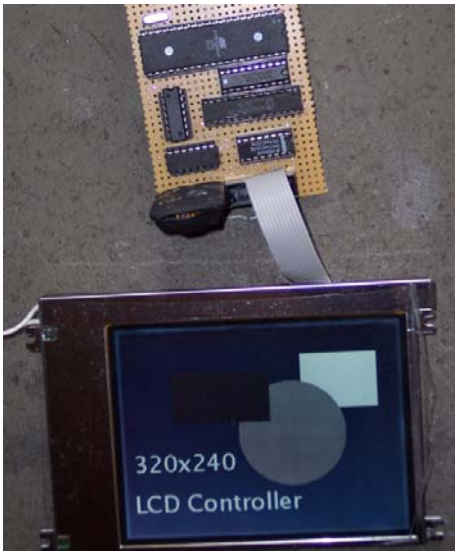
320x240 LCD Controller



- LCD Controller für ein 320x240 4bit LCD (oder auch andere Auflösungen bis max. 1024x252)

- Schnittstelle: UART

- 8x8 oder 8x12 Zeichensatz (256 Zeichen)

- Grafik möglich (Pixel, Bilder, Linien, Rechtecke, Kreise)

- 2, 4, 8 Graustufen (in der Software einstellbar)

- 1024x240 (bei 1bpp), 512x240 (bei 2bpp) oder 336x240 (bei 3bpp) virtuelle Auflösung (bei 32kByte SRAM)

Befehl	Parameter	Beschreibung
0		NOP (nichts machen)
1		Cursor auf (0,0) setzen
2		LCD (Kontrastspannung) aus
3		LCD (Kontrastspannung) an
8		Backspace
9	X Low X High Y Farbe	Set Pixel (x,y,Farbe)
10		Cursor eine Zeile nach unten
11		Zeichen an Cursorposition löschen
12		Komplettes Bild löschen
13		Cursor an Zeilenanfang setzen
14		Nicht verwenden!
15	Kontrast	Kontrast einstellen (0-255)
16	170 X Y XS YS Modus Daten…	Bild an Position (x,y) mit Größe (xs,ys) laden. Auf diesen Befehl folgen xs*ys Bytes mit Bilddaten. x und xs zählen Bytes (ein Byte ist bei 1bpp daher 8 Pixel, bei 2bpp 4 Pixel). x hat daher den Bereich 0-127, xs den Bereich 1-128. y und ys zählen Zeilen. y hat daher den Bereich 0-239, ys den Bereich 1-240. Im 3bpp Modus werden 3 Byte Blöcke gezählt. Es müssen daher Vielfache von 3 Bytes (=8 Pixel) gesendet werden. Der Bereich von x liegt bei 0-39, von xs bei 1-40. Modus schaltet zwischen 1bpp (0), 2bpp (1), und 3bpp (2) um (in der 1bpp Version wird der Wert ignoriert, muss aber gesendet werden). Der Wert 170 (0xAA) dient nur als Sicherheitsbyte, damit nicht versehentlich dieser Befehl (z.B. aufgrund eines Übertragungsfehlers) ausgeführt wird.
17	X (Low) (X High) Y	Textcursor auf (x,y) setzen. X zählt in 8 Pixel Schritten, Y in Zeilenschritten Im pixelgenauen Textmodus zählt X 1 Pixel Schritte, daher ist dann (und auch nur dann) noch ein Highbyte erforderlich
18	X1 Low X1 High Y1 X2 Low X2 High Y2 Farbe	Linie von (x1,y1) nach (x2,y2) mit Farbe zeichnen
19	X1 Low X1 High Y1 R Farbe 1 Farbe 2	Kreis mit Mittelpunkt (x,y) und Radius R zeichnen. Farbe 1 ist die Füllfarbe. Farbe 2 ist die Rahmenfarbe. Hinweis: Der Radius darf nicht größer sein als x oder y, ebenso darf er nicht 0 sein, ansonsten wird der Befehl ignoriert.
20	Schriftart	Wählt die aktive Schriftart aus (mit externem Flash) Wählt den Zoomfaktor der Schriftart aus (mit USE_FONTZOOM)
21	Textfarbe Hintergrundfarbe	Setz Textfarbe und Hintergrundfarbe

