LCDs und Microcontroller - (K)ein unbekanntes Land

FH Pforzheim
Hochschule für Gestaltung, Technik und Wirtschaft
FB2 Stg. Elektrotechnik
Microcontroller-Labor
Tiefenbronner Str. 65
75175 Pforzheim

Andreas Reber
e-mail: reber@fh-pforzheim.de
www: http//www.fh-pforzheim.de/fb2/eit/

1 Vorwort

Displays werden in der Welt der Elektronik immer wichtiger, weshalb die Programmierung von Display-Routinen im Microcontroller-Labor von Anfang an ein große Rolle spielt. Um nun nicht alles in der Laboranleitung beschreiben zu müssen, ist dieses kleine Handout entstanden, dass dem Programmierer bzw. Hardware-Freak einen kleinen Überblick über die Verwendung von Matrix-LCDs mit HD44780-Controllern gibt.

2 Beispiel-Übersicht

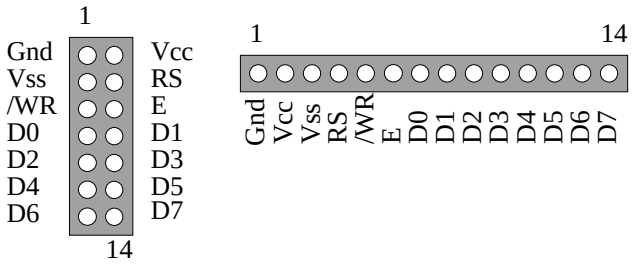
Beispiel-Applikationen können über die Labor-Homepage als PDFs geladen werden.

3 Der HD44780

Die am meisten verwendeten Punktmatrix-LC-Displays haben einen Controller, der auf dem HD44780 von Hitachi basiert. Der HD44780 ist in der Lage, für max. 80 Zeichen die Verwaltung der Daten- und Steuersignale zu übernehmen. Die Displaygröße variiert dabei von 1 x 8 Zeichen bis 4 x 20 Zeichen. Die physikalische Verbindung mit dem Microcontroller wird über einen 14-poligen Verbinder (16 Pole bei beleuchteten Displays) hergestellt, der ein oder zwei Pinreihen haben kann.

Pinbelegung :

Bezeichner	Pin	Pin	Bezeichner
Ground	1	2	Vcc
Vss	3	4	RS
/WR	5	6	E
D0	7	8	D1
D2	9	10	D3
D4	11	12	D5
D6	13	14	D7



Mit Pin 3 **Vss** wird der Kontrast der Displays eingestellt, dazu wird zwischen Vcc und Ground ein Trimmer mit 10K angeschlossen und der Abgriff mit **Vss** verbunden.

Pin 4 **RS** (**R**egister-**S**elect) wählt das Register im Controller aus, auf das geschrieben werden soll. Das **RS** muss vor **E** gesetzt werden.

In Systemen mit externem Adressbereich wird Pin 5 **/WR** mit **/WR** (Intel-System) oder mit **R/W** (Motorola) verbunden, andernfalls mit einem Portpin (z.B. bei PIC-Controllern). Auch **/WR** sollte vor der Aktivierung von **E** gesetzt werden.

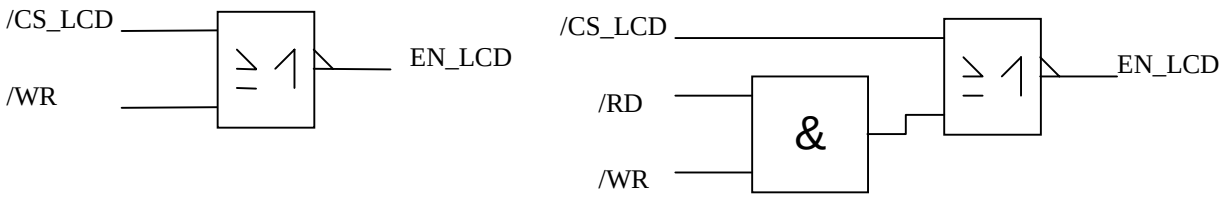
Dem Displaycontroller wird über Pin 6 **E** (**E**nable) mitgeteilt, daß die anliegenden Daten für ihn gültig sind. Dieses Signal ist high-aktiv. Der Zugriff beginnt mit der steigenden Flanke von **E**, Daten werden mit der fallenden Flanke gelesen oder geschrieben und die minimale Zykluslänge darf nicht unterschritten werden (normalerweise 500 ns, siehe Datenblatt HD44780). Zyklus bedeutet dabei die Zeit, die zwischen zwei steigenden Flanken von **E** liegt.

