

Dokumentation der LCD-Funktionen

by Matthias Bär

1. Kurzbeschreibung
2. Funktionen
 - a. Funktionsumfang der Include-Files
 - b. Benutzung der Funktionen
3. Besonderheiten
4. Unterstützte Controller
5. Codebeispiele
6. Was ich schon immer sagen wollte

1. Kurzbeschreibung

Ziel des Projektes war, einfache Funktionen zur Ansteuerung von Character-LCDs (HD44780 oder kompatible) zu erstellen, die von jedem benutzt werden können. Das Display wird im 4-bit Modus betrieben, was I/O Leitungen spart. Der gesamte Code ist in ASM geschrieben und ist ausreichend kommentiert. Somit sollte es auch Anfängern möglich sein die Funktionen zu verstehen und einzusetzen. Es wurde außerdem darauf geachtet, dass so wenig Ressourcen wie möglich verwendet werden. So werden für die Funktionen nur der Z-Pointer (nicht alle Funktionen), R0 (für Funktionen die LPM verwenden), der Stack sowie ein Register (lcd_temp) verwendet.

Wenn alle Include-Files eingebunden werden werden 300 words benötigt.

Wenn nur die lcd_basics.asm eingebunden wird werden 140 words benötigt.

Hinweis: Diese Doku erspart wahrscheinlich nicht den Blick ins Datenblatt des HD44780

2. Funktionen

Die Funktionen sind insgesamt in 6 Include-Files verteilt, die je nach bedarf eingebunden werden können.

2.a Liste der Include-Files

Include-File	Funktion	Beschreibung	Beeinflusste Register
lcd_definitions.asm	-	enthält alle wichtigen Definitionen	-
lcd_basics.asm	lcd_init	Ist die erste Funktion die aufgerufen werden muss um das Display benutzen zu können	lcd_temp ZH ZL Stack
	lcd_write_command	Sendet einen Befehl an das Display (siehe LCD-Befehle weiter unten)	lcd_temp Stack
	lcd_write_data	Sendet Daten an das Display	lcd_temp Stack
lcd_write_cgram.asm lcd_char.asm	lcd_write_cgram	Schreibt Sonderzeichen in den cgram des Displays, welche in lcd_char.asm definiert sind	lcd_temp ZH ZL Stack
lcd_shift.asm	lcd_shift_display	Schiebt das Display nach links/rechts	lcd_temp Stack
	lcd_shift_cursor	Schiebt den Cursor nach links/rechts	lcd_temp Stack
lcd_puts.asm	lcd_puts	Gibt einen String aus dem Programmspeicher an der aktuellen Curserposition aus	lcd_temp ZH ZL Stack
lcd_get_ac.asm	lcd_get_ac	Liest die aktuelle Curserposition aus	lcd_temp Stack



Bei diesen Funktionen haben die Register nach Rückkehr von der Funktion veränderte Werte

2.b Benutzung der Funktionen

Grundsätzlich werden allen Funktionen der Parameter über das Register `lcd_temp` übergeben. Rückgabewerte (nur `lcd_get_ac`) werden ebenfalls im Register `lcd_temp` übergeben. Wenn ein Pointer übergeben werden muss (nur `lcd_puts`) wird dieser vor Funktionsaufruf in den Z-Pointer geladen.

Funktion	Beschreibung
<code>lcd_init</code>	Diese Funktion muss aufgerufen werden bevor man das Display benutzen kann. => ganz am Anfang in der init-Phase des Programms
<code>lcd_write_command</code>	Diese Funktion sendet einen Befehl an das Display. Mögliche Befehle sind: <ul style="list-style-type: none">- <code>CLEAR_LCD</code> (löscht Display und setzt Cursor auf die Anfangsposition)- <code>CURSOR_HOME</code> (setzt den Cursor auf die Anfangsposition; Displayinhalt bleibt erhalten)- <code>SET_DDGRAM</code> (setzt Cursor auf die Anfangsposition)- <code>SET_CGRAM</code> (setzt Adresse auf den Anfang des CGRAMs)- <code>SET_DDGRAM + n</code> (setzt Cursor auf die Anfangsposition + n)- <code>SET_CGRAM + n</code> (setzt Adresse auf den Anfang des CGRAMs + n)
<code>lcd_write_data</code>	Diese Funktion schreibt ein Datenbyte zum Display
<code>lcd_write_cgram</code>	Diese Funktion muss einfach nur aufgerufen werden... Außerdem wird der Cursor an den Anfang der 1. Zeile gesetzt
<code>lcd_shift_display</code>	Diese Funktion schiebt den Inhalt des Display nach links/rechts Dabei wird der übergebene Parameter als Vorzeichenbehaftet interpretiert. Parameter positiv => Inhalt wird nach rechts geschoben Parameter negativ => Inhalt wird nach links geschoben
<code>lcd_shift_cursor</code>	Diese Funktion schiebt den Cursor nach links/rechts Dabei wird der übergebene Parameter als Vorzeichenbehaftet interpretiert. Parameter positiv => Cursor wird nach rechts geschoben Parameter negativ => Cursor wird nach links geschoben
<code>lcd_puts</code>	Gibt einen String an der aktuellen Cursorposition aus. Der String muss mit <code>,end_of_string'</code> terminiert werden (siehe <code>lcd_definitions.asm</code>) Der Anfang des Strings muss über den Z-Pointer übergeben werden. Kein Auto-linefeed
<code>lcd_get_ac</code>	Liest die aktuelle Adresse (Cursorposition) aus. Rückgabewert liegt in <code>lcd_temp</code>