22		Nicht verwenden!
23	Speicherseite	Schreibseite einstellen (bei >64kByte SRAM)
24	Speicherseite	Anzeigeseite einstellen (bei >64kByte SRAM)
25	Nummer Zeile 1 Zeile 2 … Zeile 8	Benutzerdefiniertes Zeichen an die entsprechende Position laden. Jedes Zeichen besteht aus 8 Zeilen (Bytes) zu je 8 Pixeln (Bits). Insgesamt sind 16 Zeichen verfügbar (Nummer 0-15)
26	SX SY DX DY XS YS (SPAGE) (DPAGE)	Kopiert einen Bildausschnitt mit XS*YS Bytes von (SX,SY) nach (DX,DY). X zählt dabei jeweils 8 Pixel Schritte, Y Zeilen. Bei >64kByte SRAM (und auch nur dann) werden zusätzlich zu den Koordinaten noch die Speicherseiten angegeben. Damit lassen sich die Bilddaten nicht nur innerhalb eines Bildes, sondern auch zwischen Speicherseiten kopieren.
27	X	Setzt den X Offset für den Textcursor (für das Beschreiben des virtuellen Bildes). Wird im 8 Graustufenmodus ignoriert.
28	X	Setzt die linke Startposition des virtuellen Bildes in 8 Pixel Schritten. Der Wertebereich liegt zwischen 0-88 bei 1bpp und 0-24 bei 2bpp.
29	X1 Low X1 High Y1 X2 Low X2 High Y2 Farbe 1 Farbe 2	Rechteck mit Eckkoordinaten (x1,y1, x2,y2) zeichnen. Farbe 1 ist die Füllfarbe. Farbe 2 ist die Rahmenfarbe.
30	Ziel Wert	Zeichen aus erweitertem Zeichensatz schreiben (nur für ASCII Codes <32 notwendig). Ziel = 0: Benutzerdefiniertes Zeichen 0-15 schreiben Ziel > 0: ASCII Zeichen 0-255 schreiben.
32-255		Buchstaben an Cursorposition zeichnen, Cursor erhöhen.

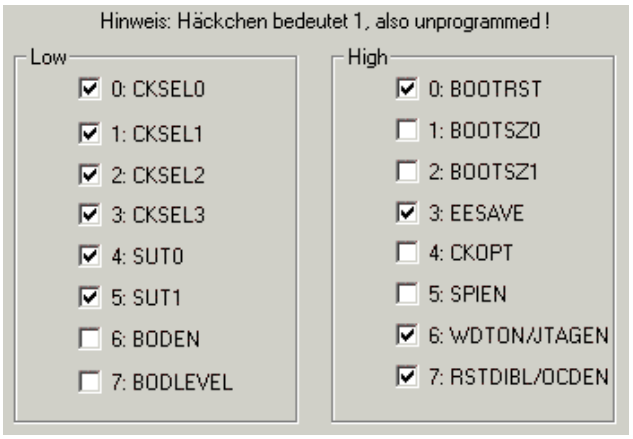
Ein paar Beispiele:

A B C (oder 65, 66, 67 als Binärwert)	schreibt die Zeichen A, B, C an die aktuelle Cursorposition
29 140 0 100 179 0 139 0 255	zeichnet ein 40x40 großes, schwarzes Rechteck, mit einem weißen Rand
19 160 0 120 15 64 128	zeichnet einen Kreis mit dem Mittelpunkt 160,120 und dem Radius 15, gefüllt mit der dunkelsten Graustufe und einem etwas helleren Rand
16 170 2 16 1 8 0 1 2 4 8 16 32 64 128	zeichnet ein 8x8 Pixel großes Bild mit den Daten 1, 2, 4, 8, 16, 32, 64, 128 (=schräge Linie) an die Position 16, 16
16 170 4 16 2 8 1 0 3 0 12 0 48 0 192 3 0 12 0 48 0 192 0	zeichnet ein 8x8 Pixel großes Bild (schräge Linie) an die Position 16, 16, also genauso wie vorher, nur im Graustufenmodus

