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 33: Listing des Leuchtschirmbildes „em80scrn.h“

Dieser Header wird in das Hauptprogramm eingebunden.

Der Controller des Displays erwartet seine Daten in folgender Organisation:

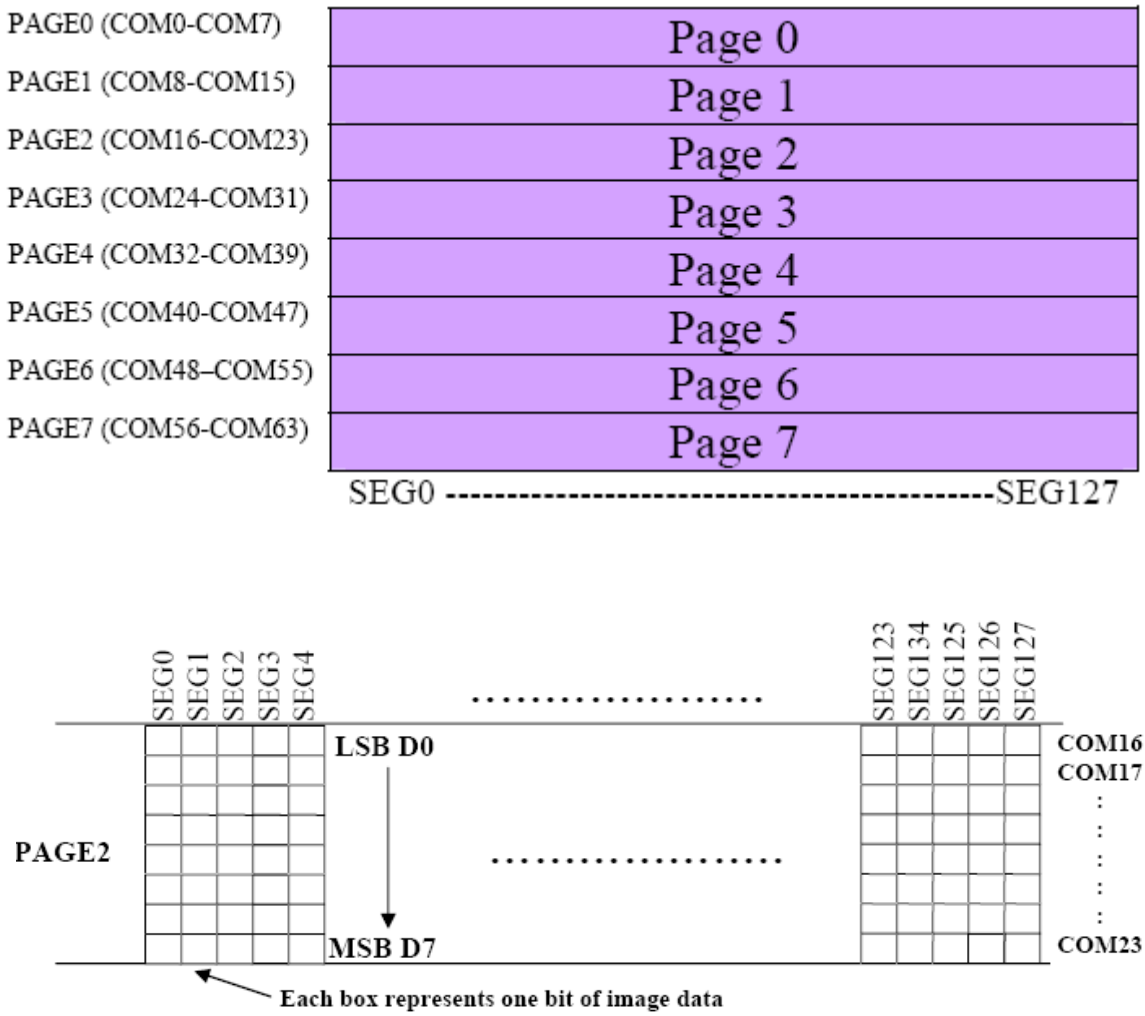


Abbildung 34: Datenformat des OLED-Displays

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 35: 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 36: 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.

