320x240 LCD Controller

- LCD Controller für ein 320x240 4bit LCD.

- Schnittstelle: UART

- Optimiert für Textausgabe

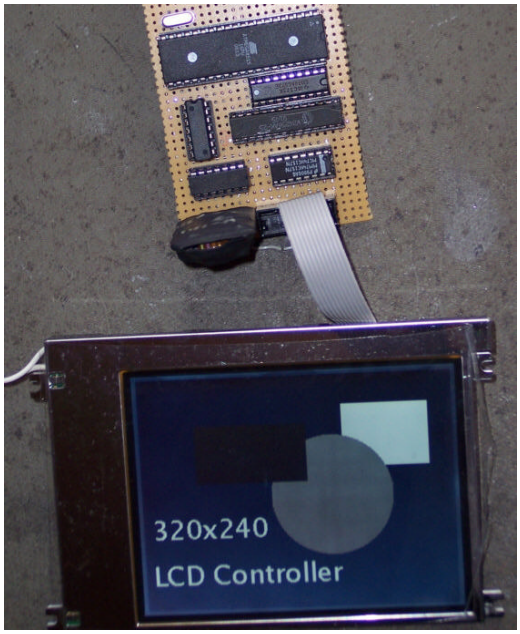
- vollständiger 8x12 Zeichensatz (256 Zeichen)

- 40x20 Zeichen

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

- 4 Graustufen (2bpp), abschaltbar

- 512x240 (bei 2bpp) oder 1024x240 (bei 1bpp) virtuelle Auflösung



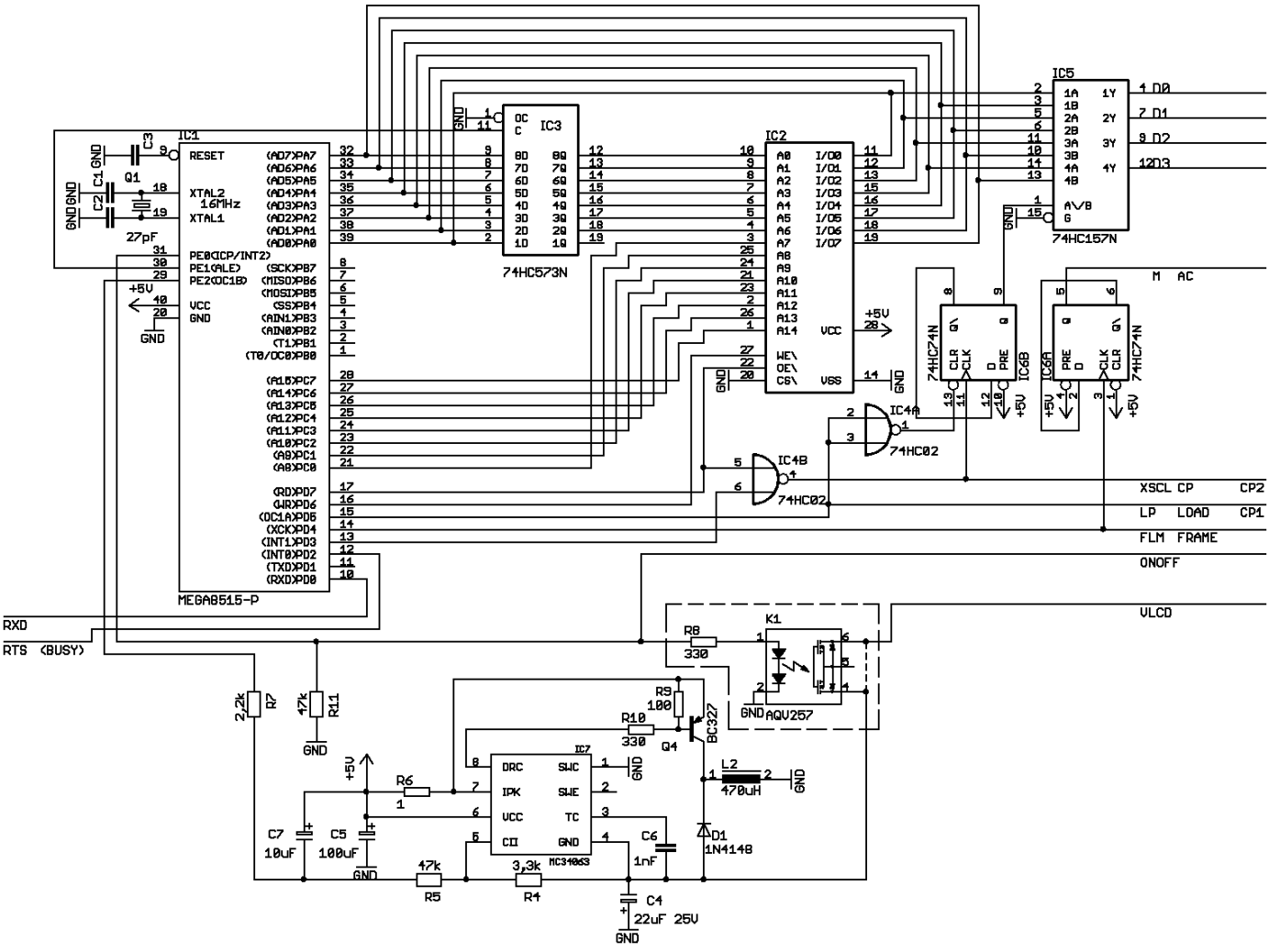
Befehl	Parameter	Beschreibung
0		NOP (nichts machen)
1		Cursor auf (0,0) setzen
2		LCD (Kontrastspannung) aus
3		LCD (Kontrastspannung) an
8		Backspace
9	X Low X High Y Farbe	Set Pixel (x,y,Farbe)
10		Cursor eine Zeile nach unten
11		Zeichen an Cursorposition löschen
12		Komplettes Bild löschen
13		Cursor an Zeilenanfang setzen
14		Nicht verwenden!
15	Kontrast	Kontrast einstellen (0-100)
16	0xAA 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. Modus schaltet zwischen 1bpp (0) und 2bpp (1) um (in der 1bpp Version wird der Wert ignoriert, muss aber gesendet werden). 0xAA (170) dient nur als Sicherheitsbyte, damit nicht versehentlich dieser Befehl (z.B. aufgrund eines Übertragungsfehlers) ausgeführt wird.
17	X Y	Textcursor auf (x,y) setzen X zählt in 8 Pixel Schritten, Y in Zeilenschritten
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.
21	Textfarbe Hintergrundfarbe	Setz Textfarbe und Hintergrundfarbe
22		Nicht verwenden!

