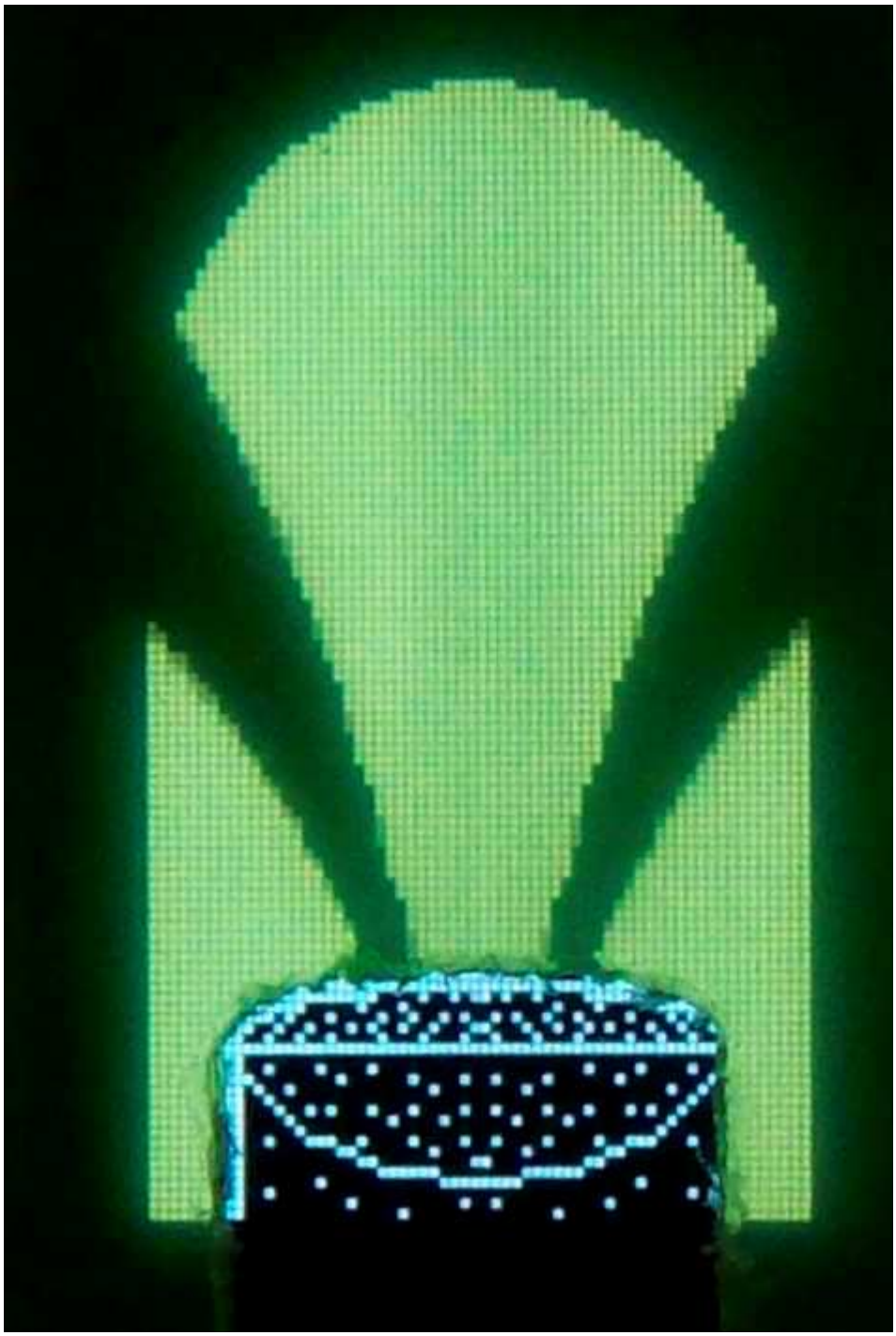


Emulation der EM80

Inhalt

Historie dieser Doku

21.11.2018 erste Veröffentlichung

05.12.2018 ADC auf 2,56V umgestellt. Optokoppler und Trimmer getauscht. Optokopplerstrom halbiert. Schaltplan und Software aktualisiert. Kapitel „Schirmgehäuse“, „Aufbau“, „Daten“ und „Historie dieser Doku“ hinzugefügt. Kapitel „Probleme“ gestrichen. Kapitel „Aussicht“ überarbeitet. Kapitel „Display“ um Farbfolie erweitert. Titelbild geändert.