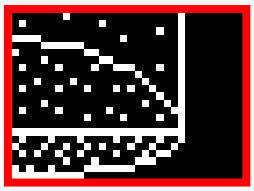
Daraus ergibt sich folgende Prozedur, die der Prozedur auf der vorigen Seite sehr ähnlich ist:

```
// 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 37: 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 38: 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 39: 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 40: 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 */ \
        ADMUX = _BV(REFS2) | _BV(REFS1) | _BV(ADLAR);       /* 2,56V referenz */    \
        ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);                  \
}

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

#include "em80func.h"                                   // zur erzeugung der kennlinie (257 indizes)

#define ADC_GET(v) {                                        /* 10bit-adc mit 8bit-kennlinie */  \
        while(ADCSRA & _BV(ADSC));                          /* conversion complete? */          \
        switch(ADCL & 0xC0) {                               /* interpolieren */                 \
                case 0x00:                                  /* 4 * ind */                       \
                        (v) = (pgm_read_byte(em80func + ADCH) << 2);                            \
                        break;                                                                  \
                case 0x40:                                  /* 3 * ind + 1 * next_ind */        \
                        (v) = pgm_read_byte(em80func + ADCH);                                   \
                        (v) += ((v) << 1);                                                      \
                        (v) += pgm_read_byte(em80func + ADCH + 1);                              \
                        break;                                                                  \
                case 0x80:                                  /* 2 * ind + 2 * next_ind */        \
                        (v) = pgm_read_byte(em80func + ADCH);                                   \
                        (v) += pgm_read_byte(em80func + ADCH + 1);                              \
                        (v) <<= 1;                                                              \
                        break;                                                                  \
                default:                                    /* 1 * ind + 3 * next_ind*/         \
                        (v) = pgm_read_byte(em80func + ADCH + 1);                               \
                        (v) += ((v) << 1);                                                      \
                        (v) += pgm_read_byte(em80func + ADCH);                                  \
                        break;                                                                  \
        }                                                                                       \
}
```

Abbildung 41: ADC-Treiber

Die relevantesten 8 Bit des vom ADC gelesenen Wertes werden als Index in eine 8-Bit-Kennlinientabelle verwendet. Die unteren beiden Bits des 10-Bit-ADC dienen der Interpolation der Tabelle.

Die Benutzng der Makros ist denkbar einfach:

```
                ADC_GET(adc_val);               // hole AD-Wandlungsergebnis ab
                ADC_START;                      // starte die AD-Wandler-Hardware
```

Die Kennlinie wird mit Paint mit 256 x 257 Pixeln gezeichnet und berücksichtigt die Röhren- und die Skalierungsunlinearität sowie die durchzuführende Inversion des ADC-Ergebnisses. Die 257 Zeilen ersparen Software-Grenzabfragen.

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[257] = {255,
                                206, 162, 140, 126, 115, 107, 100,  94,
                                 88,  83,  79,  75,  72,  69,  66,  63,
                                 61,  59,  57,  55,  53,  51,  49,  47,
                                 46,  45,  43,  42,  41,  40,  38,  37,
                                 36,  35,  34,  33,  32,  32,  31,  30,
                                 29,  28,  28,  27,  26,  25,  25,  24,
                                 24,  23,  23,  22,  22,  21,  21,  20,
                                 20,  19,  19,  18,  18,  17,  17,  16,
                                 16,  16,  16,  15,  15,  15,  14,  14,
                                 14,  13,  13,  13,  13,  12,  12,  12,
                                 12,  11,  11,  11,  11,  10,  10,  10,
                                 10,  10,   9,   9,   9,   9,   9,   9,
                                  8,   8,   8,   8,   8,   8,   8,   7,
                                  7,   7,   7,   7,   7,   7,   6,   6,
                                  6,   6,   6,   6,   6,   6,   6,   6,
                                  5,   5,   5,   5,   5,   5,   5,   5,
                                  5,   5,   5,   5,   4,   4,   4,   4,
                                  4,   4,   4,   4,   4,   4,   4,   4,
                                  4,   4,   4,   4,   3,   3,   3,   3,
                                  3,   3,   3,   3,   3,   3,   3,   3,
                                  3,   3,   3,   3,   3,   3,   3,   3,
                                  3,   3,   3,   3,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   2,
                                  2,   2,   2,   2,   2,   2,   2,   1,
                                  1,   1,   1,   1,   1,   1,   1,   1,
                                  1,   1,   1,   1,   1,   1,   1,   1,
                                  1,   1,   1,   1,   1,   1,   1,   1,
                                  1,   1,   1,   1,   1,   1,   1,   1,
                                  1,   1,   1,   1,   1,   1,   1,   1,
};

/* ENDE */
```

Abbildung 42: Listing der Kennlinie „em80func.h"

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.

```
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

		// 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;
		.....