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)
27	X	Setzt den X Offset für den Textcursor (für das Beschreiben des virtuellen Bildes).
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)	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

Schaltung:



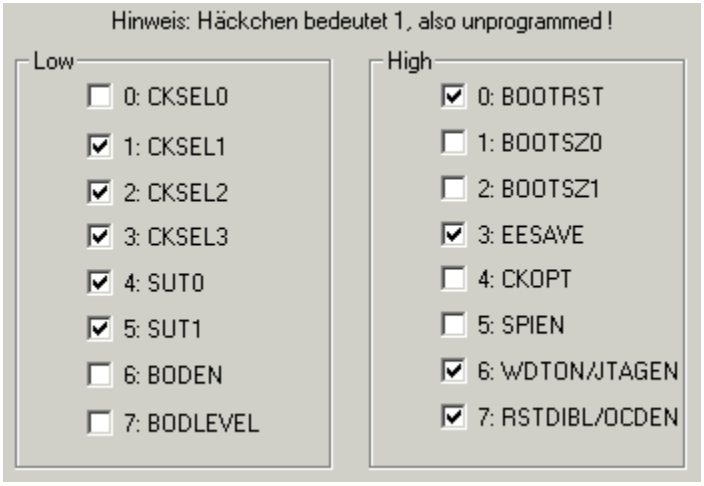
Der OptoMOSFET K1 kann bei LCDs die über einen ONOFF oder DispOFF\ Pin verfügen entfallen. Falls der OptoMOSFET nicht erhältich ist, kann dieser durch einen normalen Optokoppler ersetzt werden, wenn dieser genügend Strom für das LCD schalten kann (Stromverstärkung/Übertragungsverhältnis beachten!).

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 zur 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.

Hinweise zur Ansteuerung:
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:



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 Framerate 75

Damit wird die Bildwiederholfrequenz des LCDs eingestellt. Üblich sind etwa 60-80Hz. 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 100Hz sollte man daher nie gehen!

#define GRAYSCALE

Ist diese Zeile vorhanden/nicht auskommentiert, arbeitet die Software mit 4 Graustufen. Ansonsten werden nur 2 dargestellt. Da mit nur 2 Graustufen weniger Speicher benötigt wird, existiert ein virtuelles Bild von 1024x240 im Speicher. Es passen also 3 komplette Bilder in den Speicher.

#define GRAYMOD

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.

#define COMMAND_TIMEOUT 75

Um zu verhindern, dass bei einer Unterbrechung während des Ladens eines Bildes der Controller nicht 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.

#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 35ns sein, um diesen ohne Waitstates betreiben zu können.

#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.

Hier mal eine genauere Beschreibung 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.
Mit etwas Glück funktionieren aber selbst 100ns SRAMs ohne waitstates. Ausprobiert
habe ich aber 10ns bis 35ns (nur Cache SRAMs).

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-20ns, also 105ns
zwischen Adresse gültig bis Daten stabil, was eigentlich alle SRAMs die
obige Bedingung erfüllen, auch erfüllen.

Das Latch ist da meiner Meinung nach absolut unkritisch, da die
Zugriffszeit ja im vergleich zu der OE Zeit des Speichers extrem groß
ist. Ich verstehe bis heute nicht, warum Atmel da ein AC573 empfiehlt.
Daher verwende ich auch nur HC02 und HC573.
```

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 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_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_clear(void);
```

Löscht das LCD (setzt alle Pixel auf 0)

```
void lcd_init(void);
```

Startet den LCD Controller

```
void lcd_setstartx(unsigned char pos);
```

Setzt die Bildposition ab der das virtuelle Bild dargestellt wird. Dies ermöglicht mehrere Bildseiten oder auch ein Scrollen in X Richtung.