Emulation der EM80

Inhalt

Verzeichnis aller Abbildungen

Vorwort

Angeregt durch die Arbeiten an Emulationen magischer Augen durch BernhardWGF im WGF-Forum haben sich Peter (aka „volvofan58“) und Wolfgang (aka „Rumgucker“) folgende Ziele gesetzt:

- Offenlegung aller Details im WGF-Forum
- Emulation einer baukleinen Röhre (also EM80)
- Kosten rund € 5,--
- Hinreichende Emulation aller Röhreneigenschaften
- Anpassbarkeit an andere Röhren, z.B. Stereo-Anzeigeröhren

Diese Doku soll keine Bauanleitung für einen Röhrenersatz im hübschen Glaskolben mit eingelassenen Sockelstiften sein. Es soll ebenso wenig eine Einführung in die Hard- und Softwareentwicklung darstellen.

Nachfolgende Seiten wollen lediglich Bernhards Inspirationen aufnehmen und an einem Beispiel durchführen.

Diese Doku soll erfahrene Hard- und Softwarentwickler zu weiteren Arbeiten und Offenlegungen anregen.

Verwendete Hilfsmittel

- Programm MS Paint 5.1
- Programm IMPBMP (DIY, s. Anhang)
- Programm IMPCNT (DIY, s. Anhang)
- Entwicklungswerkzeuge für die DIY-Programme

- Entwicklungswerkzeug AVR Studio 4.18
- Programmiergerät GALEP 4

- Programm LT Spice

- Datenblätter, Netzteile, Scopes, Multimeter, Steckbrett

Hardware

Display

Wegen der Kleinheit, Reaktionsgeschwindigkeit und des guten Kontrastes wird ein OLED-Display verwendet. Gefordert sind folgende Maße:

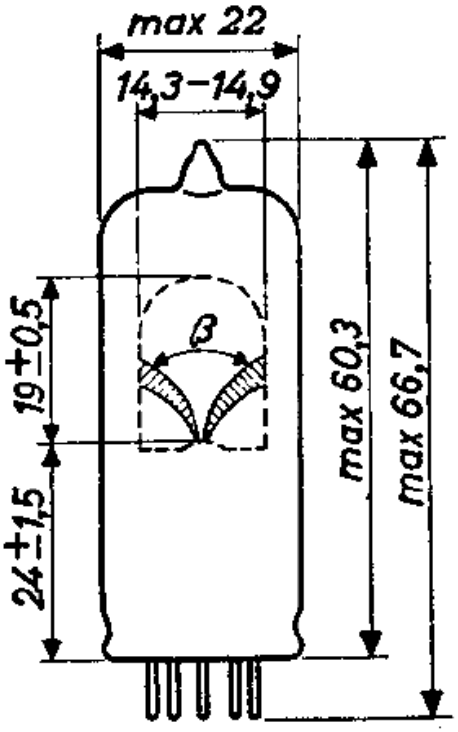


Abbildung 1: Maße EM80

Es gelang nicht, ein grün leuchtendes und preisgünstiges OLED-Display mit diesen Maßen zu finden. Für die Entwicklung der Soft- und Hardware des Labormusters wurde daher ein weißes 0,96“-Display verwendet:

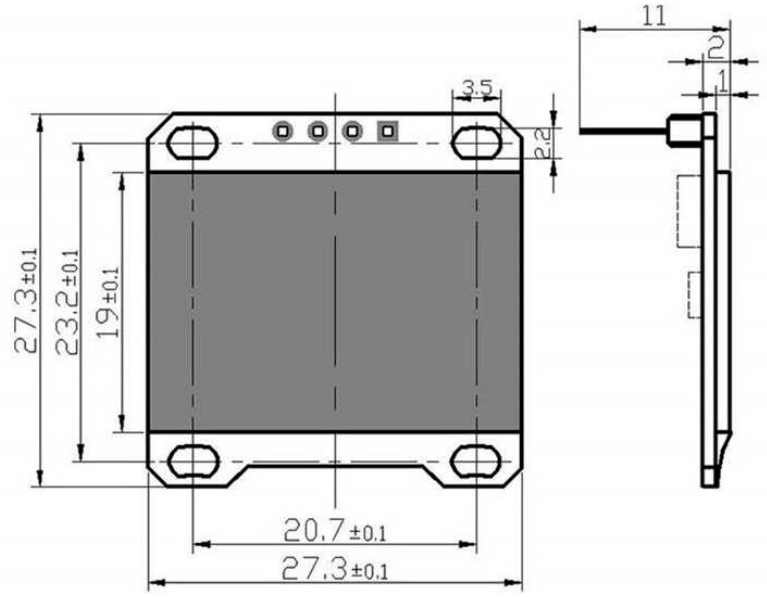
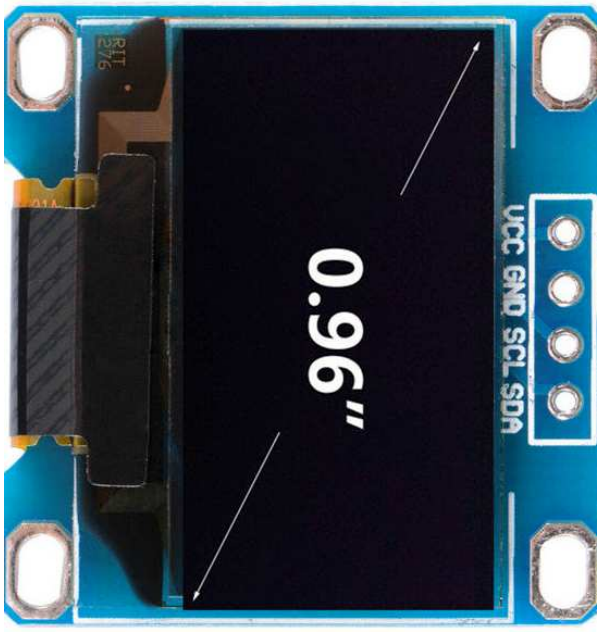


Abbildung 2: Maße OLED-Display



Die Display-Auflösung beträgt 128 x 64 Pixel.

Der aktive Bereich des hochkant stehenden Displays misst 11 x 22 mm. Das Emulationsbild ist also etwas zu schmal und nach rechts versetzt.

Zwischen Emulationsbild-Unterkante und Sockelstiften wird die Elektronik angebracht. Daher sollte das Emulationsbild ganz an die untere Kante geschoben werden.

Zwischen Displayfläche und Folienleiter befindet sich der OLED-Controller (z.B. SSD1306. SSH1306 oder ein SH1106).

Abbildung 3: Frontansicht OLED-Display

Das eigentliche Display ist auf eine Interface-Platine aufgeklebt und mit Folienleiter angelötet.

Die Platine beinhalt I²C-Pullup- und Entkopplungs-Widerstände, Reset RC-Glied, Blockkondensatoren und Kondensatoren zum Betrieb einer internen Ladungspumpe. Im Controller befindet sich ein 3.3V-LDO, so dass eine Speisung und Steuerung von 3.3 bis 5V möglich ist.

Das Display wird lediglich mit den I²C-Signalen „Clk" und „Data" gesteuert. Die ingesamt nur vier Anschlussdrähte sparen Platz, Kosten und Controller-Pins.

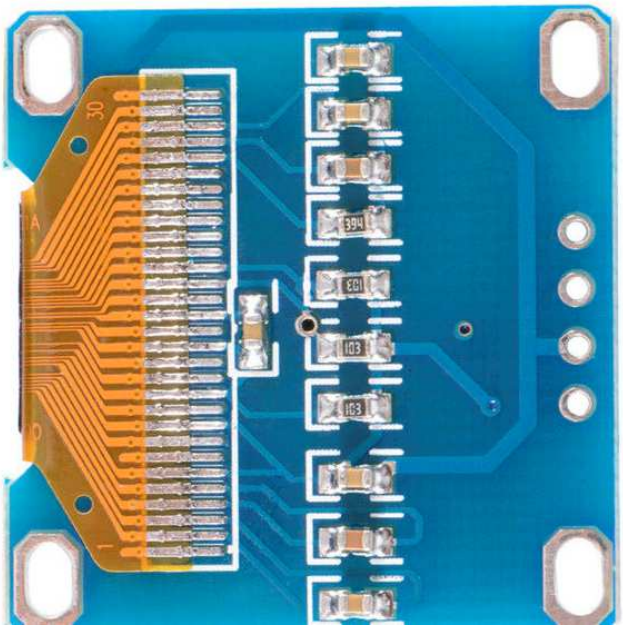


Abbildung 4: Rückansicht OLED-Display

Der Umbau auf vielfach schnelleres ISP-3 bzw. ISP-4 durch einfaches Auftrennen der zwei eingekreisten Verbindungen scheiterte, da die hierfür umzuschaltenden Selektionsanschlüsse (roter Kasten) entweder innerhalb des Controllers oder unterhalb des Folienleiters verbunden waren.

Dabei wurde nach dem Anschlussplan auf der nächsten Seite gearbeitet.

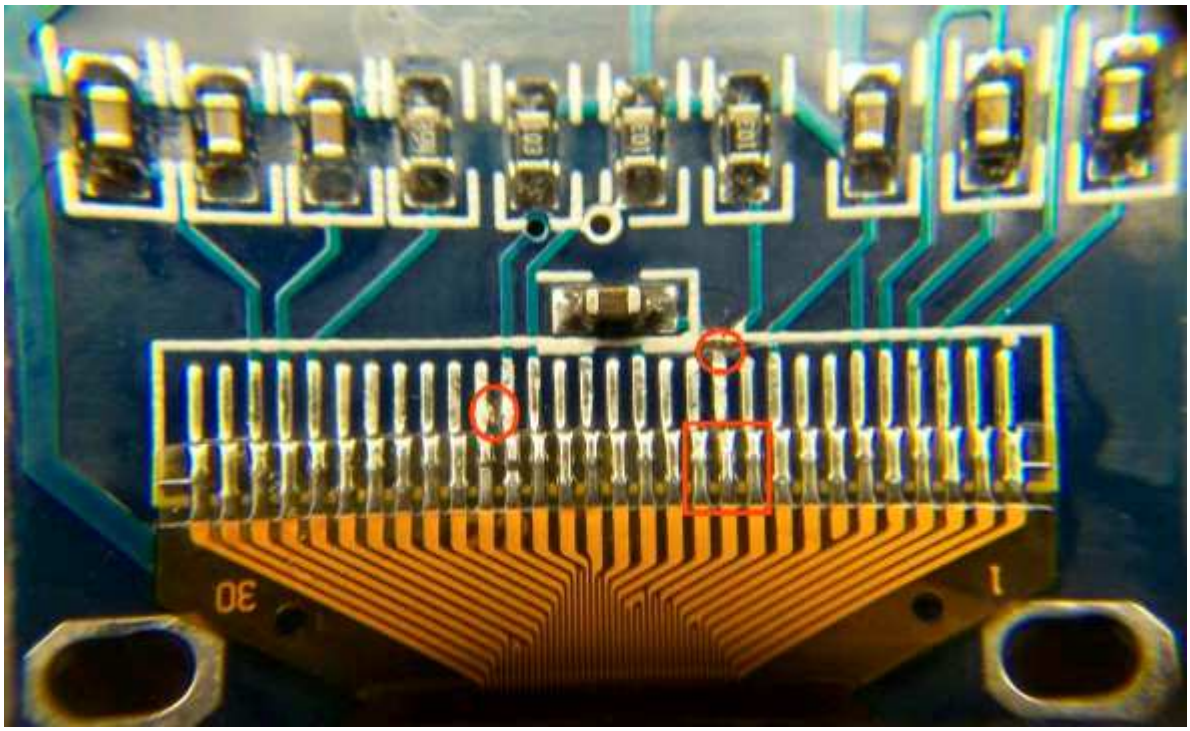


Abbildung 5: Umarbeitung des OLED-Displays

Es gelang auch nicht, das Display oder den Folienleiter von der Interface-Platine zu lösen.

Lediglich die im linken Kreis markierte Unterbrechung wurde beibehalten. Es wurde also D2 als I²C-Rückmeldeleitung gekappt. Das Display kann so lediglich Daten empfangen und keine Quittungen mehr zurücksenden. Diese Vereinfachung ist zulässig, wenn keine weiteren Einheiten an dem I²C-Bus angeschlossen werden und der Display-Controller bis auf ACK/NAK keine weitere Kommunikation zurücksendet.