3. Besonderheiten

Da während den LCD-Funktionen das Register `lcd_temp`, der Z-Pointer, `r0`, sowie der Stack verändert werden können ist es „verboten“ dort Variablen für Interrupt-Routinen zu speichern.

Weiterhin sollten die Funktionen nicht in Interrupt-Routinen verwendet werden, da sie den Controller einige μs - ms aufhalten können

Möchte man dennoch LCD-Funktionen während des normalen Programmablaufs und während Interrupt-Routinen verwenden, so müssen vor Aufruf von LCD-Funktionen im normalen Programmcode Interrupts deaktiviert werden. (naja is eig klar - damit die eine LCD-Funktion der anderen net reinpfuscht...)

4. Unterstützte Controller

Es werden alle Controller unterstützt bis auf:

- AT90S1200
- ATtiny10
- ATtiny11
- ATtiny12
- ATtiny15
- ATtiny28

5. Codebeispiele

Einfache Ausgabe

```
.device ATMEGA8
.include "m8def.inc"

.include "lcd_definitions.asm"

.CSEG
.ORG 0x00
    rjmp main

.include "lcd_basics.asm"

main:
    ldi r16, high(ramend) ;stackpointer initialisieren
    out sph, r16
    ldi r16, low(ramend)
    out spl, r16

    rcall lcd_init ;initialisierung vom Display

    ldi lcd_temp, 'a'
    rcall lcd_write_data

loop:
    rjmp loop
```

Benutzung des CGRAMs

```
.device ATMEGA8
.include "m8def.inc"

.include "lcd_definitions.asm"

.CSEG
.ORG 0x00
    rjmp main

.include "lcd_basics.asm"
.include "lcd_write_cgram.asm"

main:
    ldi r16, high(ramend) ;stackpointer initialisieren
    out sph, r16
    ldi r16, low(ramend)
    out spl, r16

    rcall lcd_init ;initialisierung vom Display
    rcall lcd_write_cgram

    ldi lcd_temp, 0 ;gibt das erste definierte sonderzeichen aus
    rcall lcd_write_data

loop:
    rjmp loop

.include "lcd_char.asm"
```

Benutzung von lcd_puts

```
.device ATMEGA8
.include "m8def.inc"

.include "lcd_definitions.asm"

.CSEG
.ORG 0x00
    rjmp main

.include "lcd_basics.asm"
.include "lcd_puts.asm"

main:
    ldi r16, high(ramend) ;stackpointer initialisieren
    out sph, r16
    ldi r16, low(ramend)
    out spl, r16

    rcall lcd_init ;initialisierung vom Display

    ldi ZH,high(string*2) ;pointer setzen
    ldi ZL,low(string*2)
    rcall lcd_puts ;string ausgeben

loop:
    rjmp loop

string:
    .db "hallo",end_of_string
```

Benutzung von lcd_puts und lcd_shift_display

```
.device ATMEGA8
.include "m8def.inc"

.include "lcd_definitions.asm"

.CSEG
.ORG 0x00
    rjmp main

.include "lcd_basics.asm"
.include "lcd_puts.asm"
.include "lcd_shift.asm"

main:
    ldi r16, high(ramend) ;stackpointer initialisieren
    out sph, r16
    ldi r16, low(ramend)
    out spl, r16

    rcall lcd_init ;initialisierung vom Display

    ldi ZH,high(string*2) ;pointer setzen
    ldi ZL,low(string*2)
    rcall lcd_puts ;string ausgeben

    ldi lcd_temp, -2
    rcall lcd_shift_display ;jetzt ist nur noch 'llo' zu lesen

loop:
    rjmp loop

string:
    .db "hallo",end_of_string
```

6. Was ich schon immer sagen wollte

ASM macht reich und sexy.

Wer in dieser Doku einen Fehler findet darf ihn behalten.

Sie dürfen diese Funktionen frei verwenden.

Bei Fragen und Anregungen können Sie mir jederzeit eine E-Mail an matthias.m.baer@web.de schreiben.

Wenn diese Funktionen häufig benutzt werden, folgen evtl. weitere Projekte.
Geplant sind Routinen zur Taster-Abfrage und ein Software-timer (um Hardware-timer zu sparen)