06.12.2018 Kapitel „Daten“ um HF-Messungen erweitert.

Aktuell Neues Kapitel „Einbau in ein Radio“

Verzeichnis aller Abbildungen

Vorwort

Angeregt durch die Arbeiten an Emulationen magischer Augen durch BernhardWGF im WGF-Forum haben sich Peter (aka „laurel1“) und Wolfgang (aka „Rumgucker“) folgende Ziele gesetzt:

- Offenlegung aller Arbeiten im WGF und anderen Foren
- Emulation und Aufbau einer beengten Röhre (EM80)
- Kosten hierfür rund € 5,--
- Hinreichende Emulation aller Röhreneigenschaften
- Anpassbarkeit an andere Röhren, z.B. Stereo-Anzeigen

Diese Doku soll keine Bauanleitung 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. So wollen wir erfahrene Hard- und Softwarentwickler zu weiteren Arbeiten und Offenlegungen anregen.

Verwendete Hilfsmittel

- PC-Programm IMPBMP zum Import von BMP-Dateien (DIY, s. Anhang)
- PC-Programm IMPCNT zum Import von BMP-Dateien (DIY, s. Anhang)
- Entwicklungsumgebung für die DIY-Programme
- Programm MS Paint 5.1 zum Erstellen von BMP-Dateien
- ATTiny85-Entwicklungsumgebung AVR Studio 4.18
- Programmiergerät GALEP 4 für ATTiny85
- Programm LT Spice 4.23 zur Schaltplanerstellung
- Sprint-Layout 6.0 zur Konstruktion von Platinen und Blechteilen
- CNC-Fräsmaschine zur Fertigung von Platinen und Blechteilen
- 3-Walzen-Presse zur Rundbiegung von Blechteilen (DIY, s. Hardware)
- Datenblätter, Netzteile, Oszillographen, Multimeter, Steckbrett.

Hardware

Display

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

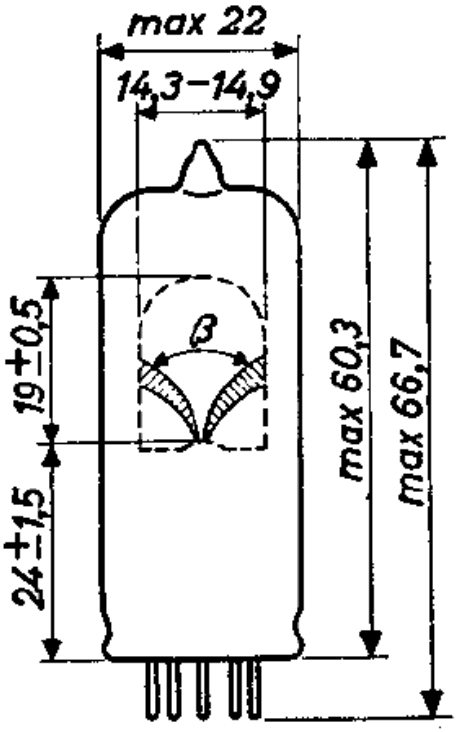


Abbildung 1: Maße EM80

Es gelang nicht, ein grün leuchtendes und preisgünstiges OLED-Display mit diesen Maßen zu finden. Es wurde daher ein weißes 0,96“-Display verwendet, was mit einer grünen Folie abgedeckt wird:

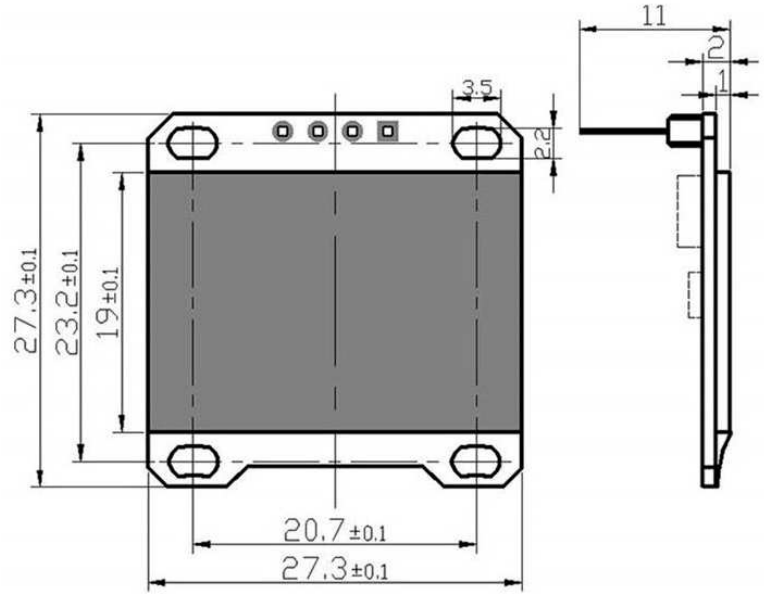
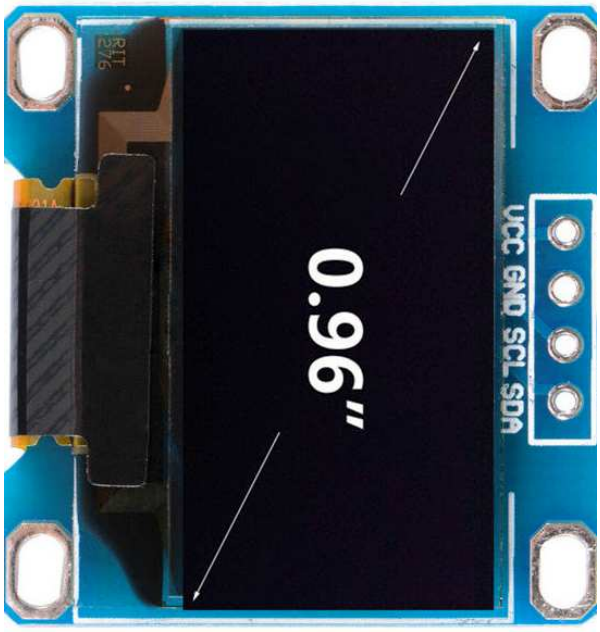


Abbildung 2: Maße OLED-Display



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.

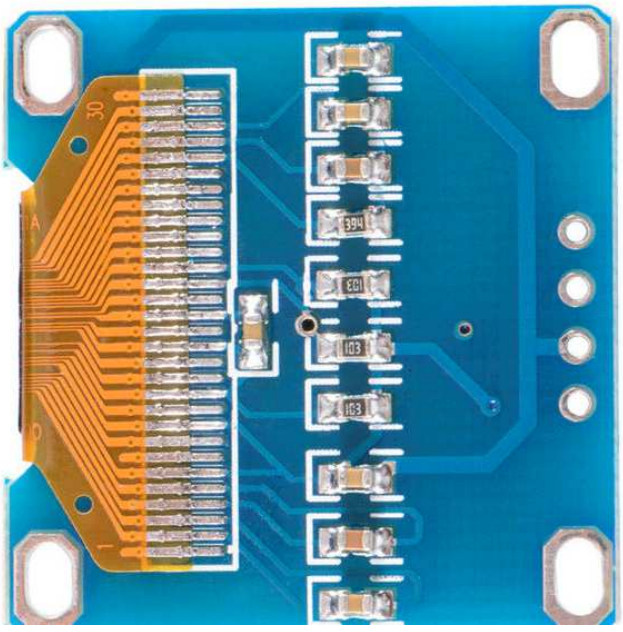


Abbildung 4: Rückansicht OLED-Display

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.