3.1 Systemeinbindung

Die Einbindung in das System erfolgt auf zwei verschiedenen Wegen, die abhängig vom Microcontroller sind. Applikationen sind unter www.fh-pforzheim.de/fb2/eit/mitarbeiter/reber/texte/fundgrube.htm zu finden.

3.1.1 LCD im Datenspeicherbereich

Die Integration in den Datenspeicherbereich des Microcontrollers ist hardwareseitig etwas aufwendiger, da neben einem Enable (high-aktives Chipselect) auch noch ein A0 und A1 für RS erzeugt werden muss. Es hat sich im Laufe der Zeit gezeigt, dass es einfacher ist, den **/WR** mit einer Adressleitung zu verbinden anstatt mit den Steuersignalen **/WR** bzw. **R/W**. Ist in einem System A0 und A1 schon gelatched vorhanden, verbindet man A0 mit RS und A1 mit /WR (man kann auch A8 und A9 nehmen, falls kein Latch vorhanden ist). Dann erhält man vier Adressen für das Display um lesend und schreibend zugreifen zu können. Spart man sich die Option lesen, kann /WR auch fest auf Gnd gelegt werden. Der Vorteil liegt jedoch in der einfacheren Programmierung, da man nur zwei oder vier Adresslabels für Daten- und Steuerregister definiert und die Daten dann mit einem normalen Ausgabebefehl auf das Display schreibt oder davon liest. Die folgende Prinzipschaltung erzeugt ein zeitgerechtes **E**-Signal für nur schreibende LCD-Zugriffe. Will man auch noch lesen, gilt die rechte Skizze



Das CS_LCD kann mittels eines 74xx138 erzeugt werden (siehe Schaltplan Schülerprojekt).

3.1.2 LCD an normalen Portpins

Wird das Display mit normalen Portpins verbunden (siehe PIC), ist die Programmierung etwas aufwendiger. In dem PIC-Beispiel wurde der komplette Port B für das Display verwendet. Wenn das Display im 4-Bit Mode betrieben wird, werden sieben Pins benötigt. Außerdem muss dann beachtet werden, dass ein Zugriff auf das Display immer in zwei Zyklen verläuft, um ein Byte zu übertragen. Dabei wird erst das höherwertigere Nibble auf D7 bis D4 ausgegeben, dann das niederwertigere. Soweit die Theorie, da in den letzten Jahren Displays aufgetaucht sind, die im 4-Bitmode nicht richtig funktionieren. Displays mit einem echten HD44780 machen keine Probleme, manche Nachbauten schon. Auffällig waren vor allem die KS0066U von Displaytech, die u.a. im Vertrieb von Reichelt sind.

3.2 Zeitliche Ansteuerung

Für beide Arten ergibt sich jedoch das Problem der zeitlichen Ansteuerung. Der Displaycontroller benötig unterschiedliche Ausführungszeiten für die einzelnen Befehle. Er besitzt zwar die Möglichkeit, in seinem Statusregister ein Busy-Flag zu setzten, doch kostet ein nicht gesteuertes Polling einfach zuviel CPU-Zeit. In den hier vorgestellten Anbindungen wurde deshalb konsequent auf Polling verzichtet und die Ansteuerung über Timer gewählt. Da im normalen Betrieb eigentlich nur eine Verzögerungszeit vorkommt, hält sich der Aufwand sehr in Grenzen, und man kann mit einem 8-Bit Timer die benötigte Zeitschleife realisieren. Im einfachen Fall wird der Timer nur gestartet und auf das Ende des Zählvorganges gewartet (eigentlich auch nur Polling). Hat man im System mehr zu tun, so installiert man entweder einen Systemtimer, der sich in bestimmten Zeitabständen mit einem Interrupt meldet oder man muss das Programm mit geschickter Schleifenprogrammierung um die Displayausgabe herum programmieren. Im Allgemeinen verwendet man, wenn nötig, lieber den Systemtimer, da dort die einzelnen Routinen besser zu erstellen und universeller änder- und einsetzbar sind.