	}
}
```

Abbildung 43: Heiz- und Helligkeitssteuerung

Hauptprogramm

Abbildung 44: 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 */ \
        ADMUX = _BV(REFS2) | _BV(REFS1) | _BV(ADLAR);       /* 2,56V referenz */    \
        ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0);                  \
}

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

#include "em80func.h"                                   // zur erzeugung der kennlinie (257 indizes)

#define ADC_GET(v) {                                        /* 10bit-adc mit 8bit-kennlinie */ \
        while(ADCSRA & _BV(ADSC));                          /* conversion complete? */ \
        switch(ADCL & 0xC0) {                               /* interpolieren */     \
                case 0x00:                                  /* 4 * ind */           \
                        (v) = (pgm_read_byte(em80func + ADCH) << 2);                \
                        break;                                                      \
                case 0x40:                                  /* 3 * ind + 1 * next_ind */ \
                        (v) = pgm_read_byte(em80func + ADCH);                       \
                        (v) += ((v) << 1);                                          \
                        (v) += pgm_read_byte(em80func + ADCH + 1);                  \
                        break;                                                      \
                case 0x80:                                  /* 2 * ind + 2 * next_ind */ \
                        (v) = pgm_read_byte(em80func + ADCH);                       \
                        (v) += pgm_read_byte(em80func + ADCH + 1);                  \
                        (v) <<= 1;                                                  \
                        break;                                                      \
                default:                                    /* 1 * ind + 3 * next_ind*/ \
                        (v) = pgm_read_byte(em80func + ADCH + 1);                   \
                        (v) += ((v) << 1);                                          \
                        (v) += pgm_read_byte(em80func + ADCH);                      \
                        break;                                                      \
        }                                                                           \
}

// ********** 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
```

```
                // 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

Daten

Höhe	55mm	(ohne Sockelstifte)
Breite	28mm	
Tiefe	21mm	
Fenster	21 x 13 mm	(1,5mm nach rechts versetzt)
Sockel	Noval	

Abbildung 45: Abmessungen

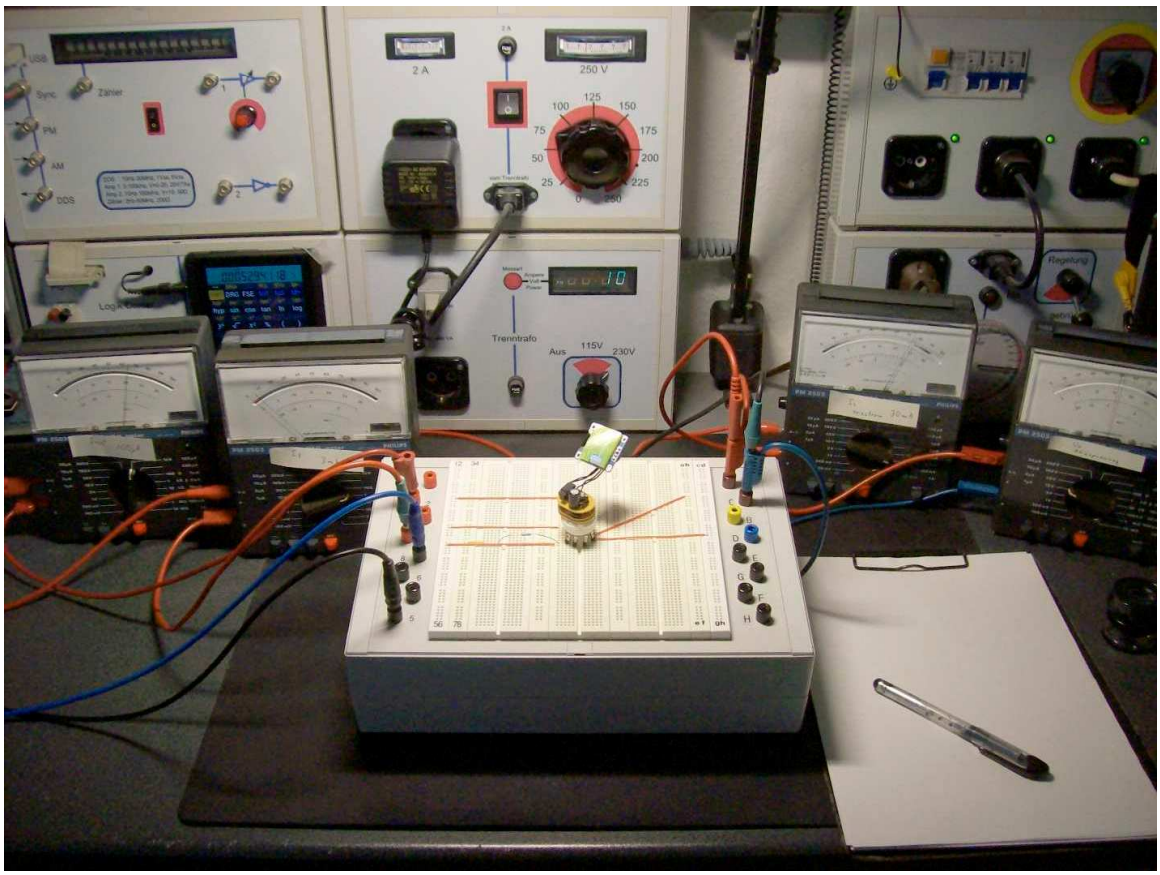


Abbildung 46: Messplatz

	Min.	Max.