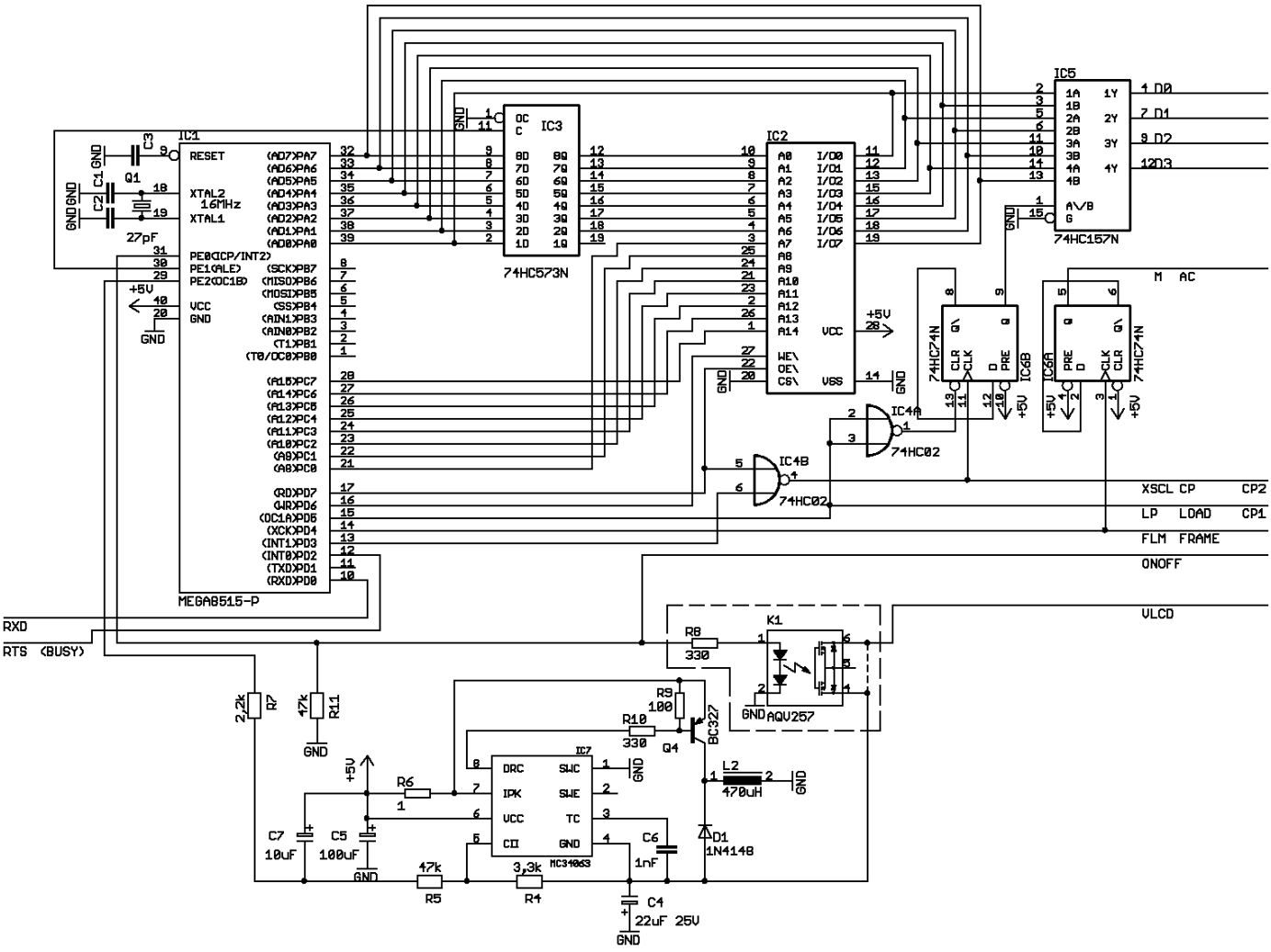
Hinweise zur Ansteuerung:

In der Beschreibung wird von einem 320x240 Display und 32kByte SRAM ausgegangen. Bei anderen Auflösungen oder Speichergrößen weichen die Bereiche der zulässigen Werte entsprechend ab.
Aufgrund der teilweise ziemlich rechenintensiven Grafikbefehle sollte der RTS/Busy Ausgang vor dem Senden eines Befehls abgefragt werden. Der Controller verfügt zwar über einen 256 Byte Befehlspuffer, aber auch dieser ist irgendwann voll, wenn mehrere aufwendige Befehle ohne Pause gesendet werden! Solange der RTS/Busy Pin Low ist, kann man gefahrlos mindestens 64 Bytes senden, da RTS/Busy aktiviert wird, wenn der Puffer zu ¾ gefüllt ist. Man braucht also nicht vor jedem Byte RTS/Busy zu prüfen, sondern es reicht einmal vor jedem Befehl (falls dieser kürzer als 64 Bytes ist.)

Die Fusebits müssen auf externen Quarz gestellt werden. Und da die Frequenz >8MHz ist, CKOPT nicht vergessen zu setzen! Um einen sicheren Betrieb zu gewährleisten sollte man auch die BOD auf 4V einschalten. Dann schaltet der AVR nämlich ab wenn die Spannung zu klein wird, was wiederum die Displayspannung über den Optomosfet abschaltet. Das ganze sieht dann in etwa so aus:



Schaltung:



Der OptoMOSFET K1 kann bei LCDs die über einen ONOFF oder DispOFF\ Pin verfügen entfallen. Falls der OptoMOSFET nicht erhältich ist, kann dieser durch einen normalen Optokoppler ersetzt werden, wenn dieser genügend Strom für das LCD schalten kann (Stromverstärkung/Übertragungsverhältnis beachten!), oder auch durch eine Schaltung aus ein paar Transistoren.
R5/R4 stellen die Spannung für das Display und somit den Kontrast ein. Der Kontrast lässt sich über einen kleinen Bereich per Software über PWM regeln. Sollte dieser Bereich für das LCD nicht passen, müssen R5/R4 angepasst werden.

IC4B generiert aus dem RD\ Impuls und dem Enable Signal (PD3) die Shiftclock Impulse für das LCD. Damit wird das Flipflop IC6B in jedem Takt umgeschaltet um so abwechselnd D0-3 und D4-7 mittels IC5 an den Ausgang zu legen. Dieser dient auch gleichzeitig als Leitungstreiber für den 4bit Datenbus. Damit IC6B zum Beginn jeder Zeile auf dem gleichen Zustand ist, wird IC6B bei jedem Latchpuls (=Zeilenende/anfang) über IC4A resettet. IC6A toggelt den M Ausgang bei jedem Bildbeginn, um die Wechselspannung im LCD zu erzeugen.

Ein paar Worte zum SRAM Timing:

Das Timing hängt hauptsächlich von der "OE\ Access Time" des SRAMs ab:
Im Datenblatt vom ATmega8515, Seite 202, Tabelle 98, Punkt 10:
Read Low to Data valid: max 1.0*tclcl-50ns. Bei 16MHz ist tclcl=62,5ns.
Das SRAM darf daher maximal 12,5ns brauchen zwischen dem Anlegen des RD Impulses bis die Daten stabil sein müssen.
Das von mir gerne verwendete IS61C256 Cache SRAM hat in der langsamsten 25ns Ausführung hier nur 9ns. Es ist also ausreichend schnell.
Bei der Low Power Variante IS62C256 dagegen, hat selbst die 45ns Variante hier 25ns. Dies kann funktionieren, da die Werte jeweils die garantierten Maximalwerte sind, muss aber nicht.