Dadurch ergibt sich die Möglichkeit, den langsamen bidirektionalen Betrieb mit pullup-Widerständen durch schnelle Gegentaktpegel zu ersetzen. Und es erspart im Treiber die dauernden Umschaltungen der Datenflussrichtung.

Der hierfür erstellte Treiber erreicht statt der 400 kbit/s einer normalen I²C-Übertragung nunmehr 1,8 Mbit/s und liegt damit nur noch wenig unter der Geschwindigkeit von ISP.

PIN NO.	SYMBOL	FUNCTION
1	NC (GND)	Reserved pin (supporting pin) The supporting pins can reduce the influences from stresses on the function pins. These pins must be connected to external ground.
2	C2N	Positive terminal of the flying inverting capacitor negative terminal of the flying boost capacitor The charge-pump capacitors are required between the terminals. They must be floated when the converter is not used.
3	C2P	
4	C1P	
5	C1N	
6	V_{BAT}	Power supply for DC/DC converter circuit This is the power supply pin for the internal buffer of the DC/DC voltage converter. It must be connected to external source when the converter is used. It should be connected to V_{DD} when the converter is not used.
7	NC	NC
8	V_{SS}	Ground of logic circuit This is a ground pin. It also acts as a reference for the logic pins. It must be connected to external ground.
9	V_{DD}	Power supply for logic circuit. This is a voltage supply pin. It must be connected to external source.
10	BS0	Communicating protocol select These pins are MCU interface selection input. See the following table:
11	BS1	
12	BS2	
13	CS#	Chip select This pin is the chip select input. The chip is enabled for MCU communication only when CS# is pulled low.
14	RES#	Power reset for controller and driver This pin is reset signal input. When the pin is low, initialization of the chip is executed.
15	D/C#	Data / command control This pin is data / command control pin. When the pin is pulled high, the input at D7 to D0 is treated as display data. When the pin is pulled low, the input at D7 to D0 will be transferred to the command register. For detail relationship to MCU interface signals, please refer to the timing characteristics diagrams. When the pin is pulled high and serial interface mode is selected, the data at SDIN is treated as data. When it is pulled low, the data at SDIN will be transferred to the command register. In I^2C mode, this pin acts as SA0 for slave address selection.
16	R/W#	Read / write select or write This pin is MCU interface input. When interfacing to a 68XX-series microprocessor, this pin will be used as read / write (R/W#) selection input. Pull this pin to "high" for read mode and pull it to "low" for write mode. When 80XX interface mode is selected, this pin will be the write (WR#) input. Data write operation is initiated when this pin is pulled low and the CS# is pulled low.
17	E/RD#	Read / write enable or read This pin is MCU interface input. When interfacing to a 68XX-series microprocessor, this pin will be used as the enable (E) signal. Read / write operation is initiated when this pin is pulled high and the CS# is pulled low. When connecting to an 80XX-microprocessor, this pin receives the read (RD#) signal. Data read operation is initiated when this pin is pulled low and CS# is pulled low.
18 to 25	D0 to D7	Host data input / output bus These pins are 8-bit bi-directional data bus to be connected to the microprocessor's data bus. When serial mode is selected, D1 will be the serial data input SDIN and D0 will be the serial clock input SCLK. When I^2C mode is selected, D2 and D1 should be tired together and serve as SDA_{out} and SDA_{in} in application and D0 is the serial clock input SCL.
26	I_{REF}	Current reference for brightness adjustment This pin is segment current reference pin. A resistor should be connected between this pin and V_{SS}. Set the current lower than 12.5 µA.
27	V_{COMH}	Voltage output high level for COM signal This pin is the input pin for the voltage output high level for COM signals. A capacitor should be connected between this pin and V_{SS}.
28	V_{CC}	Power supply for OEL panel This is the most positive voltage supply pin of the chip. A stabilization capacitor should be connected between this pin and V_{SS} when the converter is used. It must be connected to external source when the converter is not used.
29	V_{LSS}	Ground of analog circuit This is an analog ground pin. It should be connected to V_{SS} externally.
30	NC (GND)	Reserved pin (supporting pin) The supporting pins can reduce the influences from stresses on the function pins. These pins must be connected to external ground.

	I^2C	3-wire SPI	4-wire SPI	8-bit 68XX parallel	8-bit 80XX parallel
BS0	0	1	0	0	0
BS1	1	0	0	0	1
BS2	0	0	0	1	1

Abbildung 6: Anschlüsse Folienleiter

Controller und Analogteil

Es wurde aus Platz- und Kostengründen ein möglichst geringer Aufwand angestrebt.

Die EM80 verfügt über zwei Stromkreise, die galvanisch voneinander getrennt sind. Die Heizung. Und das Elektrodensystem.

Der Emulator trennt diese beiden Schaltungsteile mit einem Optokoppler. Das Elektrodensystem wird mit diskreten Bauteilen nachgebildet und aus den Anodenspannungen versorgt. Das Display und der Controller werden von der Heizung gespeist.

Es wurde Einweggleichrichtung eingesetzt, weil die Stromaufnahme nur 20mA beträgt und der Nachteil des erhöhten Spannungsabfalls einer Vollbrücke den Vorteil der verdoppelten Siebfrequenz übersteigt.

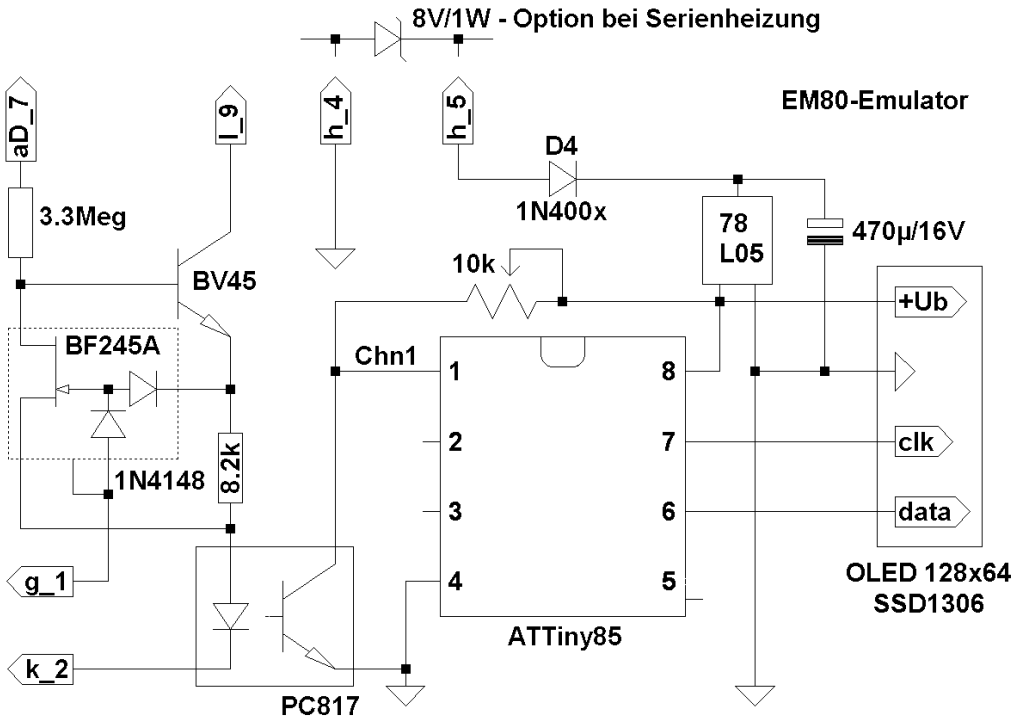


Abbildung 7: Gesamtschaltbild

Der Emulator toleriert Heizspannungen zwischen 4V~ und 12V~. Es ist auch Gleichstromheizung möglich (Pin 5 = Plus). Und es ist Serienheizung möglich, wenn man unterhalb der Röhre die eingezeichnete Zenerdiode zur Kommutierung anbringt. Es ist erlaubt, die Heizspannung um bis zu 1000V hochzulegen. Es wird ein ATTiny85 (8 kByte Flash, 512 Byte RAM) verwendet, der mit dem internen 16 MHz-Takt betrieben wird. Es verbleiben drei freie Pins zum Anschluss anderer Displays oder eines Stereo-Kanals.

Der linke Schaltungsteil bildet einen invertierenden Operationsverstärker (V=-1), dessen Ausgangsspannung einen proportionalen Optokopplerstrom treibt. Der Gitterstrom bei negativen Gitterspannungen wird durch den Sperrstrom der Si-Dioden definiert. Die Dioden und der FET müssen daher geschirmt werden.

Bei positiven Gitterspannungen fließt Gitterstrom. FETs mit Uth < 800mV sind brauchbar. Der Widerstand am Pin 7 sorgt dabei durch den Basisstrom des Hochvoltranistors BV45 für Betriebssicherheit bei hohen negativen Gitterspannungen.

MOSFETs (z.B. 2N7002) sind wegen deren hoher Eingangskapazität nicht brauchbar. Die Eingangskapazität des FETs bildet zusammen mit den Gigaohm-Widerständen der in Sperrrichtung betriebenen Dioden einen Tiefpass, der die Emulation träge macht. In der gezeigten Schaltung wird eine günstige Zeitkonstante von unter 10ms erreicht.

Der Optokopplerstrom wird von den Leuchtschirmspannung an Pin 9 bereitgestellt. Es fließt rund 1mA. Der Feinabgleich wird mit dem Trimmer auf Controllerseite durchgeführt. Innerhalb des Controllers befindet sich eine Kennlinientabelle.

Software

In einer Endlosschleife wird der Analogwert vom ADC eingelesen, dazu in vier Schritten ein passendes Schirmbild erstellt und dieses Bild und dessen Spiegelung an das Display übertragen.

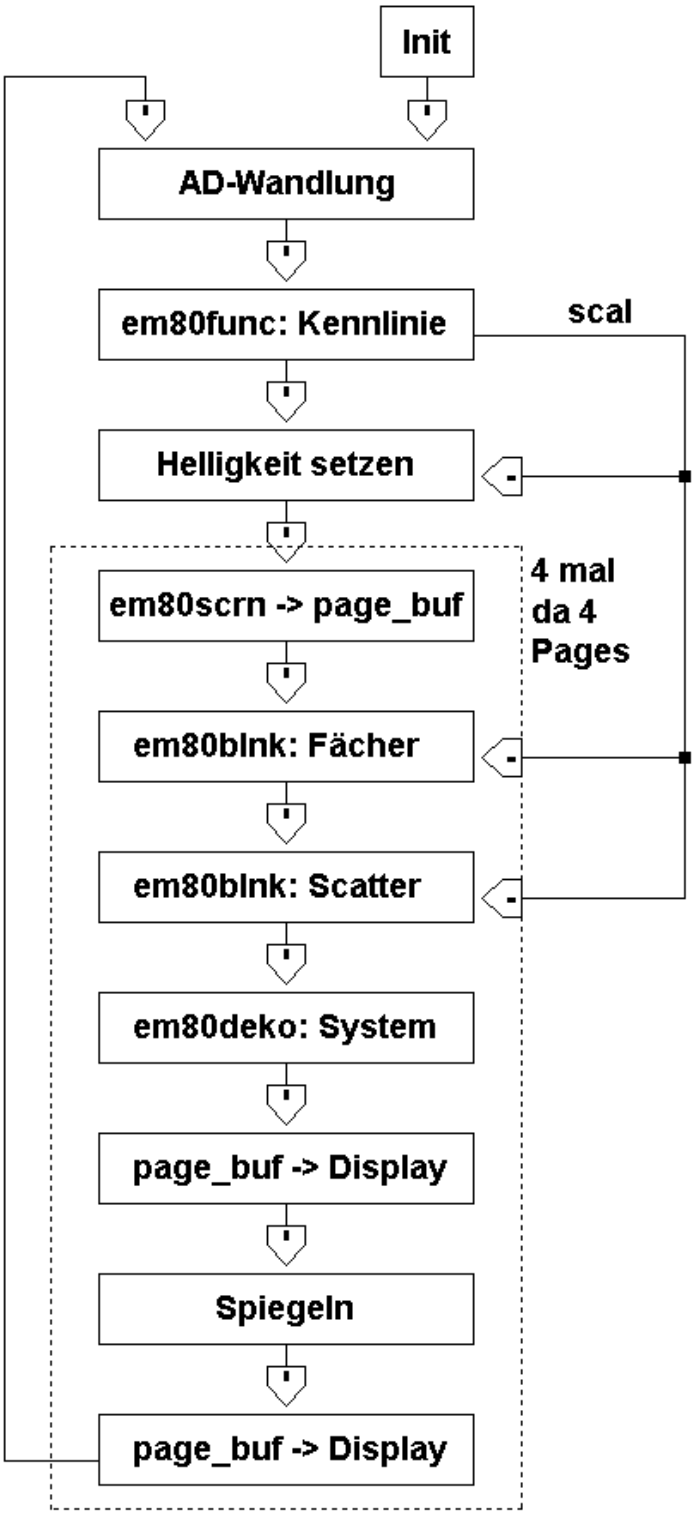


Abbildung 8: Softwareübersicht

Zusätzlich konnten folgende Feinheiten eingearbeitet werden:

- Nachbildung der Leuchtschirmkrümmung
- Nachbildung des Elektrodensystems
- Nachbildung der Helligkeitsminderung beim Auffächern
- Aufheizung ohne Leuchtschirmbild
- Nachbildung der Triodenverstärkungssteigerung während weiterer Aufheizung

Zur komfortablen Grafikerstellung wurde eine Datenübernahme vom Windows Malprogramm „Paint“ erstellt:

- Grafische Erstellung der Röhrenkennlinie
- Grafische Erstellung des Fächers mit Leuchtschirmkrümmung
- Grafische Erstellung des Leuchtschirmes
- Grafische Erstellung des Elektrodensystems

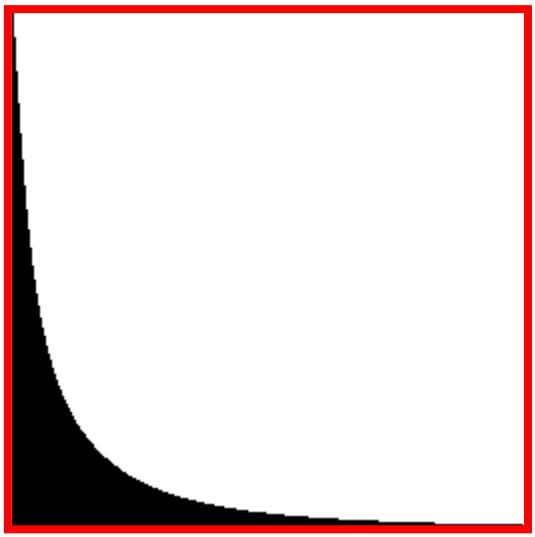


Abbildung 9: Röhrenkennlinie

Abbildung 10: Fächer mit Leuchtschirmkrümmung

Abbildung 11: Leuchtschirm

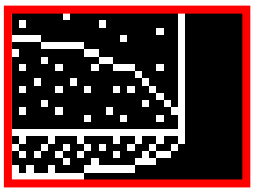
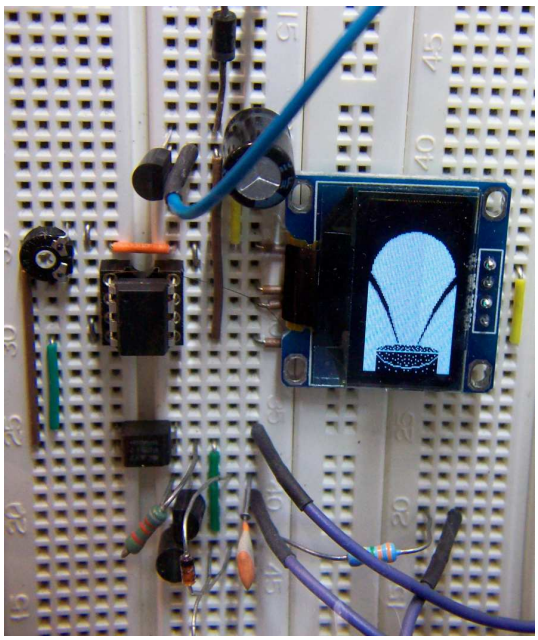


Abbildung 12: Elektrodensystem

So entsteht ein relativ realistisches Aussehen:



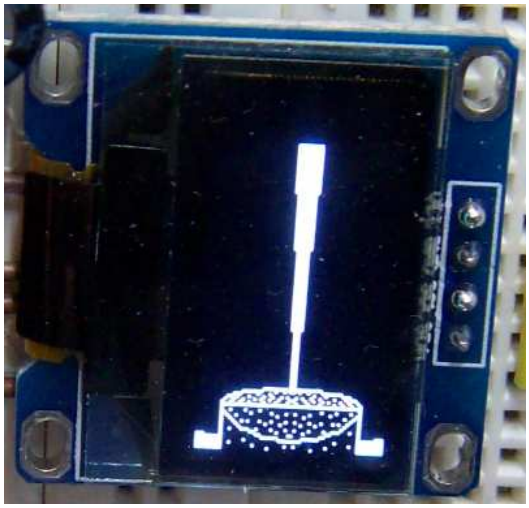


Abbildung 13: Fächer geschlossen

Abbildung 14: Fächer geöffnet

Inkl. ADC und der gezeigten Bildverarbeitung und Spiegelungen werden rund 150 Vollbild-Übertragungen pro Sekunde erzielt.

Auch die Controllerspeicher werden nur wenig genutzt:

Flash: 2302 bytes (28.1% Full)
RAM: 128 bytes (25.0% Full)
EEPROM: 0 Bytes (0.0% Full)

Display-Treiber

Für eine trägheitsfreie Darstellung muss der Display-Treiber möglichst zügig arbeitet. Das vom Display-Controller verwendete I²C-Format:

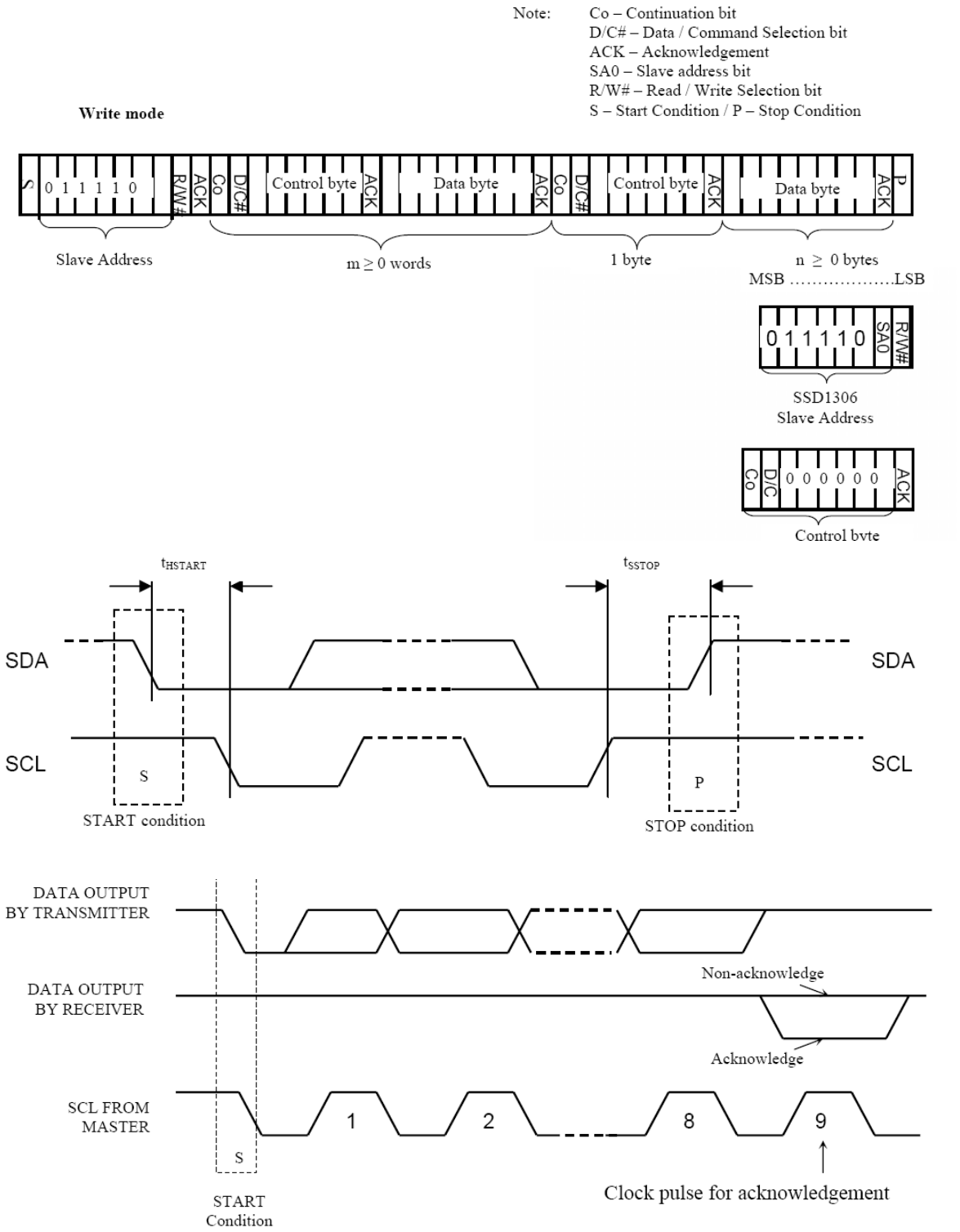


Abbildung 15: I²C-Protokoll

Die Initialisierung des Ports bewirkt ein High-Signal an Clock- und Datenleitung, die als Ausgänge betrieben werden. Da diese Sequenz nur einmal im gesamten System verwendet wird, wurde ein Makro formuliert.

```
#define USI_INI {                                   /* Data und Clock auf High */\
        USI_HIGH(USI_CLK | USI_DATA);       \
        USI_OUT(USI_CLK | USI_DATA);        \
}
```

Die eigentliche Kommunikation beginnt mit einem Start-Kommando. Dabei wird zuerst die Daten- und dann die Clockleitung nach Masse gezogen. Auch dieser Code ist als Makro geschrieben, weil eine Prozedur kaum Code sparen würde.

```
#define SEND_START {                                /* I²C-Start */             \
        USI_LOW(USI_DATA);                          /* Data auf low*/           \
        USI_DELAY;                                                              \
        USI_LOW(USI_CLK);                           /* Clock auf low*/          \
        USI_DELAY;                                                              \
}
```

Wie zuvor gezeigt, endet die Kommunikation mit einem Stop-Kommando, bei dem zuerst die Clock- und dann die Datenleitung auf High-Pegel springt. Zuvor muss allerdings der konkurrierende USI-Mode abgeschaltet werden, damit die direkten Portzugriffe funktionieren. Auch dieser Block wurde als Makro formuliert.

```
#define SEND_STOP {                                 /* I²C-Stop */              \
        USICR = 0;                                  /* USI-Mode aus */          \
        USI_HIGH(USI_CLK);                          /* Clock auf high */        \
        USI_DELAY;                                                              \
        USI_HIGH(USI_DATA);                         /* Data auf high */         \
        USI_DELAY;                                                              \
}
```

Die einzige Prozedur des Treibers gestattet die Übertragung ganzer Datenblöcke. Wahlweise aus dem Programmspeicher oder aus dem RAM. Zur Beschleunigung wird die interne Hardwareunterstützung „USI“ genutzt. Der Acknowledge-Takt wird zwar erzeugt, aber das Bit wird weder gesendet noch ausgewertet, da der Rückkanal des Display abgeklemmt wurde.

```
// *** sende feste Bytes aus dem Flash oder variable Bytes aus dem RAM
void send_buffer(BOOL flash,unsigned char* buf,unsigned char buf_len)
{
        while(buf_len--) {                          // buffer byte für byte abarbeiten
                if(flash) USIDR = pgm_read_byte(buf++);  // aus dem flash oder RAM
                else USIDR = *buf++;                // Datenregister laden und MSB ausgeben
                unsigned char bit_cnt = 9;          // 8 Datenbits + ACK-Dummy
                FOREVER {                           // alle Bits abarbeiten
                        USI_DELAY;                  // etwas warten
                        USI_CLK_CHANGE;             // USI-Mode ein, USI_CLK auf high
                        USI_DELAY;                  // etwas warten
                        USI_CLK_CHANGE;             // USI_CLK auf low
                        if(!--bit_cnt) break;       // raus, wenn alle Bits erledigt
                        USI_DATA_SHIFT;             // Datenregister schieben
                }                                   // bit für bit
        }                                           // byte für byte
}
```