3.2.1 Ausgabe mit Timer

Systemtimer

Die Displayausgabe erfolgt dann nach dem Notwendigkeitsprinzip, d.h. es wird ein allgemeines Flagr egister definiert, welches ein Flag für die Displayausgabe enthält. In der Systemtimer-Interruptroutine wird dieses Flag geprüft. Ist es gesetzt, so wird in die Ausgaberoutine für ein Zeichen/Befehl verzweigt. Falls das Flag nicht gesetzt ist, wird die Interruptroutine verlassen oder die anderen Flags bearbeitet. Die Ausgaberoutine muss nur am Ende der Ausgabe für ein Rücksetzen des Flags sorgen. Die Ausgabe selber wird von einem Unterprogramm übernommen, dass nachfolgend beschrieben wird.

Timerinterrupt

Eine zweite Möglichkeit beruht darauf, einen Timerinterrupt freizugeben, der die Ausgabe aufruft. Gesperrt wird der Timerinterrupt durch das Erkennen der 0h am Stringende. Der String-Pointer wird in einer Variablen abgelegt, die dann die eigentliche Ausgaberoutine als Pointer verwendet, da er ja nach jedem Zeichen erhöht werden muss. Den Timer auf 40 µs für die Ausgabe zu programmieren, macht keinen Sinn, da der Verwaltungsaufwand für die Routine zu groß ist. Für den Timer eine Zeit von 200 bis 300 µs sinnvoll, da die Verzögerung bei der Ausgabe nicht auffällt und der Verwaltungsaufwand für den Interrupt in einem akzeptablen Verhältnis steht. Der Interrupt selber sichert zwar keine Register, doch muss die Ausgabe Akku, aktuellen Datenpointer sichern, Variable des String-Pointers laden, Pointer-Inhalt holen und auf 0h prüfen (nicht mit `cjne`, da sonst auch noch das **PSW** gesichert werden muss), Variable erhöhen und sichern, Display-Pointer laden und Zeichen ausgeben. Danach wieder gesicherten Akku und Datenpointer zurückholen, ISR beenden (Code siehe 4.2). Die Ausgabefunktion ist auch in Systemtimer-Version verwendbar, nur wird hier das Flag zur Ausgabe gelöscht und nicht der Interrupt.

4.0 Befehlsatz des HD44780

Die folgenden Verzögerungswerte entsprechen nicht immer den Werten aus den jeweiligen Datenblättern, haben aber den Vorteil, universell für die HD44780-kompatiblen Displaycontroller benutzt werden zu können.

Steuerbefehle des HD44780

Aktion	D7	D6	D5	D4	D3	D2	D1	D0	V-Zeit	siehe
Display löschen	0	0	0	0	0	0	0	1	1.64 ms	4.0.1
Cursor in Startposition	0	0	0	0	0	0	1	-	1.64 ms	4.0.2
Betriebsart	0	0	0	0	0	1	I/D	S	40 us	4.0.3
Display/Cursor Ein/Aus	0	0	0	0	1	D	C	B	40 us	4.0.4
Display/Cursor bewegen	0	0	0	1	S/C	R/L	-	-	40 us	4.0.5
Schnittstellenbreite	0	0	1	DL	N	F	-	-	4.1 ms	4.0.6
CGRAM Adresse	0	1	ACG	ACG	ACG	ACG	ACG	ACG	40 us	4.0.7
DDRAM Adresse	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	40 us	4.0.8

4.0.1 Display löschen 00000001 (Clear Display)

Der Befehl ist mit einem Reset Display zu vergleichen, da der Ziffernspeicher mit Leerzeichen aufgefüllt und die Anzeige somit gelöscht wird.

4.0.2 Cursor in Position 1 0000001- (Return Home)

Der Cursor wird an die erste Position im Display gesetzt, ohne jedoch den Inhalt zu verändern. Bei mehrzeiligen Displays ist dies die linke obere Ecke. Bit D0 ist nicht relevant.

4.0.3 Betriebsart 0000 01 I/D S (Entry Mode Set)

Mit diesem Befehl wird dem Display mitgeteilt, wie es sich bei der Zeichenausgabe verhalten soll. Für die Bits I/D und S gilt dann folgendes :

I/D	S	Funktion
1	1	Schreibcursor wandert um eine Stelle nach rechts, Display wird nach rechts versch oben
1	0	Schreibcursor wandert um eine Stelle nach rechts, Display wird nicht verschoben
0	1	Schreibcursor wandert um eine Stelle nach links, Display wird nach links versch oben
0	0	Schreibcursor wandert um eine Stelle nach links, Display wird nicht verschoben

4.0.4 Display/Cursor Ein/Aus 0000 1 D C B (Display on/off Control)

Dieser Befehl steuert Aussehen und Verhalten von Anzeige und Cursor. D ist für die Anzeige, C für den Cursor und B für das Blinken verantwortlich.