Zusätzlich ist natürlich noch die Zugriffszeit wichtig:
Das wäre Punkt 5 im Datenblatt: Address valid to RD Low: 1.0*tctc-10ns.
Insgesamt ergibt sich dadurch eine Zeit von 2.0*tctc-60ns, also 65ns zwischen Adresse gültig bis Daten stabil, was eigentlich alle SRAMs die obige Bedingung erfüllen, auch erfüllen.

Mit viel Glück funktionieren aber selbst 100ns SRAMs ohne waitstates. Trotzdem sollte man möglichst SRAMs mit 55ns oder weniger verwenden.

Neben dem SRAM sollte man auch das Timing des Displays beachten. Einigermaßen aktuelle Displays haben mit dem Timing keine Probleme, aber einige ältere benötigen den waitstate.

Das Latch ist meiner Meinung nach unkritisch, da die Zugriffszeit ja im vergleich zu der OE- bzw. Zugriffs-Zeit des Speichers extrem groß ist.
Der von Atmel empfohlene AC573 ist meiner Meinung nach unnötig. Daher verwende ich auch nur HC02 und HC573 und hatte damit noch nie Probleme.

In der param.h sind einige Einstellmöglichkeiten wie z.B. die Anschlussbelegung, sowie einige Konstanten definiert:

#define F_CPU 16000000UL

Hier wird der Quarztakt eingestellt, falls F_CPU nicht global über das AVR Studio definiert wird. Dann kann diese Zeile auskommentiert werden.

#define UART_BAUD_RATE 57600

Hier wird die UART Baudrate eingestellt.

#define Framerate 75

Damit wird die Bildwiederholfrequenz des LCDs eingestellt. Üblich sind etwa 50-80Hz. Wegen den Graustufen sollte man eher einen etwas höheren Wert einstellen, damit das Display nicht flimmert. Allerdings verursacht eine höhere Frequenz natürlich auch eine höhere Rechenleistung. Über etwa 120Hz sollte man daher nie gehen!

#define GRAYSCALE 4

Ist diese Zeile vorhanden/nicht auskommentiert, arbeitet die Software mit 4/8 Graustufen. Ansonsten werden nur 2 dargestellt. Da mit nur 2 Graustufen weniger Speicher benötigt wird als vorhanden ist, existiert ein virtuelles Bild von 1024x240 im Speicher. Es passen also 3 komplette Bilder in den Speicher. Bei 4 Graustufen ist das virtuelle Bild 512x240 Pixel groß. Bei 8 Graustufen ist das virtuelle Bild 336x240 Pixel groß.
Hinter der Zeile wird die Anzahl an Graustufen angegeben. Dieser Wert darf 4 oder 8 betragen.

#define GRAYMOD 1

Ist diese Zeile vorhanden/nicht auskommentiert, wird eine etwas modifizierte Graustufenerzeugung verwendet. Normalerweise würde eine Graustufe dazu führen, dass ein Pixel z.B. nur in jedem 3. Frame an ist. Bei 75fps würde der Pixel also mit 25fps blinken, was man bei großen Flächen als deutliches Flimmern wahrnimmt. Um dies zu vermeiden, wird die Graustufenerzeugung so verändert, dass nicht große Flächen gemeinsam blinken, sondern, die aktiven Pixel werden zeilenweise verteilt, so dass in jedem Bild in jeder Zeile Pixel eingeschaltet sind. Die Verringerung des Flimmerns erkauft man sich durch einen etwas schlechteren Kontrast der Graustufen. Der Wert dahinter beeinflusst die Schrittweite der Modulation. Bei 4 Graustufen eignet sich ein Wert von 1, bei 8 Graustufen erzielt man mit Werten um 3 die besten Ergebnisse bei niedrigen Bildwiederholraten. Der Wert darf nicht größer als Graustufen-2 sein.

#define COMMAND_TIMEOUT 75

Um zu verhindern, dass bei einer Unterbrechung während des Ladens eines Bildes der Controller endlos auf Daten wartet, bricht jede Pause größer als der eingestellte Wert das Laden eines Bildes ab. Die Werte gelten in Frames. Ein Wert von 75 würde also in diesem Fall eine Zeit von 1s ergeben. Der maximale Wert beträgt 255.