U_{h_ac}	3,5 V~ (12 mA~)	13 V~ (26 mA~)
U_{h_dc}	5 V=	17 V=
V_{ao}		600 V
W_a		< 500 mW
V_{lo}		600 V
V_l	150 V	300 V
V_g	+1 V	-100 V
I_k		2 mA
R_g		10 Meg
V_{kf}		1 kV
R_{kf}		20 k

Abbildung 47: Grenzdaten

Vb		250 V	
Vl		250 V	
Ra		470 k	
Rg		3,3 Meg	
Vg	-1 V		-14 V
β	5 °		50 °
I_a	0,065 mA		0,063 mA
I_l	0 mA		0,5 mA

Abbildung 48: Betriebsdaten

Anmerkungen:

Das Schirmgehäuse ist potentialfrei und muss beim Einbau mit dem Metallchassis des Gerätes verbunden werden.

Sowohl der Katoden- als auch der Heizkreis sollten wechselstrommäßig mit Erde verbunden sein, um kapazitive Brummkopplungen in den extrem hochohmig gesteuerten nFET kurzzuschließen.

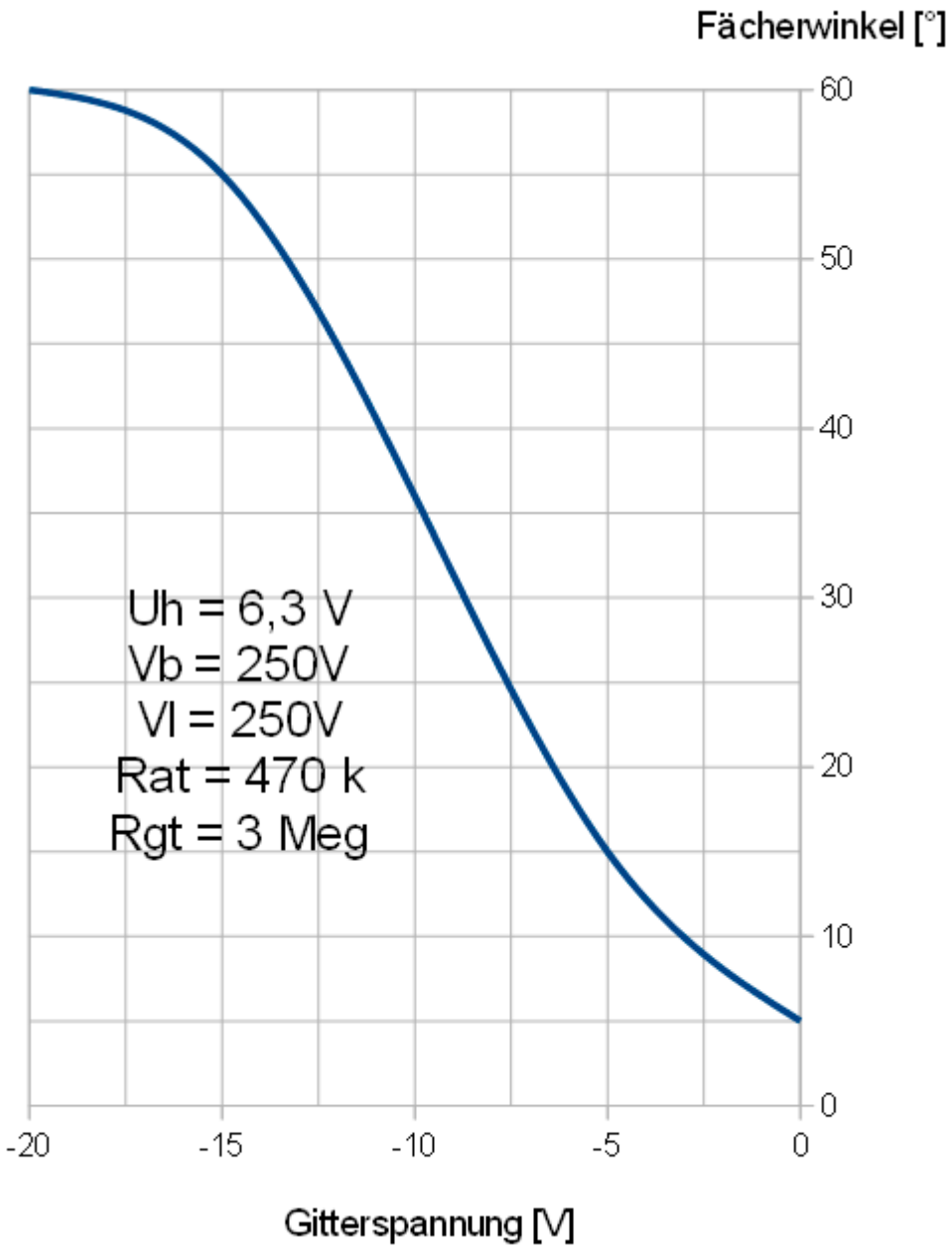