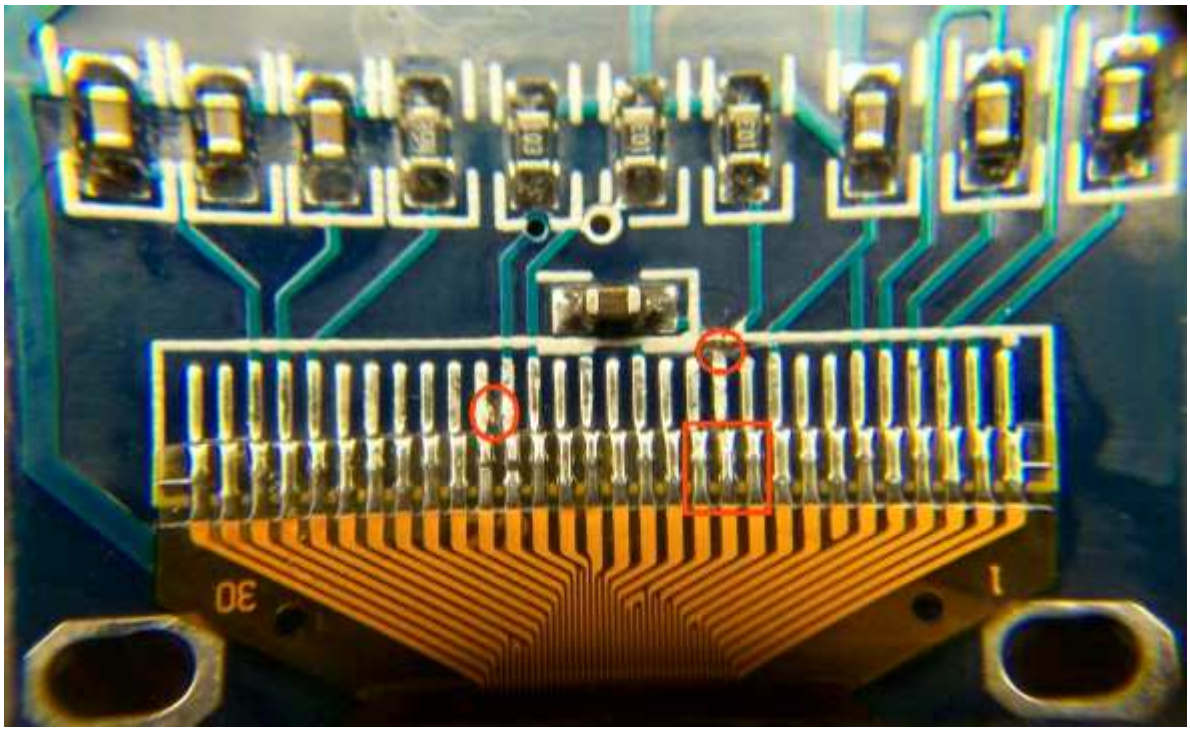


Abbildung 5: Umarbeitung des OLED-Displays

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.

<table>
<tr><th>PIN NO.</th><th>SYMBOL</th><th>FUNCTION</th></tr>
<tr><td>1</td><td>NC (GND)</td><td>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.</td></tr>
<tr><td>2</td><td>C2N</td><td rowspan="4">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.</td></tr>
<tr><td>3</td><td>C2P</td></tr>
<tr><td>4</td><td>C1P</td></tr>
<tr><td>5</td><td>C1N</td></tr>
<tr><td>6</td><td>V_{BAT}</td><td>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.</td></tr>
<tr><td>7</td><td>NC</td><td>NC</td></tr>
<tr><td>8</td><td>V_{SS}</td><td>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.</td></tr>
<tr><td>9</td><td>V_{DD}</td><td>Power supply for logic circuit.
This is a voltage supply pin. It must be connected to external source.</td></tr>
<tr><td>10</td><td>BS0</td><td rowspan="3">Communicating protocol select
These pins are MCU interface selection input. See the following table:
<table><tr><th></th><th>I²C</th><th>3-wire SPI</th><th>4-wire SPI</th><th>8-bit 68XX parallel</th><th>8-bit 80XX parallel</th></tr><tr><td>BS0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>BS1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>BS2</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>11</td><td>BS1</td></tr>
<tr><td>12</td><td>BS2</td></tr>
<tr><td>13</td><td>CS#</td><td>Chip select
This pin is the chip select input. The chip is enabled for MCU communication only when CS# is pulled low.</td></tr>
<tr><td>14</td><td>RES#</td><td>Power reset for controller and driver
This pin is reset signal input. When the pin is low, initialization of the chip is executed.</td></tr>
<tr><td>15</td><td>D/C#</td><td>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 I²C mode, this pin acts as SA0 for slave address selection.</td></tr>
<tr><td>16</td><td>R/W#</td><td>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.</td></tr>
<tr><td>17</td><td>E/RD#</td><td>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.</td></tr>
<tr><td>18 to 25</td><td>D0 to D7</td><td>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 I²C 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.</td></tr>
<tr><td>26</td><td>I_{REF}</td><td>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.</td></tr>
<tr><td>27</td><td>V_{COMH}</td><td>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}.</td></tr>
<tr><td>28</td><td>V_{CC}</td><td>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.</td></tr>
<tr><td>29</td><td>V_{LSS}</td><td>Ground of analog circuit
This is an analog ground pin. It should be connected to V_{SS} externally.</td></tr>
<tr><td>30</td><td>NC (GND)</td><td>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.</td></tr>
</table>