#define HWLP

Ist diese Zeile vorhanden/nicht auskommentiert, wird das LP/LOAD/CP1 Signal per Output Compare1A erzeugt. Das Signal ist daher fest auf dem OC1A Pin. Da das Timing nun durch die Hardware erzeugt wird und nicht mehr durch andere Interrupts beeinflusst wird, sind eventuell auftretende Bildstörungen weg. Allerdings wird Timer1 auch von der PWM für den Kontrast verwendet. Der PWM Wert wird in der Software daher entsprechend skaliert, was eventuell etwas ungleiche Abstände zwischen den einzelnen Werten bewirkt.

#define TESTPULSE

Ist diese Zeile vorhanden/nicht auskommentiert, wird an PortB0 ein high Impuls ausgegeben, solange sich der Controller im LCD Timerinterrupt befindet.

#define FASTLOOP

Ist diese Zeile vorhanden/nicht auskommentiert, wird die Schleife mit der Datenausgabe entrollt, also anstelle der Schleife stehen die Befehle 80x im Code. Kostet Flash, ist aber rund 20% schneller.

#define WAITSTATE

Ist diese Zeile vorhanden/nicht auskommentiert, wird das SRAM Timing und auch das LCD Timing verlangsamt. Der verwendete SRAM sollte schneller als etwa 50ns sein, um diesen ohne Waitstates betreiben zu können.

#define XSIZE 320

Diese Zeile gibt die Bildbreite des LCDs in Pixel an. Der Wert darf maximal 1024 betragen, denn dies ist die maximal unterstütze Auflösung. Bei 4 Graustufen reduziert sich der Wert auf 512 und bei 8 Graustufen auf 336.

#define YSIZE 240

Diese Zeile gibt die Bildhöhe in Zeilen an. Da nicht der gesamte 64k Adressbereich des Speicherinterfaces des mega8515 genutzt werden kann, sind hier keine 256 sondern nur 252 möglich. Bei einem mega128, der 4kByte internes SRAM verwendet, sind daher nur 240 Zeilen möglich.

#define XOScan 0

Über diese Zeile ist es möglich, mehr Pixel an das Display zu senden, als beschrieben werden können. Dies ist bei manchen Displays notwendig.

#define YOScan 0

Über diese Zeile ist es möglich, mehr Zeilen an das Display zu senden, als beschrieben werden können. Dies ist bei manchen Displays notwendig, denn einige 320x240 Displays werden mit 1/241 oder 1/242 Duty angesteuert. Dann muss dieser Wert auf 1 bzw. 2 gesetzt werden.

#define LCD8bit

Ist diese Zeile vorhanden/nicht auskommentiert, wird das LCD mit 8 anstelle von 4bits angesteuert. Der 8bit→4bit Multiplexer entfällt dann. Falls das LCD diesen Modus unterstützt, sollte dieser verwendet werden, denn das halbiert nahezu die benötigte Rechenleistung.

#define NLINE 16

Ist diese Zeile vorhanden/nicht auskommentiert, wird an PortD1 ein zusätzliches M (AC drive) Signal ausgegeben, das nicht einmal pro Frame, sondern alle N Zeilen toggelt. Dies kann Übersprechen auf dem LCD verhindern, jedoch sollte vor der Verwendung überprüft werden, ob das sich ergebende Timing DC frei ist. Wenn man sich also nicht auskennt, sollte man am besten das normale M Signal verwenden.

#define DDRAM 1024

Mittels dieses Parameters wird die Startadresse des externen SRAMs eingestellt. Dieser Wert muss größer der Endadresse des internen SRAMs sein.

#define SRAM64

Falls ein SRAM mit 64kByte oder mehr vorhanden ist, wird die Unterstützung dafür hiermit frei geschaltet falls diese Zeile nicht auskommentiert ist. Dann wird die bisher ungenutzte Adressleitung A0 verwendet. Dadurch verdoppelt sich die Bildbreite des virtuellen Bildes und somit auch die maximal unterstützte Auflösung.