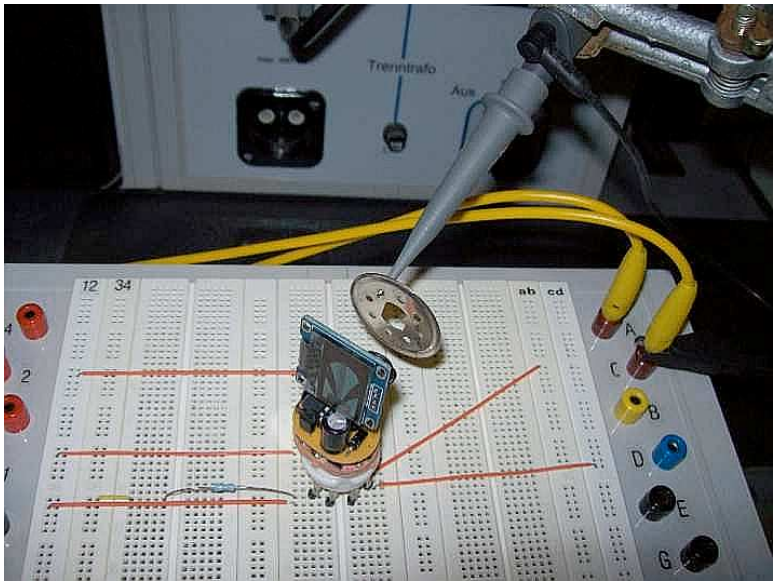


Abbildung 49: Kennlinie



Zur Bestimmung der HF-Abstrahlung wurde ein DSO genutzt. Es wurde eine kapazitive Sonde nahe der Elektronik angebracht.

Abbildung 50: Kapazitive Sonde

Bedingt durch den offenen Messaufbau mitten in Hamburg zeigt die FFT auch tagsüber ein gewisses Grundrauschen. Beachtet werden sollen nur Pegel oberhalb der unteren roten Linie. Dargestellt ist der Bereich zwischen 0 und 20 MHz.

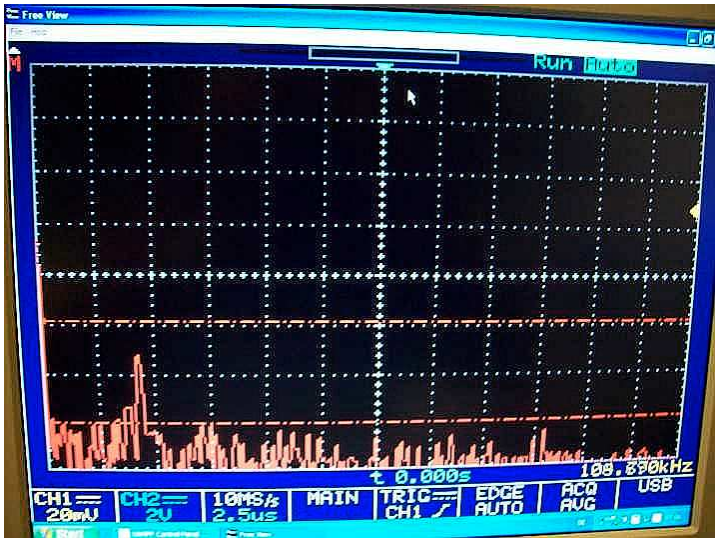


Abbildung 51: FFT Grundrauschen

Sobald man die Emulation in Betrieb nimmt, wird die untere Schwelle vielfach überschritten. Die obere rote Linie liegt 20 dB über der unteren Linie. Der Maßstab ist logarithmisch.