Der ganze Treiber wird als usi_drv.h ins Hauptprogramm eingebunden.

```
//********* OLED-Treiber
// Da ich die I²C-Datenrückleitung des OLED-Displays abgeklemmt hab, muss ich
// kein AKC/NACK empfangen und komme ohne Tristate aus. Dadurch 1.8 MBit/s.
// Es werden trotzdem 9 Bits pro Byte benötigt. Das 9. Bit ist dummy-Takt.

#define USI_OUT(b)                      DDRB |= (b)             // Portpins als Ausgang
#define USI_HIGH(b)                     PORTB |= (b)            // Portpins auf High
#define USI_LOW(b)                      PORTB &= ~(b)           // Portpins auf Low

#define USI_CLK_CHANGE                  USICR = _BV(USIWM0) | _BV(USITC)
#define USI_DATA_SHIFT                  USICR = _BV(USIWM0) | _BV(USICLK)

#define USI_DELAY                       asm volatile ("nop")

#define USI_INI {                                               /* Data und Clock auf High */\
        USI_HIGH(USI_CLK | USI_DATA);       \
        USI_OUT(USI_CLK | USI_DATA);        \
}

#define SEND_START {                                            /* I²C-Start */             \
        USI_LOW(USI_DATA);                                      /* Data auf low*/           \
        USI_DELAY;                                                                          \
        USI_LOW(USI_CLK);                                       /* Clock auf low*/          \
        USI_DELAY;                                                                          \
}

#define SEND_STOP {                                             /* I²C-Stop */              \
        USICR = 0;                                              /* USI-Mode aus */          \
        USI_HIGH(USI_CLK);                                      /* Clock auf high */        \
        USI_DELAY;                                                                          \
        USI_HIGH(USI_DATA);                                     /* Data auf high */         \
        USI_DELAY;                                                                          \
}

// *** sende feste Bytes aus dem Flash oder variable Bytes aus dem RAM
void send_buffer(BOOL flash,unsigned char* buf,unsigned char buf_len)
{
        while(buf_len--) {                                      // buffer byte für byte abarbeiten
                if(flash) USIDR = pgm_read_byte(buf++); // aus dem flash oder RAM
                else USIDR = *buf++;                            // Datenregister laden und MSB ausgeben
                unsigned char bit_cnt = 9;                      // 8 Datenbits + ACK-Dummy
                FOREVER {                                       // alle Bits abarbeiten
                        USI_DELAY;                              // etwas warten
                        USI_CLK_CHANGE;                         // USI-Mode ein, USI_CLK auf high
                        USI_DELAY;                              // etwas warten
                        USI_CLK_CHANGE;                         // USI_CLK auf low
                        if(!--bit_cnt) break;                   // raus, wenn alle Bits erledigt
                        USI_DATA_SHIFT;                         // Datenregister schieben
                }                                               // bit für bit
        }                                                       // byte für byte
}

/* ENDE */
```

Abbildung 16: Listing des I²C-Treibers „usi_drv.h“

Eine exemplarische Nutzung des Treibers (die Controllerbefehle sind im Datenblatt des SSD1306 gelistet), wie sie in fast gleicher Form auch im Hauptprogramm verwendet wird:

```
#include „usi_drv.h“                                    // usi-Treiber einbinden

PROGMEM const unsigned char oled_init[] = {
                0x3C << 1,                              /* adresse */
                0x00,                                   /* control byte. Von nun ab nur Commands */
                0x8D, 0x14,                             /* set charge pump, turn on charge pump */
                0xAF,                                   /* display on */
                0x81                                    /* helligkeit im nächsten byte */
};

int main()
{
        unsigned char light = 0;

        USI_INI;                                        // das wird nur einmal gemacht

        while(1) {
                SEND_START;                             // init oled, setze helligkeit
                send_buffer(TRUE,(unsigned char*)oled_init,sizeof(oled_init));          // aus Flash senden
                light++;                                //erhöhe die helligkeit bis überlauf
                send_buffer(FALSE,&light,1);            // ein byte helligkeit aus dem RAM senden
                SEND_STOP;
        }
}
```

Schirmbild

Das Leuchtschirmbild wird mit Windows Paint als „em80scrn.bmp“ gemalt.

Es genügt die Erstellung eines Halbbildes, da die Software lediglich ein Halbbild verarbeitet. Die zweite Hälfte des Gesamtbildes wird durch Spiegelung erzeugt.

Paint speichert die erste Zeile dieses hochkant stehenden 32 x 128 Pixel großen Bildes mit vier Bytes ab. Beginnend mit dem MSB des ersten Bytes und endend mit dem LSB des vierten Bytes. In der zweiten der insgesamt 128 Zeilen wird sinngemäß gespeichert. Also MSB des 5.Bytes bis LSB des 8.Bytes. Das Bild benötigt also 4 Bytes mal 128 Zeilen = 512 Bytes.

Die Paint-Datei wird mit einem DOS-Box-Programm „IMPBMP.EXE“ (im Anhang gelistet) in einen `C`-Header „em80scrn.h“ umgewandelt, der diese beschriebene Datenstruktur direkt wiedergibt:

```
PROGMEM const unsigned char em80scrn[512] = {
                                0x00, 0x00, 0x00, 0xFF,
                                0x00, 0x00, 0x00, 0xFF,
                                ......
                                0x00, 0x00, 0x00, 0xFF,
                                0x00, 0x00, 0x00, 0xFF,
                                0x00, 0x00, 0x01, 0xFF,
                                0x00, 0x00, 0x03, 0xFF,
                                0x00, 0x00, 0x0F, 0xFF,
                                0x00, 0x00, 0x7F, 0xFF,
                                0x00, 0x3F, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFF,
                                .........
                                0xFF, 0xFF, 0xFF, 0xFF,
                                0xFF, 0xFF, 0xFF, 0xFE,
                                0xFF, 0xFF, 0xFF, 0xFE,
                                0xFF, 0xFF, 0xFF, 0xFE,
                                0xFF, 0xFF, 0xFF, 0xFE,
                                0xFF, 0xFF, 0xFF, 0xFE,
                                0xFF, 0xFF, 0xFF, 0xFC,
                                0xFF, 0xFF, 0xFF, 0xFC,
                                0xFF, 0xFF, 0xFF, 0xFC,
                                0xFF, 0xFF, 0xFF, 0xFC,
                                0xFF, 0xFF, 0xFF, 0xFC,
                                0xFF, 0xFF, 0xFF, 0xF8,
                                0xFF, 0xFF, 0xFF, 0xF8,
                                0xFF, 0xFF, 0xFF, 0xF8,
                                0xFF, 0xFF, 0xFF, 0xF8,
                                0xFF, 0xFF, 0xFF, 0xF0,
                                0xFF, 0xFF, 0xFF, 0xF0,
                                0xFF, 0xFF, 0xFF, 0xE0,
                                .........
                                0x00, 0x00, 0x00, 0x00,
                                0x00, 0x00, 0x00, 0x00,
                                0x00, 0x00, 0x00, 0x00,
};
```

Abbildung 17: Listing des Leuchtschirmbildes „em80scrn.h“

Dieser Header wird in das Hauptprogramm eingebunden.

Der Controller des Displays erwartet seine Daten in folgender Organisation:

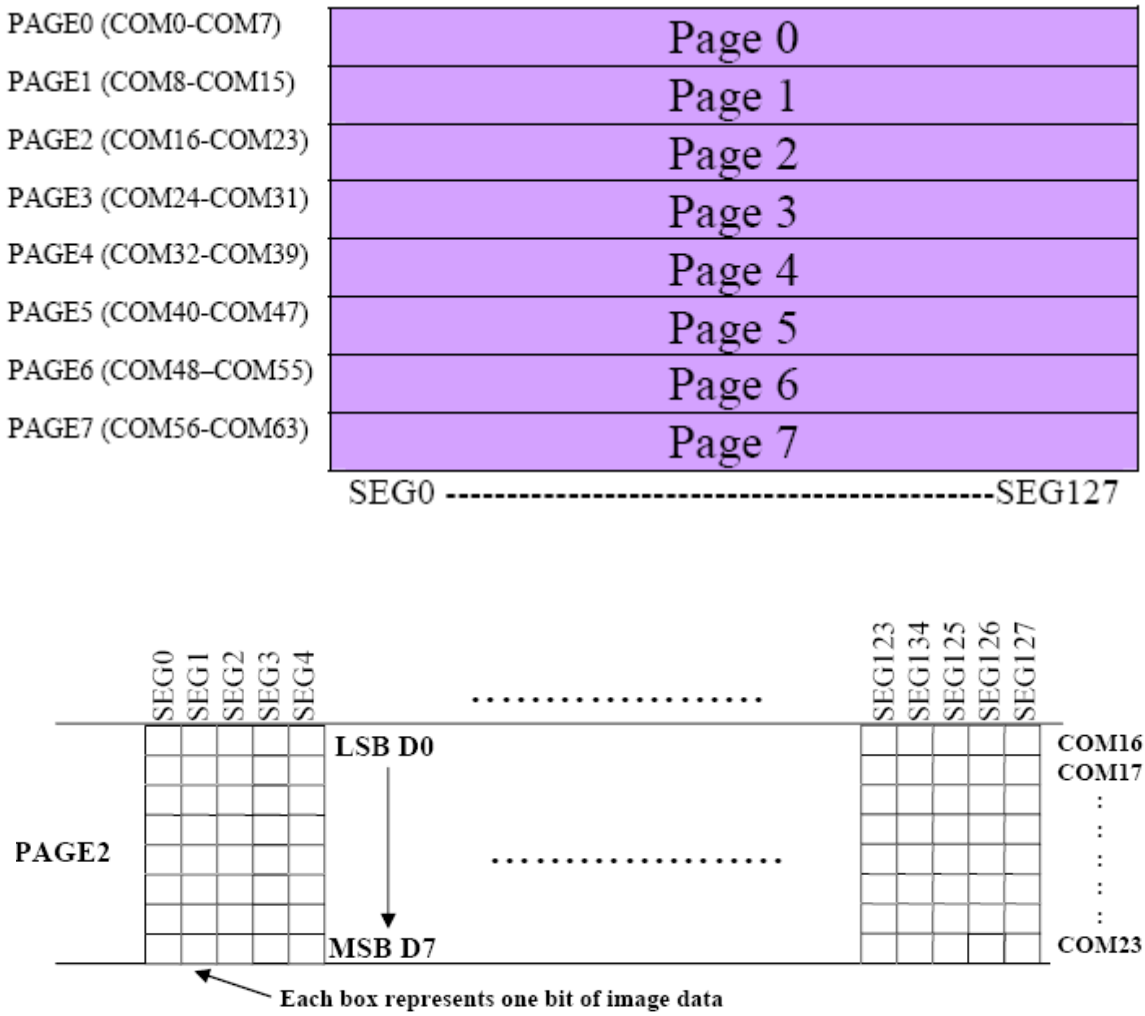


Abbildung 18: Datenformat des OLED-Displays

Um also beispielsweise die oberste Page 0 des Controllers zu befüllen, muss lediglich folgende Schleife programmiert werden:

```
unsigned char page_buf[128];                    // arbeitsspeicher für grafik und oled-ausgabe

void scrn_2_pagebuf(void)
{
        const unsigned char* scrn = em80scrn + 3;  // 4.byte in Paint -> 1.Byte der Controller Page 0
        unsigned char page_col;

        for(page_col = 0;page_col < sizeof(page_buf);page_col++,scrn += 4)
                page_buf[page_col] = pgm_read_byte(scrn);

}
```

Nach dem Durchlauf der Schleife befinden sich alle Bytes aller 128 Spalten der Page 0 in richtiger Anordnung im page_buf und können als Block an das Display gesendet werden. Es würde 1/8 des OLED-Bildes sichtbar werden.

Leuchtfächer

Die simple Abbildung des „em80scrn“-Schirmbildes reicht jedoch noch nicht. Denn die weißen Pixel des Schirmbildes beschreiben lediglich die Pixel, die leuchten könnten. Ob sie dann wirklich leuchten ist vom Leuchtfächer abhängig.