#define USEA0

Ist diese Zeile vorhanden/nicht auskommentiert, wird auch A0 des SRAMs verwendet. Stattdessen wird A7 nicht mehr angesteuert. Dies ist notwendig, um LCDs mit 8bit Interface sinnvoll anzusteuern. Bei 64kByte SRAMs wird diese Option automatisch aktiviert, da A0 und A7 angesteuert werden.

#define LARGEMEM 128

Ist diese Zeile vorhanden/nicht auskommentiert, wird die Unterstützung für >64kByte SRAM freigeschaltet. Die virtuelle Auflösung ändert sich dadurch nicht, da nur 64kByte für das Display verwendet werden. Jedes zusätzliche 64k liefert eine weitere Seite mit der virtuellen Auflösung zwischen denen man umschalten kann. Die Speichergröße muss daher in 64kByte Schritten angegeben werden. Unterstützt werden bis zu 1024kByte. Die Adressleitungen A16-A19 kommen an PortB0-3.

#define VLCD_EEPROM

Ist diese Zeile vorhanden/nicht auskommentiert, wird der eingestellte Kontrast im EEPROM abgespeichert und beim Einschalten automatisch wieder geladen. Zum Abspeichern muss zunächst das Komplement des Kontrastwertes als Kontrasteinstellung gesendet werden, also 255-Wert und anschließend der richtige Wert. Dieser wird dann ins EEPROM geschrieben.

#define USE_XFONT

Ist diese Zeile vorhanden/nicht auskommentiert, wird die Unterstützung für zusätzliche Schriftarten in einem externen SPI Flash freigeschaltet. Unterstützt werden bis zu 8 beliebig große Schriftarten. Der Flash wird an PortB4-7 angeschlossen.

#define USE_FONTZOOM

Ist diese Zeile vorhanden/nicht auskommentiert, wird die Unterstützung für große Schriftarten eingebaut. Die vorhandene Schriftart wird dann um Faktor 1-8 gezoomt, wodurch diese pixelig erscheint. Dafür benötigt man keinen externen Flash.

#define USE_FONTPIXEL

Ist diese Zeile vorhanden/nicht auskommentiert, ist der Text pixelgenau positionierbar. Dies ist allerdings mit einem erhöhten Rechenaufwand verbunden, da nun der Text Pixel für Pixel geschrieben werden muss.

Falls zusätzliche Grafikfunktionen in die Software eingebaut werden sollten, hier eine kurze Beschreibung der Schnittstelle:

```
void lcd_writecgram(    unsigned char x,          // X Position in 8 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char nr,         // Zeichennummer, Bereich 0-15
                        unsigned char textcol,    // Textfarbe (2bpp, an MSB ausgerichtet)
                        unsigned char backcol);   // Hintergrundfarbe (2bpp, an MSB ausgerichtet)
```

Schreibt ein benutzerdefiniertes Zeichen auf das LCD an die angegebene Position (x,y).

```
void lcd_writechar (    unsigned char x,          // X Position in 8 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c,          // Zeichen, Bereich 0-255
                        unsigned char tcol,       // Textfarbe (2bpp, an MSB ausgerichtet)
                        unsigned char bcol);      // Hintergrundfarbe (2bpp, an MSB ausgerichtet)
```