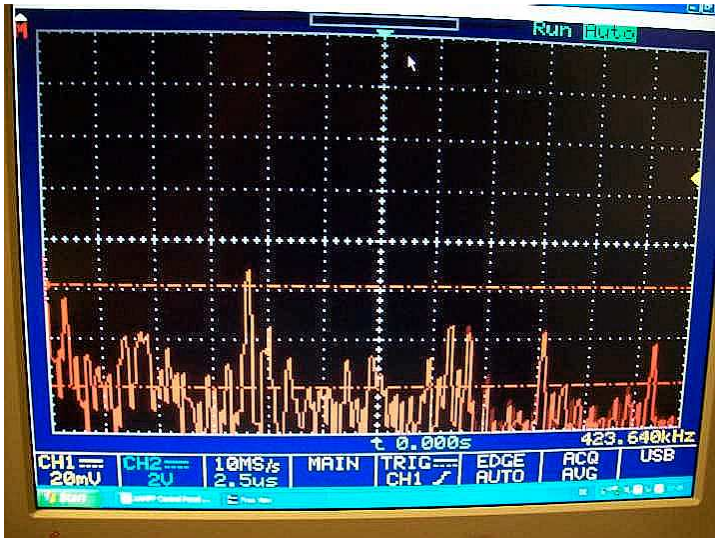


Abbildung 52: FFT ohne Schirmung

Dann wurde die Elektronik provisorisch ins Schirmblech eingeschoben.

Abbildung 53: Elektronik im Schirmblech

Das isolierte Blech zeigt praktisch keine Wirkung. Es fängt die internen Störungen auf und strahlt sie unvermindert nach außen ab.

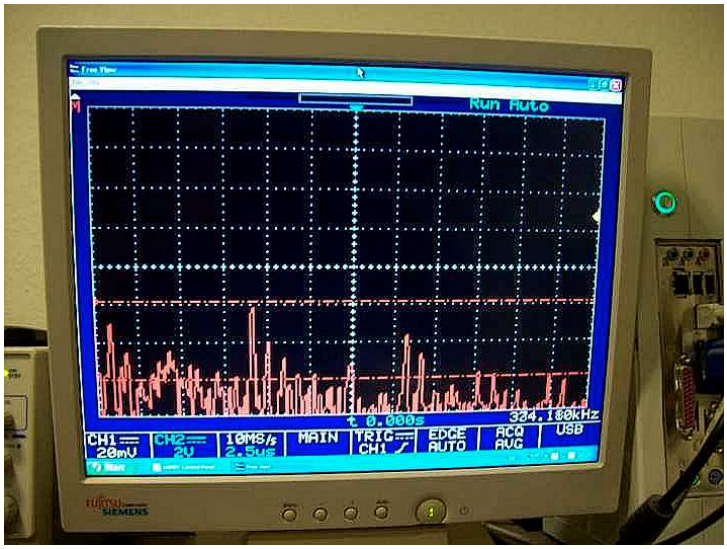
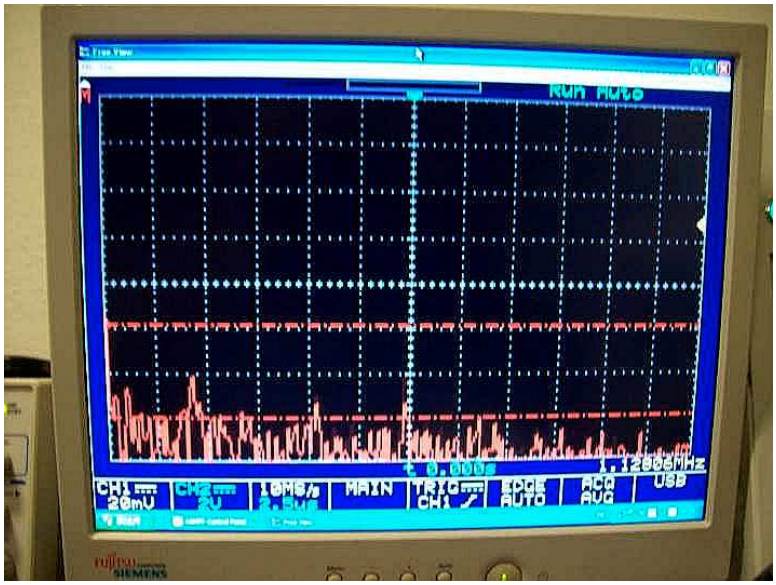


Abbildung 54: Isoliertes Schirmblech



Sobald man jedoch das Schirmblech erdet, wird die HF abgeleitet. Die Abstrahlungen werden um 20dB gemindert.