Der Leuchtfächer wird in Paint als „em80blnk.bmp“ erstellt. Allerdings in liegender Orientierung mit 128 Spalten und 64 Zeilen:

Ein weiteres im Anhang gelistetes DOS-Box-Tool „CNTBMP“ konvertiert diese Zeichnung in eine Tabelle, in der für jede Zeile die Anzahl der schwarzen (dunklen) Pixel in der Zeile gezählt werden:

```
PROGMEM const unsigned char em80blnk[64] = {
                                 2,  5,  8, 11, 14, 17, 20, 23,
                                26, 29, 31, 34, 37, 40, 42, 45,
                                47, 50, 52, 54, 56, 58, 60, 62,
                                64, 66, 67, 69, 71, 72, 74, 75,
                                77, 78, 80, 81, 83, 84, 85, 87,
                                88, 89, 91, 92, 93, 95, 96, 97,
                                98, 100, 101, 102, 103, 105, 106, 107,
                               109, 110, 111, 111, 111, 111, 111, 111,
};

/* ENDE */
```

Abbildung 19: Listing des Leuchtfächers „em80blnk.h“

Der Header em80blnk.h wird ins Hauptprogramm eingebunden. Mit dessen Daten werden die einzelnen Bits des zuvor beschriebenen page_buf skaliert maskiert.

Dazu werden die Bytes des em80scrn-Leuchtschirms nicht einfach nur kopiert sondern mit mit einer variablen Maske undiert und danach in den page_buf abgelegt. Die Maske beginnt mit 0x00 und wird dann nach und nach Bitweise aufgefüllt. Nach und nach erscheint so der Leuchtfächer auf dem durch em80scrn begrenzten Bereich. Sobald die Maske mit 0xFF gefüllt ist, wird em80scrn lediglich nach page_buf kopiert.

Zur Weiterschiebung der Maske dient die em80blnk-Tabelle. Immer, wenn ein Spaltenzähler den nächsten Zahlenwert erreicht hat, wird ein Weißbit in die Maske geschoben und es wird der nächste Zahlenwert aus der em80blnk-Tabelle gelesen.

Der Index dieses nächsten Zahlenwertes ist jedoch abhängig von der Skalierung. Kleine Skalierungen lassen den Index nur langsam steigen. Große Skalierungen bewirken große Sprünge und somit starke Stauchungen des Fächers.

Die MIN-Makros begrenzen den em80blnk-Zugriff auf die Höchstwerte der Tabelle. Weiter kann der Fächer nicht gestaucht werden.

Diese Prozedur ermöglicht also das skalierte Einblenden gewölbter Leuchtfächer in den Leuchtschirmbereich und stellt den Kern der gesamten Software dar.

```
// page_buf mit schirmbild füllen und fächereinblendung mit Stauchung in rows-Richtung
void scrn_2_pagebuf(unsigned char page_num,unsigned int scal)
{                                                       // flash-speicher-pointer setzen
        const unsigned char* scrn = em80scrn + page_num;
        unsigned char page_col = 0;                     // von links nach rechts scannen
        unsigned char mask = 0x00;                      // alle pixel dunkel
        unsigned char col_cnt;                          // inhalt von blnk-Tabelle
        unsigned int blnk_ind;                          // index in blnk-Tabelle

        blnk_ind = (scal += _BV(SFT_SCAL)) * (page_num << 3);
        blnk_ind = MIN((sizeof(em80blnk) << SFT_SCAL) - 1,blnk_ind);
        col_cnt = pgm_read_byte(em80blnk + (blnk_ind >> SFT_SCAL));

        for(;page_col < sizeof(page_buf);page_col++,scrn += 4) {
                if(mask == 0xFF) {                      // dann nur noch kopieren
                        page_buf[page_col] = pgm_read_byte(scrn);
                        continue;
                }                                       // sonst zuvor maskieren
                page_buf[page_col] = (pgm_read_byte(scrn) & mask);

                while(page_col > col_cnt) {             // mask ganz abarbeiten
                        if((mask = (0x80 | (mask >> 1))) == 0xFF) break;
                        blnk_ind += scal;               // em80blnk-zugriff clip + scal
                        blnk_ind = MIN((sizeof(em80blnk) << SFT_SCAL) - 1,blnk_ind);
                        col_cnt = pgm_read_byte(em80blnk + (blnk_ind >> SFT_SCAL));
                }
        }
}
```

Abbildung 20: Erstellung des Leuchtschirms mit skaliertem Fächer

Scatterbereich

Links und rechts vom Elektrodensystem treten Elektronen aus. Auch deren Abbild wird in gleicher Weise am Leuchtschirm gebogen wie das normale Fächerbild.

Man kann die em80blnk-Tabelle auch für die Erstellung des Scatter-Bereiches nutzen. Allerdings muss die Stauchung hier nicht in der Richtung der Reihen sondern in Richtung der Spalten stattfinden.

Auch hier wird wieder eine Maske verwendet, die jedoch mit einem weißen Byte beginnt und pixelweise zunehmend geschwärzt wird, wenn der Schleifenzähler das jeweilige em80blnk-Limit erreicht.

Sobald das Limit erreicht ist, wird ein weiteres schwarzes Bit in die Maske geschoben und der Index eines neuen Limits bestimmt.

Das Scattering wird dem Display-Buffer hinzugesetzt, wozu em80scrn mit dem Scatter undiert wird und dann das Ergebnis mit dem pagebuf oderiert wird.

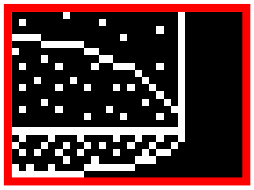
Daraus ergibt sich folgende Prozedur, die der Prozedur auf der vorigen Seite sehr ähnlich ist:

```
// page_buf mit scatter ergänzen, stauchung in cols-richtung
void sctr_2_pagebuf(unsigned char page_num,unsigned int scal)
{                                                   // flash-speicher-pointer setzen
        const unsigned char* scrn = em80scrn + page_num;
        const unsigned char* blnk = em80blnk + (page_num << 3);
        unsigned char page_col = 0;                 // von links nach rechts scannen
        unsigned char mask = 0xFF;                  // alle pixel hell
        unsigned int dda_limit = pgm_read_byte(blnk) << SFT_SCAL;
        unsigned int dda_delta = 0;

        scal += _BV(SFT_SCAL);                      // offset skalierung
                                                    // scanne den page_buf
        for(;page_col < sizeof(page_buf);page_col++,scrn += 4) {
                // scrn wird maskiert und mit page_buf oderiert
                page_buf[page_col] |= (pgm_read_byte(scrn) & mask);

                dda_delta += scal;                  // dda weiterstellen
                while(dda_delta > dda_limit) {      // mask ganz abarbeiten
                        // bitmaske eingrenzen und neuen dda-endwert holen
                        if(!(mask >>= 1)) return;   // schwarze maske: raus hier!
                        dda_limit = pgm_read_byte(++blnk) << SFT_SCAL;
                }
        }
}
```

Abbildung 21: Einblenden des skalierten Scatterings

Elektrodensystem

Das Elektrodensystem im em80scrn besteht nur aus dessen Umriss. Um hier eine Verschönerung einzubauen, wird folgende kleine Paint-Grafik mit 32 Spalten und einer beliebigen Anzahl von Zeilen (hier 23) als em80deko.bmp erstellt und mit IMPBMP zu em80deko.h konvertiert:



```
PROGMEM const unsigned char em80deko[92] = {
                                0x01, 0x00, 0x01, 0x00,
                                0x40, 0x08, 0x01, 0x00,
                                0x00, 0x00, 0x09, 0x00,
                                0xF0, 0x01, 0x01, 0x00,
                                0x0F, 0xC0, 0x01, 0x00,
                                0x80, 0x30, 0x01, 0x00,
                                ................
                                ................
                                0x00, 0x00, 0x01, 0x00,
                                0xFF, 0xFF, 0xFF, 0x00,
                                0x44, 0x42, 0x49, 0x00,
                                0x89, 0x28, 0x92, 0x00,
                                0x52, 0x42, 0x2C, 0x00,
                                0x01, 0x10, 0x70, 0x00,
                                0xA4, 0x3F, 0x80, 0x00,
                                0xFF, 0xC0, 0x00, 0x00,
};

/* ENDE */
```

Abbildung 22: Listing des Elektrodensystems „em80deko.h“

Das Elektrodensystem wird während der Aufheizzeit alleine dargestellt bzw. nach der Aufheizung mit dem fertigen page_buf oderiert:

```
// page_buf mit dekos aufhübschen
void deko_2_pagebuf(unsigned char page_num,BOOL heizung)
{
        const unsigned char* deko = em80deko + page_num;
        unsigned char page_col;

        if(heizung) {                                   // ausschleilich elektrodensystem darstellen
                for(page_col = 0;page_col < sizeof(em80deko) / 4;page_col++,deko += 4)
                        page_buf[page_col] = pgm_read_byte(deko);       // deko setzen
                for(;page_col < sizeof(page_buf);page_col++)
                        page_buf[page_col] = 0;                         // rest schwarz
        }                                       // ansonsten elektrodensystem nachträglich einblenden
        else for(page_col = 0;page_col < sizeof(em80deko) / 4;page_col++,deko += 4)
                page_buf[page_col] |= pgm_read_byte(deko);              // deko oderieren
}
```

Abbildung 23: Einblenden des Elektrodensystems

Spiegelung

Im Controller werden lediglich vier der acht Display-Pages mit den zuvor beschriebenen Verfahren erstellt. Die vier fehlenden Seiten entstehen durch Spiegelung. Dazu wird jedes Byte des jeweiligen page_buf mit einer Tabelle gespiegelt und danach der geänderte page_buf ans Display gesendet..

```
// das em80-Bild ist spiegelsymmetrisch. Um ein Byte zügig zu spiegeln
// wird eine tabelle benutzt.

PROGMEM const unsigned char reflect_tab[256] = {
                0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,        /* 0x00 - 0x07 */
                0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,        /* 0x08 - 0x0F */
                0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,        /* 0x10 - 0x17 */
                0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,        /* 0x18 - 0x1F */
                0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,        /* 0x20 - 0x27 */
                0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,        /* 0x28 - 0x2F */
                0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,        /* 0x30 - 0x37 */
                ........................
                ........................
                ........................
                0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,    /* 0xB8 - 0xBF */
                0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,        /* 0xC0 - 0xC7 */
                0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,        /* 0xC8 - 0xCF */
                0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,    /* 0xD0 - 0xD7 */
                0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,    /* 0xD8 - 0xDF */
                0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,        /* 0xE0 - 0xE7 */
                0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,        /* 0xE8 - 0xEF */
                0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,        /* 0xF0 - 0xF7 */
                0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF         /* 0xF8 - 0xFF */
};


//********* spiegeln und dann senden
void refl_send_pagebuf(void)                                    // die eigentliche spiegelung
{
        unsigned char ind;

        for(ind = 0;ind < sizeof(page_buf);ind++)
                page_buf[ind] = pgm_read_byte(reflect_tab + page_buf[ind]);
                                                                // gleich senden
        send_buffer(FALSE,page_buf,sizeof(page_buf));
        SEND_STOP;
}
```

Abbildung 24: Spiegelung

ADC und Röhrenkennlinie

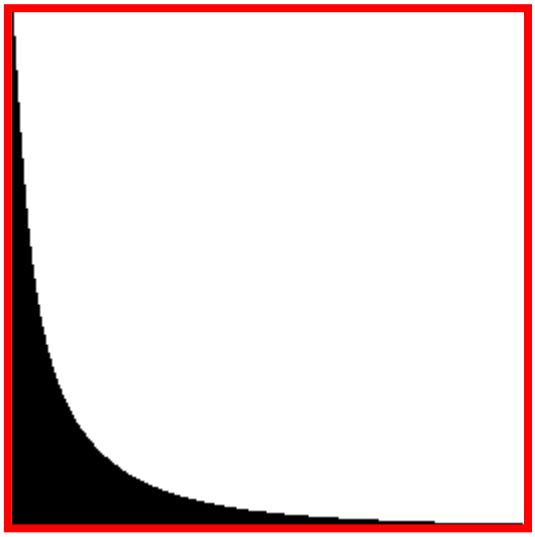
Der ADC wird mit drei einfachen Makros einmalig initialisiert, gestartet und abgefragt. Der Wandler arbeitet parallel zur Software und speichert sein Wandlungsergebnis, bis es von der Software abgefragt wird.

```
// ******* AD-Wandler **************
#define ADC_INI {                                                       \
        PORTB &= ~ANA_IN;                       /* analog-eingang */    \
        DIDR0 = ANA_IN;                         /* 16 MHz / 128 = 125kHz */ \
        ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);      \
}

#define ADC_START       ADCSRA |= _BV(ADSC)         // neue wandlung starten

#define ADC_GET(v) {                                                    \
        while(ADCSRA & _BV(ADSC));              /* conversion complete? */ \
        (v) = ADC;                              /* wert abholen */      \
}

#include "em80func.h"                           // zur erzeugung der kennlinie
```

Abbildung 25: ADC-Treiber

Die relevantesten 8 Bit des vom ADC gelesenen Wertes werden als Index in eine Kennlinientabelle verwendet.