Schreibt einen 8x8 / 8x12 Buchstaben auf das LCD an die angegebene Position (x,y).

```
void lcd_string(        unsigned char x,            // X Position in 8 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        char *txt,                  // Nullterminierter Textstring im Flash
                        unsigned char tcol,         // Textfarbe (2bpp, an MSB ausgerichtet)
                        unsigned char bcol);        // Hintergrundfarbe (2bpp, an MSB ausgerichtet)
```
Schreibt einen Text aus dem Flash an Position (x,y) auf das LCD.

```
void lcd_writebyte (    unsigned char x,            // X Position in 8 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c);           // Datenbyte das 8 Pixel enthält
```
Schreibt 1 Byte Pixeldaten (8 Pixel) auf das LCD an die angegebene Position (x,y).

```
void lcd_writebyteg (   unsigned char x,            // X Position in 4 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c);           // Datenbyte das 4 Pixel enthält
```
Schreibt 1 Byte Pixeldaten (4 Pixel) mit 2bpp Farben auf das LCD an die angegebene Position (x,y).

```
void lcd_writebyteg8 (  unsigned char x,            // X Position in 8 Pixel Schritten, Bereich 0-63
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c,            // Datenbytes die 8 Pixel enthalten
                        unsigned char d,
                        unsigned char e);
```
Schreibt 3 Byte Pixeldaten (8 Pixel) mit 3bpp Farben auf das LCD an die angegebene Position (x,y).

```
void lcd_writebyteg8b ( unsigned char x,            // X Position in 8 Pixel Schritten, Bereich 0-63
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char b0,           // Datenbytes die 8 Pixel enthalten
                        unsigned char b1,
                        unsigned char b2);
```
Schreibt 3 Byte Pixeldaten (8 Pixel) mit 3bpp Farben auf das LCD an die angegebene Position (x,y). b0 sind dabei die LSBs der 8bit Pixeldaten, b2 die MSBs.

```
void lcd_writebyteex (  unsigned char x,            // X Position in 8 Pixel Schritten, Bereich 0-63/0-127
                        unsigned char y,            // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c,            // Datenbyte das 8 Pixel enthält
                        unsigned char textcol,      // Farbe die den 1en aus dem Byte zugewiesen
                        unsigned char backcol);     // Farbe die den 0en aus dem Byte zugewiesen
```
Schreibt 1 Byte Pixeldaten (8 Pixel) auf das LCD und erweitert die Farben auf die angegeben Werte.

```
void lcd_setpixel (     unsigned short x,           // X Position in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short y,           // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char c);           // Pixelfarbe (2bpp, an MSB ausgerichtet)
```
Zeichnet einen Pixel an (x,y) mit einer bestimmten Farbe.

```
void lcd_line (         unsigned short x1,          // X Position in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short y1,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned short x2,          // X Position in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short y2,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char color);       // Linienfarbe (2bpp, an MSB ausgerichtet)
```
Zeichnet eine Linie von (x1,y1) nach (x2,y2) mit einer bestimmten Farbe.

```
void lcd_block (        unsigned short x1,          // X Position in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short y1,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned short x2,          // X Position in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short y2,          // Y Position in 1 Pixel Schritten, Bereich 0-239
                        unsigned char color,        // Füllfarbe (2bpp, an MSB ausgerichtet)
                        unsigned char color2);      // Rahmenfarbe (2bpp, an MSB ausgerichtet)
```
Zeichnet ein Rechteck auf das LCD und löscht alles was vorher an dieser Stelle war.

```
void lcd_circle(        unsigned short xm,          // X Position Mittelpunkt in 1 Pixel Schritten, Bereich 0-511/0-1023
                        unsigned short ym,          // Y Position Mittelpunkt in 1 Pixel Schritten, Bereich 0-239
                        unsigned char r,            // Radius in 1 Pixel Schritten, Bereich 1-119
                        unsigned char color1,       // Füllfarbe (2bpp, an MSB ausgerichtet)
                        unsigned char color2);      // Rahmenfarbe (2bpp, an MSB ausgerichtet)
```
Zeichnet ein Kreis auf das LCD und löscht alles was vorher an dieser Stelle war.

```
void lcd_setstartx(     unsigned char pos);         // X Position in 8 Pixel Schritten, Bereich 0-24/0-88
```
Setzt die Bildposition ab der das virtuelle Bild dargestellt wird. Dies ermöglicht mehrere Bildseiten oder auch ein Scrollen in X Richtung.

```
void lcd_clear(         unsigned char pattern);     // Wert mit denen das SRAM beschrieben wird.
```
Löscht das LCD (setzt alle Werte im SRAM auf den angegeben Wert. Dieser Wert entspricht nicht dem Farbwert!)

```
void lcd_init(void);
```
Startet den LCD Controller