Abbildung 55: Geerdetes Schirmblech

Einbau in ein Radio

Die EM80-Emulation soll in einem Radio eingebaut werden. Dabei ist einiges zu beachten.

Denn es ist die erhebliche Brummempfindlichkeit der Emulation zu berücksichtigen, die der geforderten leistungsfreien Steuerbarkeit geschuldet ist. Die Brummempfindlichkeit entsteht durch die dazu notwendige äußerst hochohmige Beschaltung des FET-Gates.

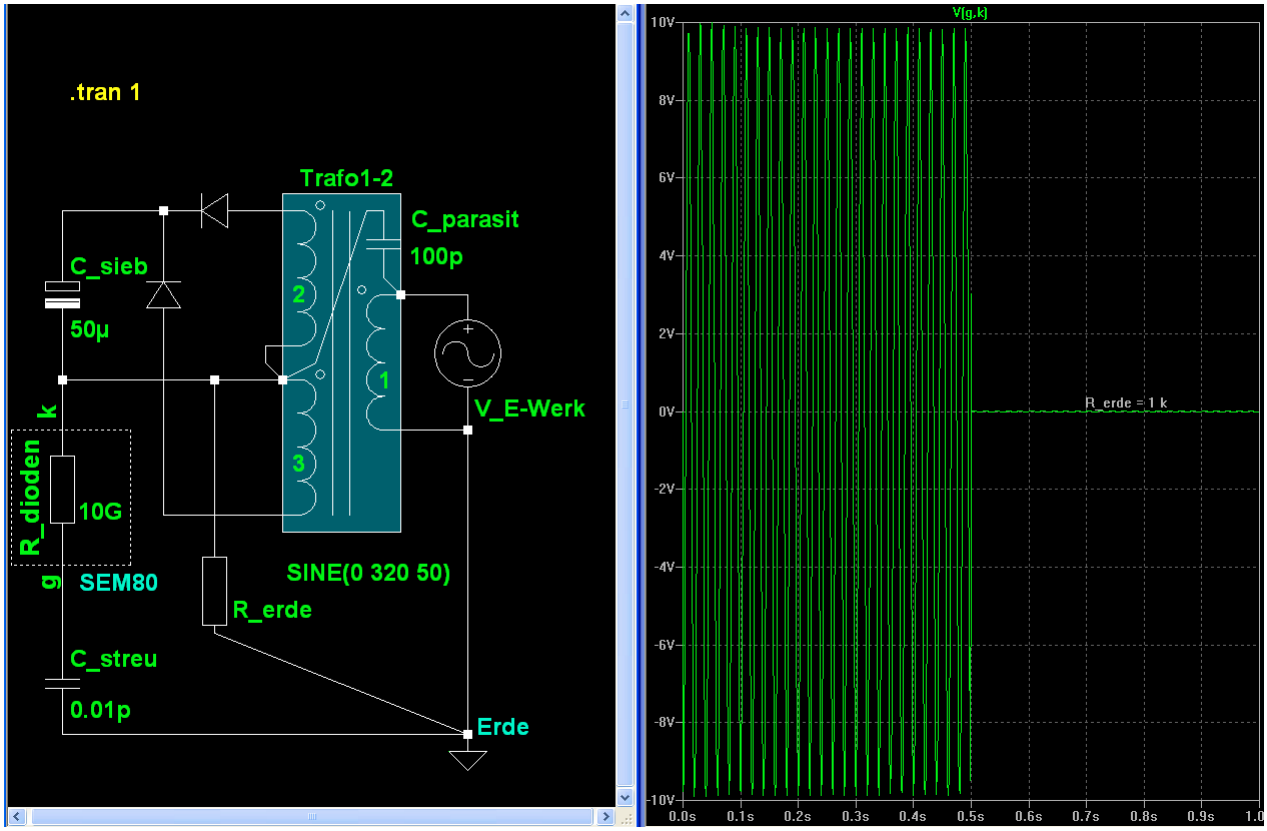


Abbildung 56: Simulation des Netz-Brummens

Angenommen sei ein Radio mit Trafo-Netzteil. Über dessen unvermeidliche parasitären Kapazitäten wird das erdbezogene Netzbrummen ins Gerät in folgender Weise eingeschleppt:

Zwischen Primär- und Sekundärwicklungen des Trafos besteht eine parasitäre Kapazität von angenommenen 100pF. Innerhalb des Emulators befinden sich zwischen Gate des FETs und Katodenanschluss Dioden in Sperrichtung. Deren wirksamer Widerstand wird mit 10 Gigaohm angenommen. Zwischen Gate des FETs und Erde wird eine Streukapazität von 10 Femtofarad (0,01 pF) angenommen.