Die Kennlinie wird mit Paint mit 256 x 256 Pixeln gezeichnet und berücksichtigt die Röhren- und die Skalierungsunlinearität.



Die Anzahlen der zeilenweisen schwarzen (dunklen) Pixel dieser Kurve wird mit CNTBMP zu em80func.h gewandelt und ins Hauptprogramm eingebunden.

```
PROGMEM const unsigned char em80func[256] = {
                                  1,   1,   1,   1,   1,   1,   1,   1,
                                  1,   1,   1,   1,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   3,   3,   3,
                                  3,   3,   3,   3,   3,   3,   3,   3,
                                  3,   3,   3,   3,   3,   4,   4,   4,
                                  4,   4,   4,   4,   4,   4,   4,   4,
                                  4,   4,   4,   4,   5,   5,   5,   5,
                                  5,   5,   5,   5,   5,   5,   5,   5,
                                  5,   6,   6,   6,   6,   6,   6,   6,
                                  6,   6,   6,   6,   6,   6,   7,   7,
                                  7,   7,   7,   7,   7,   7,   7,   7,
                                  7,   7,   8,   8,   8,   8,   8,   8,
                                  8,   8,   8,   8,   9,   9,   9,   9,
                                  9,   9,   9,   9,   9,  10,  10,  10,
                                 10,  10,  10,  10,  10,  10,  11,  11,
                                 11,  11,  11,  11,  11,  12,  12,  12,
                                 12,  12,  12,  12,  13,  13,  13,  13,
                                 13,  13,  14,  14,  14,  14,  14,  14,
                                 15,  15,  15,  15,  15,  16,  16,  16,
                                 16,  17,  17,  17,  17,  17,  18,  18,
                                 18,  18,  19,  19,  19,  20,  20,  20,
                                 20,  21,  21,  21,  22,  22,  22,  23,
                                 23,  23,  24,  24,  25,  25,  25,  26,
                                 26,  27,  27,  28,  28,  29,  29,  30,
                                 30,  31,  31,  32,  32,  33,  34,  34,
                                 35,  36,  37,  37,  38,  39,  40,  41,
                                 41,  42,  43,  44,  45,  46,  48,  49,
                                 50,  51,  53,  54,  55,  57,  58,  60,
                                 62,  64,  66,  68,  70,  73,  75,  78,
                                 81,  84,  88,  92,  96, 101, 106, 112,
                                118, 127, 136, 148, 163, 183, 211, 255,
};

/* ENDE */
```

Abbildung 26: Listing der Kennlinie „em80func.h"

Zur Benutzung der 8-Bit-Tabelle wird der 10 Bit-ADC entsprechend skaliert:

```
                ADC_GET(adc_val);                       // hole AD-Wandlungsergebnis ab
                ADC_START;                              // starte die AD-Wandler-Hardware
                                                        // röhrenkennlinie einarbeiten
                adc_val = (pgm_read_byte(em80func + (adc_val >> 2)) << 2) | (adc_val & 0x03);
```

Heizungsemulation und Helligkeitssteuerung

Die Hauptschleife des Programms wird rund 150 mal in der Sekunde durchlaufen. Zur Emulation der Aufheizung wird ein „heiz_timer“ inkrementiert. Bis zum Erreichen eines Schwellwertes wird ausschließlich das Elektrodensystem dargestellt. Der Leuchtschirm bleibt dunkel.

Nach Überschreitung der Schwelle konkurrieren heiz_timer und der ADC. Der kleinere Wert von beiden steuert die Skalierung des Leuchtfächers und des Scatter-Bereichs. Da heiz_timer weiter inkrementiert wird, ergibt sich eine scheinbare Zuname der Ablenkungsverstärkung, bis schließlich der ADC allein die Steuerung der beweglichen Grafikteile übernommen hat.

Weiterhin wird die Helligkeit aus der Größe des Leuchtfächers abgeleitet. Je größer der Fächer, desto geringer dessen Helligkeit. Dies entspricht dem Verhalten einer realen Röhre.

Abbildung 27: Heiz- und Helligkeitssteuerung

```
int main()
{
	unsigned int adc_val;
	signed int heiz_time = -1000;		// rund 5 sekunden heiztime emulieren
	unsigned char light;

	PORTB = 0xFF;				// pullups ein: keine wackelnden ports
	ADC_INI;				// ADC konfigurieren
	USI_INI;				// USI konfigurieren

	FOREVER {				// ********* die ewige schleife des hauptprogramms
		SEND_START;			// init oled, setze helligkeit
		send_buffer(TRUE,(unsigned char*)oled_init,sizeof(oled_init));

		ADC_GET(adc_val);		// hole AD-Wandlungsergebnis ab
		ADC_START;			// starte die AD-Wandler-Hardware
						// röhrenkennlinie einarbeiten
		adc_val = (pgm_read_byte(em80func + (adc_val >> 2)) << 2) | (adc_val & 0x03);

		// hier wird das anheizen simuliert. ein paar Sekunden sieht man nur
		// das elektrodensytem. Dann erscheint der Leuchtschirm und und erreicht
		// nach wenigen sekunden seine maximale Helligkeit und die interne triode
		// erreicht ihre maximal verstärkung.
		// besonders deutlich ist der effekt bei gitterspannungen nahe 0V.
		// bei hohen negativen gitterspannungen beginnt die röhre mit einem
		// großen leuchtfächer ohne ihre helligkeit oder verstärkung
		// noch weiter steigern zu können.
		// hier wird die anheizzeit trickreich mit dem ad-Wandler verbunden

		if(++heiz_time < 0) light = 0xFF;	// elektrodensystem volle helligkeit
		else {					// langsam kommt der leuchtschirm
			if(heiz_time > 4711) heiz_time--;	// obergrenze irgendwo festhalten
			adc_val = MIN(adc_val,heiz_time);// aussteuerung steigt langsam
			light = adc_val >> 2;			// wandle 10 adc-bit ins helligkeits-byte
		}
		send_buffer(FALSE,&light,1);	// helligkeit senden
		SEND_STOP;
		.....
	}
}
```

