or      r17,r18
            out     PORTD, r16
            rcall   LCD_enable
            out     PORTD, r17
            rcall   LCD_enable
            rcall   wait5ms
            ret
;-----------------------------------------------------------------------
LCD_enable: sbi     PORTD, 3                ; Enable high
            nop                             ; kurz warten
            nop
            nop
            cbi     PORTD, 3                ; Enable wieder low
            ret
;-----------------------------------------------------------------------
LCD_clear:  ldi     r16, 0b00000001         ; Display löschen
            rcall   LCD_cmd
            rcall   wait5ms
            ret
;-----------------------------------------------------------------------
LCD_off:    ldi     r16, 0b00001000
            rcall   LCD_cmd
            rcall   wait5ms
            ret
;-----------------------------------------------------------------------
LCD_on:     ldi     r16, 0x0E
            rcall   LCD_cmd
            rcall   wait5ms
            ret
;-----------------------------------------------------------------------
```

Befehl	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Ausführungszeit	Erläuterungen
Display Clear	L	L	L	L	L	L	L	L	L	H	1,64 µs	
Return Home	L	L	L	L	L	L	L	L	H	X	1,64 µs	Cursor geht zur ersten Stelle, Adresszähler auf Adresse 0
Entry Mode Set	L	L	L	L	L	L	L	H	I/D	SH	42 µs	- **I/D: Cursor-Bewegungsrichtung** I/D = H: Adresspointer inkrementieren I/D = L: Adresspointer dekrementieren - **SH: Displayinhalt bei Schreiben in** Display-RAM (nicht) verschieben SH = H: Displayinhalt verschieben SH = L: Displayinhalt nicht verschieben
Display on/off	L	L	L	L	L	L	H	D	C	B	42 µs	- **D: Display ein/aus** D = H: Display ein D = L: Display aus - **C: Cursor ein/aus** C = H: Cursor ein C = L: Cursor aus - **B: Zeichen am Cursor blinkt** B = H: Blinken an B = L: Blinken aus
Shift	L	L	L	L	L	H	S/C	R/L	X	X	42 µs	**Cursor/Displayinhalt verschieben** S/C R/L L L Cursor eine Stelle nach links L H Cursor eine Stelle nach rechts H L Cursor/Displayinhalt nach links H H Cursor/Displayinhalt nach rechts
Set Function	L	L	L	L	H	DL	N	F	X	X	42 µs	**Display-Funktion definieren** - DL = H: 8-Bit-Interface DL = L: 4-Bit-Interface - N = H: 2-Zeilen-Display N = L:1-Zeilen-Display - F = H: Punktmatrix 5 x 10 F = L: Punktmatrix 5 x 7
Set CGRAM Adress	L	L	L	H	CGRAM-Adresse						42 µs	Zeichengenerator-RAM-Adresse setzen
Set DDRAM	L	L	H	DDRAM-Adresse							42 µs	Display-RAM-Adresse setzen
Busy Flag und Adresse setzen	L	H	BF	Adresszähler für CG- und DD-RAM							0 µs	- BF = H: Controller arbeitet Instruktion ab BF = L: Controller bereit
Write Data	H	L	Daten (schreiben)								46 µs	Daten in CG-/DD-RAM schreiben, Speicher je nach vorherigem Set-Befehl
Read Data	H	H	Daten (lesen)								46 µs	Daten aus CG/DD-RAM lesen Speicher je nach vorherigem Set-Befehl

Tabelle 1: Befehlstabelle des Display-Controllers

Bevor wir das Hauptprogramm betrachten, gehen wir etwas weiter hinunter und finden dort die bereits angesprochenen Warte-Routinen und die im vorherigen Kapitel ausführlich beschriebene Initialisierung des Displays.

Von „LCD_data“ bis „LCD_enable“ gibt es etwas Neues – die Steuer-Routinen für die Übertragung der Kommandos und Daten an den LCD-Controller.

Die Sprungmarken „LCD_data“ bzw. „LCD_cmd“ schalten die Leitung RS um, um so anzukündigen, welche Art von Daten nun folgen.

Unter „LCD-out“ erfolgt dann die Ausgabe an den LCD-Controller in der Reihenfolge:

- oberes Halbbyte separieren
- oberes Halbbyte und RS-Zustand an Port D bereitstellen
- Enable-Impuls an den LCD-Controller schicken
- unteres Halbbyte separieren
- unteres Halbbyte und RS-Zustand an Port D bereitstellen
- Enable-Impuls an den LCD-Controller schicken, dazu dient das Unterprogramm „LCD-enable“, das das Enable-Signal für kurze Zeit setzt
- Ausführungszeit des LCD-Controllers abwarten

In dieser Weise findet man alle Unterprogramme und Sprungmarken immer wieder als Standard für die LCD-Ansteuerung vor. Natürlich gibt es noch eine Anzahl weiterer Kommandos, so kann man den Cursor ein- und ausschalten, ihn blinken lassen, Zeichen an bestimmten Stellen im Display erscheinen lassen usw. All dies beschreibt das myAVR-Lernheft für das LCD-Add-on detailliert.

Wir haben uns hier allein auf die in unserem Programmbeispiel verwendeten Kommandos beschränkt.

Fehlt nur noch das Hauptprogramm! Und das gestaltet sich in diesem Falle recht einfach. Wir laden ein ASCII-Zeichen in ein Register und lassen es über die Sprungmarke „LCD_data“ und das Unterprogramm „LCD_out“ in zwei 4-Bit-Teilen an den LCD-Controller ausgeben. Da dessen Zeichensatz ebenfalls weitgehend dem ASCII-Zeichensatz entspricht, ist hier die Eingabe sehr einfach, man muss sich nicht mit Binärdaten mühen, die ASCII-Zeichen werden automatisch umgesetzt. Anders wäre dies bei selbst kreierten Zeichen – auch das ist möglich. Das Lernheft zeigt hier sehr interessante und ausführliche Beispiele, wie man es macht. So findet man hier u. a. auch ein Beispiel zur Umsetzung eines Analogwertes per ADC in eine adäquate LCD-Anzeige und Programmieranleitungen für C und BASCOM. Unseres Erachtens nach ist dieses Lernheft ohnehin eine der eingängigsten deutschen LCD-Programmieranleitungen auf dem Markt. Ist alles compiliert und auf den AVR gebrannt, sollte nun „Hallo myAVR“ im Display der auf das myAVR-Board gesteckten LCD-Add-on-Platine erscheinen.

Mit diesem einfachen Beispiel gelingt ideal der Einstieg in die LCD-Programmierung, man kann das Programm gleich als Spielwiese benutzen, längere Texte anzeigen lassen, beide Zeilen belegen, Texte an gewünschter Stelle im Display erscheinen lassen usw. Viel Spaß! ELV