In der Simulation wird die Katode die ersten 0,5s isoliert gegen Erde betrieben. Ab 0,5s wird dann ein Erdungswiderstand von 1k Ohm aktiviert. Dargestellt wird die Spannung zwischen Gate und Katode. Eine von Erde entkoppelte Katode führt zur vollen Durchsteuerung des FET mit 10Vs. Der Leuchtfächer der Emulation flackert wild hin und her. Sobald man jedoch die Katode ausreichend erdet, wird der Brummstrom an der Röhre vorbei geleitet und es ist kein Flackern mehr wahrzunehmen.

Auch in der Realität wirkt sich eine unzureichende Erdung der Katode dramatisch aus:

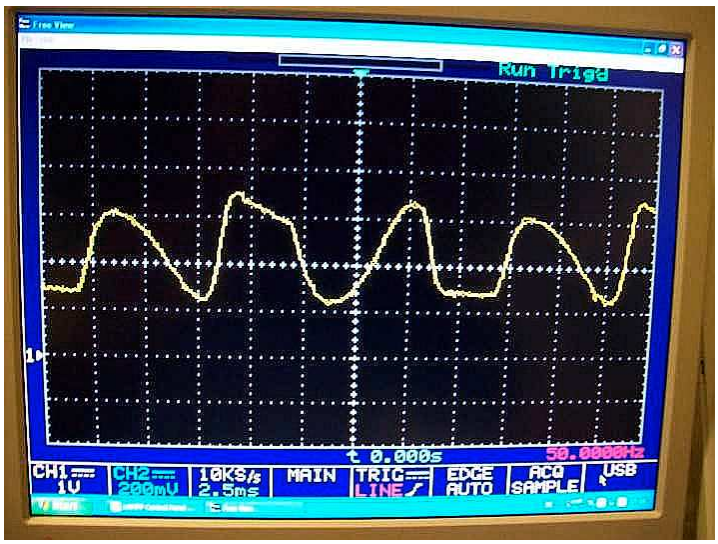


Abbildung 57: Brummen am ADC ohne Erdung der Katode

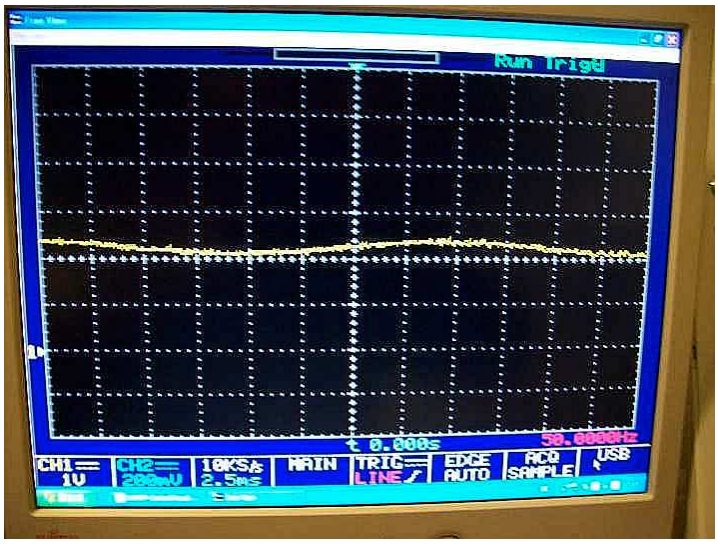


Abbildung 58: Brummen am ADC mit Erdung der Katode

Die Messung einer einwandfreien Erdung ist einfach.

Die Tastkopfspitze eines an dem Erdanschluss des Netzes geerdeten Scopes wird mit dem Katodenanschluss der EM80-Fassung verbunden. Der Masseanschluss des Tastkopfes bleibt unangeschlossen. Das Scope wird eine erhebliche Brummspannung von vielen Volt anzeigen.

Sobald man die Katode wechselstrommäßig erfolgreich geerdet hat, zeigt das Scope nur noch wenige Millivolt. Damit ist das über den Netztrafo eingeschleppte Brummen beseitigt worden.

Wie genau eine Erdung durchgeführt werden muss, muss fallweise entschieden werden. Allstromgeräte sind zum Beispiel schwierig zu erden. Per Batterie betriebene Geräte sind von diesen Problemen dagegen prinzipiell nicht betroffen.