Hauptprogramm

Abbildung 28: Listing des Hauptprogramms „mageye.c“

```
#include <avr/io.h>
#include <avr/pgmspace.h>                       // zugriff auf flash-speicher

FUSES =
{
        .low = (FUSE_CKSEL1 & FUSE_CKSEL2 & FUSE_CKSEL3 & FUSE_SUT0),
        .high = FUSE_RSTDISBL,
        .extended = EFUSE_DEFAULT,
};


#define FOREVER         while(1)                // alte gewohnheiten....
#define FALSE           false
#define TRUE            true
#define MIN(a,b)        (a) < (b) ? (a) : (b)
#ifndef __cplusplus
        typedef enum {FALSE,TRUE} BOOL;
#endif

/****************** nun gehts los ********************************************/

// ********** portbelegungen
#define USI_CLK         _BV(PB2)                // Clock-Ausgang
#define USI_DATA        _BV(PB1)                // Daten-Ausgang
#define ANA_IN          _BV(ADC0D)              // = PB5


#include "usi_drv.h"                            // display-treiber per usi-hardware

// ********** oled befehle
PROGMEM const unsigned char oled_init[] = {
        0x3C << 1,                              /* adresse */
        0x00,                                   /* control byte. Von nun ab nur Commands */
        0x8D, 0x14,                             /* set charge pump, turn on charge pump */
        0xAF,                                   /* display on */
        0x81                                    /* helligkeit im nächsten byte */
};

PROGMEM const unsigned char oled_page0[] = {
        0x3C << 1,                              /* adresse */
        0x80, 0xB0,                             /* control byte. Kommando: page 0 */
        0x80, 0x00,                             /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                             /* control byte. Kommando: high-spalte 0 */
        0x40                                    /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page1[] = {
        0x3C << 1,                              /* adresse */
        0x80, 0xB1,                             /* control byte. Kommando: page 1 */
        0x80, 0x00,                             /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                             /* control byte. Kommando: high-spalte 0 */
        0x40                                    /* control byte. Von nun ab nur Daten */
};
```

```
PROGMEM const unsigned char oled_page2[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB2,                     /* control byte. Kommando: page 2 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page3[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB3,                     /* control byte. Kommando: page 3 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page4[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB4,                     /* control byte. Kommando: page 4 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page5[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB5,                     /* control byte. Kommando: page 5 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page6[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB6,                     /* control byte. Kommando: page 6 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};

PROGMEM const unsigned char oled_page7[] = {
        0x3C << 1,                      /* adresse */
        0x80, 0xB7,                     /* control byte. Kommando: page 7 */
        0x80, 0x00,                     /* control byte. Kommando: low-spalte 0 */
        0x80, 0x10,                     /* control byte. Kommando: high-spalte 0 */
        0x40                            /* control byte. Von nun ab nur Daten */
};
```

```
#define SFT_SCAL        6                       // hoher wert zur sprung-redizierung

unsigned char page_buf[128];                    // arbeitsspeicher für grafik und oled-ausgabe

#include "em80scrn.h"                           // von paintbrush mit "impbmp" (DIY) konvertierte bmp
                                                // x = 32 bit (=4 Byte), rows = 128: Leuchtschirm
#include "em80blnk.h"                           // von paintbrush mit "cntbmp" (DIY) "gezählte" bmp
                                                // rows = 64: Fächerrand-Zahlen (0...127)

// page_buf mit schirmbild füllen und fächereinblendung mit Stauchung in rows-Richtung
void scrn_2_pagebuf(unsigned char page_num,unsigned int scal)
{                                               // flash-speicher-pointer setzen
        const unsigned char* scrn = em80scrn + page_num;
        unsigned char page_col = 0;             // von links nach rechts scannen
        unsigned char mask = 0x00;              // alle pixel dunkel
        unsigned char col_cnt;                  // inhalt von blnk-Tabelle
        unsigned int blnk_ind;                  // index in blnk-Tabelle

        blnk_ind = (scal += _BV(SFT_SCAL)) * (page_num << 3);
        blnk_ind = MIN((sizeof(em80blnk) << SFT_SCAL) - 1,blnk_ind);
        col_cnt = pgm_read_byte(em80blnk + (blnk_ind >> SFT_SCAL));

        for(;page_col < sizeof(page_buf);page_col++,scrn += 4) {
                if(mask == 0xFF) {              // dann nur noch kopieren
                        page_buf[page_col] = pgm_read_byte(scrn);
                        continue;
                }                               // sonst zuvor maskieren
                page_buf[page_col] = (pgm_read_byte(scrn) & mask);

                while(page_col > col_cnt) {     // mask ganz abarbeiten
                        if((mask = (0x80 | (mask >> 1))) == 0xFF) break;

                        blnk_ind += scal;       // em80blnk-zugriff clip + scal
                        blnk_ind = MIN((sizeof(em80blnk) << SFT_SCAL) - 1,blnk_ind);
                        col_cnt = pgm_read_byte(em80blnk + (blnk_ind >> SFT_SCAL));
                }
        }
}
```

```
// page_buf mit scatter ergänzen, stauchung in cols-richtung
void sctr_2_pagebuf(unsigned char page_num,unsigned int scal)
{                                                       // flash-speicher-pointer setzen
        const unsigned char* scrn = em80scrn + page_num;
        const unsigned char* blnk = em80blnk + (page_num << 3);
        unsigned char page_col = 0;                     // von links nach rechts scannen
        unsigned char mask = 0xFF;                      // alle pixel hell
        unsigned int dda_limit = pgm_read_byte(blnk) << SFT_SCAL;
        unsigned int dda_delta = 0;

        scal += _BV(SFT_SCAL);                          // offset skalierung
                                                        // scanne den page_buf
        for(;page_col < sizeof(page_buf);page_col++,scrn += 4) {
                // scrn wird maskiert und mit page_buf oderiert
                page_buf[page_col] |= (pgm_read_byte(scrn) & mask);

                dda_delta += scal;                      // dda weiterstellen
                while(dda_delta > dda_limit) {          // mask ganz abarbeiten
                        // bitmaske eingrenzen und neuen dda-endwert holen
                        if(!(mask >>= 1)) return;   // schwarze maske: raus hier!
                        dda_limit = pgm_read_byte(++blnk) << SFT_SCAL;
                }
        }
}

#include "em80deko.h"                                   // von paintbrush mit "impbmp" (DIY) konvertierte bmp
                                                        // x = 32 bit (=4 Byte), rows = ?: elektrodensystem
// page_buf mit dekos aufhübschen
void deko_2_pagebuf(unsigned char page_num,BOOL heizung)
{
        const unsigned char* deko = em80deko + page_num;
        unsigned char page_col;

        if(heizung) {                                   // ausschleilich elektrodensystem darstellen
                for(page_col = 0;page_col < sizeof(em80deko) / 4;page_col++,deko += 4)
                        page_buf[page_col] = pgm_read_byte(deko);        // deko setzen
                for(;page_col < sizeof(page_buf);page_col++)
                        page_buf[page_col] = 0;     // rest schwarz
        }                                               // ansonsten elektrodensystem nachträglich einblenden
        else for(page_col = 0;page_col < sizeof(em80deko) / 4;page_col++,deko += 4)
                page_buf[page_col] |= pgm_read_byte(deko);               // deko oderieren
}

// ***** sämtliche grafikverarbeitung machen und dann senden
void make_send_pagebuf(unsigned char page_num,unsigned int scal,BOOL heizung)
{
        if(!heizung) {
          scrn_2_pagebuf(page_num,scal);            // screen mit skaliertem fächer
           sctr_2_pagebuf(page_num,scal);           // scatter seitlich am elektrodensystem
        }
        deko_2_pagebuf(page_num,heizung);               // elektrodensystem
        send_buffer(FALSE,page_buf,sizeof(page_buf));   // rausssenden
        SEND_STOP;
}
```

```
// das em80-Bild ist spiegelsymmetrisch. Um ein Byte zügig zu spiegeln
// wird eine tabelle benutzt.

PROGMEM const unsigned char reflect_tab[256] = {
                0x00,0x80,0x40,0xC0,0x20,0xA0,0x60,0xE0,        /* 0x00 - 0x07 */
                0x10,0x90,0x50,0xD0,0x30,0xB0,0x70,0xF0,        /* 0x08 - 0x0F */
                0x08,0x88,0x48,0xC8,0x28,0xA8,0x68,0xE8,        /* 0x10 - 0x17 */
                0x18,0x98,0x58,0xD8,0x38,0xB8,0x78,0xF8,        /* 0x18 - 0x1F */
                0x04,0x84,0x44,0xC4,0x24,0xA4,0x64,0xE4,        /* 0x20 - 0x27 */
                0x14,0x94,0x54,0xD4,0x34,0xB4,0x74,0xF4,        /* 0x28 - 0x2F */
                0x0C,0x8C,0x4C,0xCC,0x2C,0xAC,0x6C,0xEC,        /* 0x30 - 0x37 */
                0x1C,0x9C,0x5C,0xDC,0x3C,0xBC,0x7C,0xFC,        /* 0x38 - 0x3F */
                0x02,0x82,0x42,0xC2,0x22,0xA2,0x62,0xE2,        /* 0x40 - 0x47 */
                0x12,0x92,0x52,0xD2,0x32,0xB2,0x72,0xF2,        /* 0x48 - 0x4F */
                0x0A,0x8A,0x4A,0xCA,0x2A,0xAA,0x6A,0xEA,        /* 0x50 - 0x57 */
                0x1A,0x9A,0x5A,0xDA,0x3A,0xBA,0x7A,0xFA,        /* 0x58 - 0x5F */
                0x06,0x86,0x46,0xC6,0x26,0xA6,0x66,0xE6,        /* 0x60 - 0x67 */
                0x16,0x96,0x56,0xD6,0x36,0xB6,0x76,0xF6,        /* 0x68 - 0x6F */
                0x0E,0x8E,0x4E,0xCE,0x2E,0xAE,0x6E,0xEE,        /* 0x70 - 0x77 */
                0x1E,0x9E,0x5E,0xDE,0x3E,0xBE,0x7E,0xFE,        /* 0x78 - 0x7F */
                0x01,0x81,0x41,0xC1,0x21,0xA1,0x61,0xE1,        /* 0x80 - 0x87 */
                0x11,0x91,0x51,0xD1,0x31,0xB1,0x71,0xF1,        /* 0x88 - 0x8F */
                0x09,0x89,0x49,0xC9,0x29,0xA9,0x69,0xE9,        /* 0x90 - 0x97 */
                0x19,0x99,0x59,0xD9,0x39,0xB9,0x79,0xF9,        /* 0x98 - 0x9F */
                0x05,0x85,0x45,0xC5,0x25,0xA5,0x65,0xE5,        /* 0xA0 - 0xA7 */
                0x15,0x95,0x55,0xD5,0x35,0xB5,0x75,0xF5,        /* 0xA8 - 0xAF */
                0x0D,0x8D,0x4D,0xCD,0x2D,0xAD,0x6D,0xED,        /* 0xB0 - 0xB7 */
                0x1D,0x9D,0x5D,0xDD,0x3D,0xBD,0x7D,0xFD,        /* 0xB8 - 0xBF */
                0x03,0x83,0x43,0xC3,0x23,0xA3,0x63,0xE3,        /* 0xC0 - 0xC7 */
                0x13,0x93,0x53,0xD3,0x33,0xB3,0x73,0xF3,        /* 0xC8 - 0xCF */
                0x0B,0x8B,0x4B,0xCB,0x2B,0xAB,0x6B,0xEB,        /* 0xD0 - 0xD7 */
                0x1B,0x9B,0x5B,0xDB,0x3B,0xBB,0x7B,0xFB,        /* 0xD8 - 0xDF */
                0x07,0x87,0x47,0xC7,0x27,0xA7,0x67,0xE7,        /* 0xE0 - 0xE7 */
                0x17,0x97,0x57,0xD7,0x37,0xB7,0x77,0xF7,        /* 0xE8 - 0xEF */
                0x0F,0x8F,0x4F,0xCF,0x2F,0xAF,0x6F,0xEF,        /* 0xF0 - 0xF7 */
                0x1F,0x9F,0x5F,0xDF,0x3F,0xBF,0x7F,0xFF         /* 0xF8 - 0xFF */
};

//********** spiegeln und dann senden
void refl_send_pagebuf(void)                            // die eigentliche spiegelung
{
        unsigned char ind;

        for(ind = 0;ind < sizeof(page_buf);ind++)
                page_buf[ind] = pgm_read_byte(reflect_tab + page_buf[ind]);
                                                        // gleich senden
        send_buffer(FALSE,page_buf,sizeof(page_buf));
        SEND_STOP;
}
```

```
// ******* AD-Wandler **************
#define ADC_INI {                                                       \
        PORTB &= ~ANA_IN;                       /* analog-eingang */    \
        DIDR0 = ANA_IN;                         /* 16 MHz / 128 = 125kHz */ \
        ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);      \
}

#define ADC_START       ADCSRA |= _BV(ADSC)             // neue wandlung starten

#define ADC_GET(v) {
        \
        while(ADCSRA & _BV(ADSC));              /* conversion complete? */  \
        (v) = ADC;                              /* wert abholen */          \
}

#include "em80func.h"                           // zur erzeugung der kennlinie


// ********* die ewige schleife des hauptprogramms
int main()
{
        unsigned int adc_val;
        signed int heiz_time = -1000;           // rund 5 sekunden heiztime emulieren
        unsigned char light;

        PORTB = 0xFF;                           // pullups ein: keine wackelnden ports
        ADC_INI;                                // ADC konfigurieren
        USI_INI;                                // USI konfigurieren

        FOREVER {
                SEND_START;                     // init oled, setze helligkeit
                send_buffer(TRUE,(unsigned char*)oled_init,sizeof(oled_init));

                ADC_GET(adc_val);               // hole AD-Wandlungsergebnis ab
                ADC_START;                      // starte die AD-Wandler-Hardware
                                                // röhrenkennlinie einarbeiten
                adc_val = (pgm_read_byte(em80func + (adc_val >> 2)) << 2) | (adc_val & 0x03);
```

```
                // hier wird das anheizen simuliert. ein paar Sekunden sieht man nur
                // das elektrodensytem. Dann erscheint der Leuchtschirm und und erreicht
                // nach wenigen sekunden seine maximale Helligkeit und die interne triode
                // erreicht ihre maximal verstärkung.
                // besonders deutlich ist der effekt bei gitterspannungen nahe 0V.
                // bei hohen negativen gitterspannungen beginnt die röhre mit einem
                // großen leuchtfächer ohne ihre helligkeit oder verstärkung
                // noch weiter steigern zu können.
                // hier wird die anheizzeit trickreich mit dem ad-Wandler verbunden

                if(++heiz_time < 0) light = 0xFF;   // elektrodensystem volle helligkeit
                else {                              // langsam kommt der leuchtschirm
                        if(heiz_time > 4711) heiz_time--;   // obergrenze irgendwo festhalten
                        adc_val = MIN(adc_val,heiz_time);// aussteuerung steigt langsam
                        light = adc_val >> 2;               // wandle 10 adc-bit ins helligkeits-byte
                }
                send_buffer(FALSE,&light,1);// helligkeit senden
                SEND_STOP;

                SEND_START;                         // wg. spiegelung: mein buffer 0 -> oled buffer 3
                send_buffer(TRUE,(unsigned char*)oled_page3,sizeof(oled_page3));
                make_send_pagebuf(0,adc_val,heiz_time < 0);

                SEND_START;                         // mein gespiegelter buffer 0 -> oled buffer 4
                send_buffer(TRUE,(unsigned char*)oled_page4,sizeof(oled_page4));
                refl_send_pagebuf();

                SEND_START;                         // wg. spiegelung: mein buffer 1 -> oled buffer 2
                send_buffer(TRUE,(unsigned char*)oled_page2,sizeof(oled_page2));
                make_send_pagebuf(1,adc_val,heiz_time < 0);

                SEND_START;                         // mein gespiegelter buffer 1 -> oled buffer 5
                send_buffer(TRUE,(unsigned char*)oled_page5,sizeof(oled_page5));
                refl_send_pagebuf();

                SEND_START;                         // wg. spiegelung: mein buffer 2 -> oled buffer 1
                send_buffer(TRUE,(unsigned char*)oled_page1,sizeof(oled_page1));
                make_send_pagebuf(2,adc_val,heiz_time < 0);

                SEND_START;                         // mein gespiegelter buffer 2 -> oled buffer 6
                send_buffer(TRUE,(unsigned char*)oled_page6,sizeof(oled_page6));
                refl_send_pagebuf();

                SEND_START;                         // wg. spiegelung: mein buffer 3 -> oled buffer 0
                send_buffer(TRUE,(unsigned char*)oled_page0,sizeof(oled_page0));
                make_send_pagebuf(3,adc_val,heiz_time < 0);

                SEND_START;                         // mein gespiegelter buffer 3 -> oled buffer 7
                send_buffer(TRUE,(unsigned char*)oled_page7,sizeof(oled_page7));
                refl_send_pagebuf();
        }
}

/* ENDE */
```

Anhang

Bekannte Probleme

Das Gate des FET im Analogteil ist extrem hochohmig beschaltet (Sperrströme der Siliziumdioden). Die Dioden und der FET müssen kapazitätsarm elektrostatisch geschirmt werden.

Es gibt erhebliche HF-Störungen durch den Datenbetrieb zwischen Controller und Display und durch das Display selbst. Controller und Display sollten also dicht nebeneinander angeordnet werden. Trotzdem ist eine Verwendung im Radio problematisch.

Zukunft

- Beschaffung eines formgünstigeren und grünen Displays.
- Platinenkonstruktion. Ggfls. nach dem Tinkertoy-Prinzip gestapelt.
- Aufbau.
- Emulation anderer Röhrentypen.

Alternativen

Die beiden Verfasser hatten 2017 im military-tubes-Forum einen Mini-Inverter zum Betrieb verdunkelter Magischer Augen entworfen.

Die Wirkung beruhte auf der Beobachtung, dass eine angelegte Gleichspannung deutlich weniger Helligkeit erbringt, als eine mit HF überlagerte Gleichspannung gleichen Spannungsmittelwertes.

Es ergaben sich nur geringe Verminderungen der (mittleren) Ablenkwinkel und somit einen Vorteil gegenüber dem bekannten Hochsetzen der Anodengleichspannung. Allerdings wird die Abbildungsschärfe etwas reduziert.

Die geringen Verminderungen des Ablenkwinkels sind subjektiv, da gerade der Leuchtfächer gerade dann besonders gut zu sehen ist, wenn die hohe Amplitude der Leuchtschirmspitzenspannung die Ablenkung reduziert.