D	C	B	Funktion
1	1	1	Anzeige Ein, Cursor Ein, Blinken Ein
1	1	0	Anzeige Ein, Cursor Ein , Blinken Aus
1	0	1	Anzeige Ein , Cursor Aus, Blinken Ein
1	0	0	Anzeige Ein , Cursor Aus, Blinken Aus
0	x	x	Anzeige Aus (Daten bleiben erhalten), Cursor- und Blinkeinstellung irrelevant

4.0.5 Display/Cursor bewegen 0001 S/C R/L - - (Cursor or Display Shift)

Dieser Befehl ermöglicht die Bewegungssteuerung der Anzeige und des Cursors, z.B. für Laufschriften. Die Bits D1 und D0 werden nicht berücksichtig.

S/C	R/L	Funktion
1	1	Anzeige nach rechts verschieben, Cursor folgt
1	0	Anzeige nach links verschieben, Cursor folgt
0	1	Cursor nach rechts verschieben
0	0	Cursor nach links verschieben

4.0.6 Schnittstellenbreite 001 DL N F – (Function Set)

Dieser Befehl wird nur in der Initialisierungsphase verwendet. Er stellt das verwendete Businterface zwischen Microcontroller und Display ein. Er muss als erster Befehl bei der Initialisierung auf das Display durchgeführt werden.

DL	N	F	Funktion
1	1	1	8 Bit Interface, 2 Zeilen, Font 5 x 10 Dots
1	1	0	8 Bit Interface, 2 Zeilen, Font 5 x 8 Dots
1	0	1	8 Bit Interface, 1 Zeile, Font 5 x 10 Dots
1	0	0	8 Bit Interface, 1 Zeile, Font 5 x 8 Dots
0	1	1	4 Bit Interface, 2 Zeilen, Font 5 x 10 Dots
0	1	0	4 Bit Interface, 2 Zeilen, Font 5 x 8 Dots
0	0	1	4 Bit Interface, 1 Zeile, Font 5 x 10 Dots
0	0	0	4 Bit Interface, 1 Zeile, Font 5 x 8 Dots

4.0.7 CGRAM Adresse 01 ACG ACG ACG ACG ACG ACG (Set CGRAM address)

Dieser Befehl ermöglicht den Zugriff auf die Daten des Character Generator RAMs.

4.0.8 DDRAM Adresse 1 ADD ADD ADD ADD ADD ADD ADD (Set DDRAM address)

Mit den Bits D6 bis D0 kann der Cursor an eine beliebige Adresse im Display gesetzt werden, an der dann das nächste Zeichen ausgegeben wird. Zu beachten ist hierbei, dass der Cursor im sichtbaren Bereich positioniert werden muss, um etwas zu sehen.

4.1 Adressverteilung innerhalb des Displays

Adressverteilung bedeutet dabei, welche Adresse im DDRAM angesprochen werden muss, wenn der Cursor an eine bestimmte Stelle im Display gesetzt werden soll. Der HD44780 kann max. 80 Zeichen darstellen, die im Displayspeicher von 00h bis 4Fh liegen. Adressiert werden sie über den Adresszähler, der nach einem Zugriff automatisch de- oder inkrementiert wird (je nach Einstellung). Das Setzen des Adresszählers erfolgt mit dem Befehl *DDRAM Adresse*. Die erste Displayposition hat die DDRAM Adresse 00h, da der Befehl jedoch ein MSB mit dem Wert 1 hat, muss man zu jeder Displayposition den Offset 7Fh addieren um den Datenwert zu erhalten.

Im Laufe der Zeit sind Displays aufgetaucht, die eine andere Adressverteilung haben als man denkt. So wird von Reichelt ein 1 x 16 Zeichen Display vertrieben, welches intern als 2 x 8 aufgebaut ist, was dazu führt, dass nach dem achten Zeichen der Cursor auf C0 gesetzt werden muss, damit das neunte Zeichen dann auch wirklich zu sehen ist. Verhält sich eine Ausgabe nicht so wie man es sich erhofft, so kann man in einer Schleife einfach mal das ganze Display fortlaufend mit dem kompletten Alphabet beschreiben, so erkennt man Cursor-Fehler am besten.