Zusätzlich wurde spekuliert, dass der pulsierende Elektronenbeschuss den Leuchtschirm nach und nach regeneriert, denn die Röhren wurden während den Versuchen immer leuchtstärker. Was allerdings auch katodenseitige Ursachen gehabt haben kann.

Trotz des Bemühens zur Erzeugung einer möglichst sinusförmigen HF waren die Funkstörungen des Inverters erheblich. Die Anbringung von Schirmungen und Drosseln wurde nur wenig erprobt und waren auch nur bedingt sinnvoll, da besonders der unschirmbare Leuchtschirm HF abstrahlt.

IMPBMP

Dieses kleine ´c++´-Tool konvertiert monochrome MS-Paint BMP-Dateien in einen ATTiny `C´-Header. Die Abmessungen des Pixelbildes dürfen die unter COLS und ROWS definierten Werte nicht überschreiten. Das Programm wird z.B. in der Windows DOS-Box aufgerufen mit „IMPBMP em80scrn“ und meldet Fehler bzw. den Wandlungserfolg.

Abbildung 29: Listing „IMPBMP.CPP“

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

#define YES       1
#define NO        0
#define ERROR  -1

#define UINT     unsigned int
#define WORD   unsigned int
#define DWORDunsigned long
#define LONG    long

static struct {    /* bmfh */
   UINT    bfType;
   DWORD   bfSize;
   UINT    bfReserved1;
   UINT    bfReserved2;
   DWORD   bfOffBits;
} bmfh;


static struct {    /* bmih */
   DWORD   biSize;
   LONG    biWidth;
   LONG    biHeight;
   WORD    biPlanes;
   WORD    biBitCount;
   DWORD   biCompression;
   DWORD   biSizeImage;
   LONG    biXPelsPerMeter;
   LONG    biYPelsPerMeter;
   DWORD   biClrUsed;
   DWORD   biClrImportant;
} bmih;


#define ROWS    512
#define COLS     512


static int my_write(FILE *dest,char *buf)
{
          if(strlen(buf) != fwrite(buf,sizeof(char),strlen(buf),dest)) {
            printf("\007\n\nSchreibfehler! Abbruch\n\n\n\n");
            return(NO);
          }
          return(YES);
}
```

```
main(int argc,char *argv[])
{
        FILE *source, *dest;
        char buffer[60000L], hbuf[80];
        int col, row_len, rows, cols;
        unsigned char byte;


        clrscr();

        if(argc != 2) {
          printf("\007\n\nBenutzung:\n\n\n");
          printf("\"IMPBMP datei <Return>\"\n\n");
          printf("\"datei\" ist die Bitmap, aus der eine gleichnamige Header-Datei erzeugt wird...\n\n");
          printf("die Bitmap muss einfarbig sein, die Groesse ist egal.\n\n\n\n");
          goto err;
        }

        /* quelldatei öffnen */
        strcpy(buffer,argv[1]);
        strcat(buffer,".BMP");
        source = fopen(buffer,"rb");
        if(!source) {
          printf("\007\n\n...kann die Datei \"%s\" nicht finden!\n\n",buffer);
          printf("Bitte Verzeichnis beachten und Dateinamen genau pruefen\n\n\n\n");
          goto err;
        }

        /* zieldatei öffnen */
        strcpy(buffer,argv[1]);
        strcat(buffer,".H");
        dest = fopen(buffer,"rb");
        if(dest) {
          printf("\007\n\nDie Datei \"%s\" gibt es schon!\n\n",buffer);
          printf("...ich will sie nicht ueberschreiben\n\n\n\n");
          goto err;
        }
        fclose(dest);

        dest = fopen(buffer,"wb");
        if(!dest) {
          printf("\007\n\n...kann die Datei \"%s\" nicht erzeugen!\n\n",buffer);
          printf("Wahrscheinlich ist die Festplatte voll\n\n\n\n");
          goto err;
        }

        /* bitmap-header einlesen */
        if((fread(&bmfh,sizeof(char),sizeof(bmfh),source) != sizeof(bmfh))||(strncmp((char
*)&(bmfh.bfType),"BM",2))) {
          printf("\007\n\nFehler im Bitmap-Header! Abbruch\n\n\n\n");
          goto err;
        }

        /* info-header einlesen */
        if((fread(&bmih,sizeof(char),sizeof(bmih),source) != sizeof(bmih))||(bmih.biBitCount !=
1)||(bmih.biCompression)) {
```

```
            printf("\007\n\nFehler im Info-Header: keine monochrome Bitmap! Abbruch\n\n\n\n");
            goto err;
          }

          /* color-table überspringen */
          col = (size_t)(bmfh.bfOffBits) - sizeof(bmfh) - sizeof(bmih);
          if(col != fread(buffer,sizeof(char),col,source)) {
            printf("\007\n\nFehler in der Farbtabelle! Abbruch\n\n\n\n");
            goto err;
          }

          /* beginn der datei in C-notation schreiben: name, cols, rows */
          rows = min(ROWS,(int)(bmih.biHeight));
          cols = min(COLS,(int)(bmih.biWidth));
          sprintf(buffer,"PROGMEM const unsigned char %s[%d] = {",argv[1],cols * rows / 8);
          if(!my_write(dest,buffer)) goto err;

          /* nun alle daten auf einen schlag einlesen (32 Bit boundaries!) */
          bmih.biWidth /= 8;
          if((size_t)(bmih.biWidth) % 4)
            row_len = (size_t)(bmih.biWidth) + 4 - (size_t)(bmih.biWidth) % 4;
          else row_len = (size_t)(bmih.biWidth);
          fread(buffer,row_len,(size_t)(bmih.biHeight),source);

          for(--rows;rows >= 0;rows--) {
            sprintf(hbuf,"\r\n\t\t\t");
            if(!my_write(dest,hbuf)) goto err;
            for(col = 0;col < cols;col += 8) {
              byte = buffer[(col >> 3) + rows * row_len];
              sprintf(hbuf,"0x%.2X, ",byte);
              if(!my_write(dest,hbuf)) goto err;
            }
          }
          /* ende der Datei in C-Notation schreiben */
          sprintf(buffer,"\r\n};\r\n\n/* ENDE */");
          my_write(dest,buffer);

          printf("\n\nHeader-Datei erfolgreich erstellt\n\n\n\n");

err:      fclose(dest);
          fclose(source);
          return(0);
}


/* ENDE */
```

CNTBMP

Dieses kleine ´c++´-Tool konvertiert monochrome MS-Paint BMP-Dateien in ATTiny `C´-Header. Dazu zählt es in jeder Reihe die Anzahl der ungesetzten (schwarzen) Pixel und gibt diese Anzahl als Zahlenwert aus. Dies wird z.B, genutzt, um die Röhrenkennlinie mit Paint zu konstruieren.

Die Abmessungen des Pixelbildes dürfen die unter COLS und ROWS definierten Werte nicht überschreiten. Das Programm wird z.B. in der Windows DOS-Box aufgerufen mit „CNTBMP em80func“ und meldet Fehler bzw. den Wandlungserfolg.

Abbildung 30: Listing „CNTBMP.CPP"

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>

#define YES       1
#define NO        0
#define ERROR  -1

#define UINT     unsigned int
#define WORD   unsigned int
#define DWORDunsigned long
#define LONG    long

static struct {    /* bmfh */
   UINT    bfType;
   DWORD   bfSize;
   UINT    bfReserved1;
   UINT    bfReserved2;
   DWORD   bfOffBits;
} bmfh;


static struct {    /* bmih */
   DWORD   biSize;
   LONG    biWidth;
   LONG    biHeight;
   WORD    biPlanes;
   WORD    biBitCount;
   DWORD   biCompression;
   DWORD   biSizeImage;
   LONG    biXPelsPerMeter;
   LONG    biYPelsPerMeter;
   DWORD   biClrUsed;
   DWORD   biClrImportant;
} bmih;

#define ROWS    512
#define COLS    512
```

```
static int my_write(FILE *dest,char *buf)
{
        if(strlen(buf) != fwrite(buf,sizeof(char),strlen(buf),dest)) {
          printf("\007\n\nSchreibfehler! Abbruch\n\n\n\n");
          return(NO);
        }
        return(YES);
}

main(int argc,char *argv[])
{
        FILE *source, *dest;
        char buffer[60000L], hbuf[80];
        int col, row_len, rows, cols, pixel_cnt, bit;
        unsigned char byte;


        clrscr();

        if(argc != 2) {
          printf("\007\n\nBenutzung:\n\n\n");
          printf("\"CNTBMP datei <Return>\"\n\n");
          printf("\"datei\" ist die Bitmap, aus der eine gleichnamige Header-Datei erzeugt wird...\n\n");
          printf("die Bitmap muss einfarbig sein, die Groesse ist egal.\n\n\n\n");
          goto err;
        }

        /* quelldatei öffnen */
        strcpy(buffer,argv[1]);
        strcat(buffer,".BMP");
        source = fopen(buffer,"rb");
        if(!source) {
          printf("\007\n\n...kann die Datei \"%s\" nicht finden!\n\n",buffer);
          printf("Bitte Verzeichnis beachten und Dateinamen genau pruefen\n\n\n\n");
          goto err;
        }

        /* zieldatei öffnen */
        strcpy(buffer,argv[1]);
        strcat(buffer,".H");
        dest = fopen(buffer,"rb");
        if(dest) {
          printf("\007\n\nDie Datei \"%s\" gibt es schon!\n\n",buffer);
          printf("...ich will sie nicht ueberschreiben\n\n\n\n");
          goto err;
        }
        fclose(dest);

        dest = fopen(buffer,"wb");
        if(!dest) {
          printf("\007\n\n...kann die Datei \"%s\" nicht erzeugen!\n\n",buffer);
          printf("Wahrscheinlich ist die Festplatte voll\n\n\n\n");
          goto err;
        }
```

```
        /* bitmap-header einlesen */
        if((fread(&bmfh,sizeof(char),sizeof(bmfh),source) != sizeof(bmfh))||(strncmp((char
*)&(bmfh.bfType),"BM",2))) {
          printf("\007\n\nFehler im Bitmap-Header! Abbruch\n\n\n\n");
          goto err;
        }

        /* info-header einlesen */
        if((fread(&bmih,sizeof(char),sizeof(bmih),source) != sizeof(bmih))||(bmih.biBitCount !=
1)||(bmih.biCompression)) {
          printf("\007\n\nFehler im Info-Header: keine monochrome Bitmap! Abbruch\n\n\n\n");
          goto err;
        }

        /* color-table überspringen */
        col = (size_t)(bmfh.bfOffBits) - sizeof(bmfh) - sizeof(bmih);
        if(col != fread(buffer,sizeof(char),col,source)) {
          printf("\007\n\nFehler in der Farbtabelle! Abbruch\n\n\n\n");
          goto err;
        }

        /* beginn der datei in C-notation schreiben: name, cols, rows */
        rows = min(ROWS,(int)(bmih.biHeight));
        cols = min(COLS,(int)(bmih.biWidth));
        sprintf(buffer,"PROGMEM const unsigned char %s[%d] = {",argv[1],rows);
        if(!my_write(dest,buffer)) goto err;

        /* nun alle daten auf einen schlag einlesen (32 Bit boundaries!) */
        bmih.biWidth /= 8;
        if((size_t)(bmih.biWidth) % 4)
          row_len = (size_t)(bmih.biWidth) + 4 - (size_t)(bmih.biWidth) % 4;
        else row_len = (size_t)(bmih.biWidth);
        fread(buffer,row_len,(size_t)(bmih.biHeight),source);

        for(--rows;rows >= 0;rows--) {
          if((rows % 8) == 7) {
            sprintf(hbuf,"\r\n\t\t\t");
            if(!my_write(dest,hbuf)) goto err;
          }
          pixel_cnt = 0;
          for(col = 0;col < cols;col += 8) {
            byte = buffer[(col >> 3) + rows * row_len];
            for(bit = 7;bit >= 0;bit--) {
              if(!(byte & (1 << bit))) pixel_cnt++;
            }
          }
          sprintf(hbuf,"%3.d, ",pixel_cnt);
          if(!my_write(dest,hbuf)) goto err;
        }
        /* ende der Datei in C-Notation schreiben */
        sprintf(buffer,"\r\n};\r\n\n/* ENDE */");
        my_write(dest,buffer);

        printf("\n\nHeader-Datei erfolgreich erstellt\n\n\n\n");
err:    fclose(dest);
        fclose(source);
        return(0);
}
```