Adressverteilung für ein Display mit 4 x 20 Zeichen

Spalte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Zeile 1 (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
Wert (h)	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
Zeile 2 (h)	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
Wert (h)	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
Zeile 3 (h)	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
Wert (h)	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F	A0	A1	A2	A3	A4	A5	A6	A7
Zeile 4 (h)	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67
Wert (h)	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5	E6	E7

Mit obiger Verteilung sind alle Displayvarianten abgedeckt, d.h. wenn man den Positionswert der vierten Spalte, zweite Zeile für ein Display mit 2 x 20 Zeichen sucht, erhält man C3h. Dieser Wert gilt auch für ein Display mit nur 2 x 8 Zeichen. Zeile drei und vier die Verlängerung von Zeile eins und zwei, wie man an den Adressen erkennen kann, so dass auch Displays mit mehr als 20 Zeichen pro Zeile abgedeckt sind. Displays mit vier Zeilen kann man sich als zweizeilige Displays vorstellen, die nach der Spalte 20 abgetrennt und der Rest dann unten angefügt wurde.

4.2 Initialisierung

Die Grundinitialisierung verläuft immer nach dem selben Schema.

- Schnittstellenbreite einstellen
- Anzeige löschen
- Display/Cursor Ein/AUS
- Betriebsart

Der Befehl *Betriebsart* ist optional, falls man die Defaulteinstellung nach *Display löschen* (Cursor geht nach rechts, Cursor in Position eins) nicht übernehmen will, kann man dort den aktuellen Mode einstellen.

Code für den 805x / ADuC812:

```
; ************************************************************************
; *      UP LCD-Initialisierung                                          *
; *                                                                      *
; *      0x38h: Interface mit 8 Bit, kleiner Font, 2 Zeilen              *
; *                                                                      *
; ************************************************************************
LCD_INI: mov  DPTR,#pLCD_Inst        ; 2 x 38H einschreiben
         mov  A,#038H                ; 8 Bit, 5x7 Dots, 2 Lines
         movx @dptr,A
         CALL Delay_L                ; 4.1 ms
         movx @dptr,A
         CALL Delay_L                ; 4.1 ms

         mov  A,#01H                 ; Ini 01H ->Clear Display
         movx @dptr,A
         CALL Delay_L                ; 4.1 ms

         mov  A,#0CH                 ; Ini 0CH ->Display on, Cursor off, Blinking off
         movx @dptr,A
         CALL Delay_K                ; 40 us

         ret                         ; Ende Initialisierung
```

Die beiden Zeitschleifen Delay_K und Delay_L erzeugen die benötigte Wartezeit.

Für den Normalbetrieb kann Delay_L dann mit 1.6 ms programmiert werden, da dort die 4.1 ms nicht mehr nötig sind. Die 1.6 ms werden dann für den Befehl *Cursor in Startposition* (Cursor Home) notwendig, doch kann die gleiche Funktion über *DDRAM Adresse* erreicht werden, die nur 40 µs benötigt.

4.3 Beispielcode aus 3.2.1

```
; ************************************************************************
; *     UP LCD-Ausgabe
; *        Wird vom Timer0 aufgerufen
; *        Sperrt Timer0 am Ende der Ausgabe
; *        Benötigt die Variablen pAusgabeH und pAusgabeL
; ************************************************************************
LCD_OUT:    clr  TR0                    ; Timer0 stoppen
            push ACC
            push DPH
            push DPL

            mov  DPH,pAusgabeH          ; Variable laden
            mov  DPL,pAusgabeL
            clr  A
            movc A,@A+DPTR
            jz   _S_ENDE                ; Stringende erreicht ?
            inc  DPTR
            mov  pAusgabeH,DPH          ; Variable sichern
            mov  pAusgabeL,DPL
            mov  DPTR,#pLCD_Data        ; LCD-Pointer laden
            movx @DPTR,A
            pop  DPL
            pop  DPH
            pop  ACC
            setb TR0                    ; Timer0 starten
            ret
_S_ENDE:    clr  ET0                    ; Interrupt sperren
            pop  DPL
            pop  DPH
            pop  ACC
            setb TR0                    ; Timer freigeben
            ret
```