Abbildung 6: Anschlüsse Folienleiter

Um eine grün leuchtende Darstellung zu erreichen, wurde eine Farbfolie angefertigt und mit Teppichklebeband an zwei Punkten der Glasfläche angeklebt.

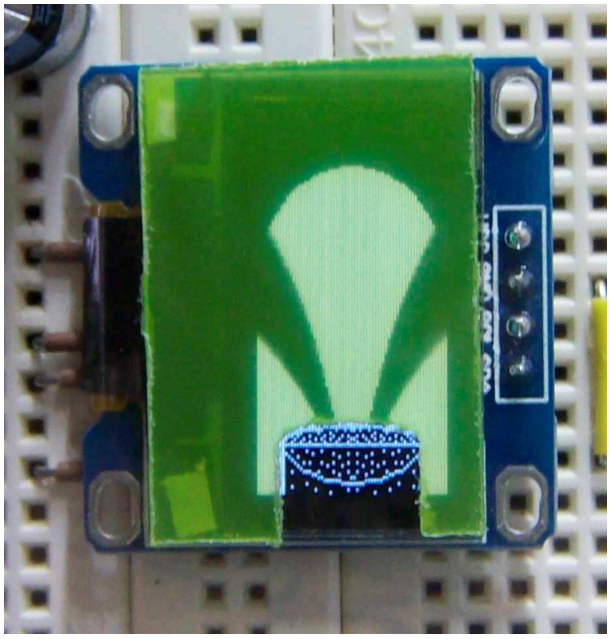


Abbildung 7: Farbfilter-Montage

Der Effekt ist brauchbar.

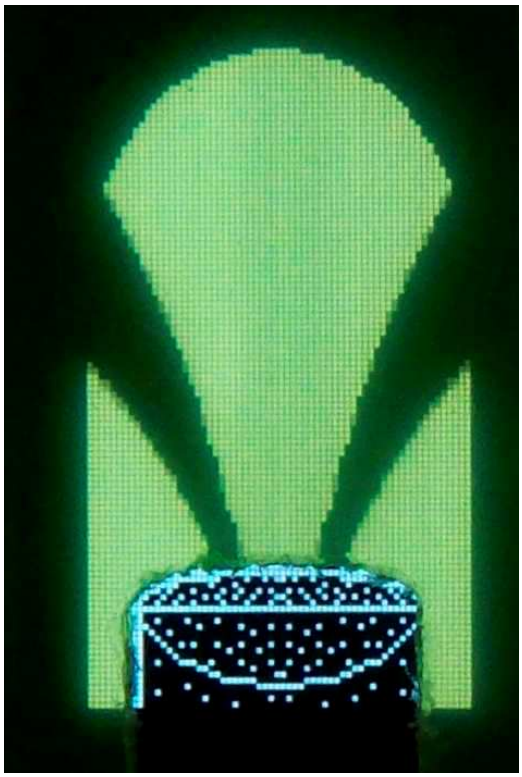


Abbildung 8: Farbfilter in Aktion

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.

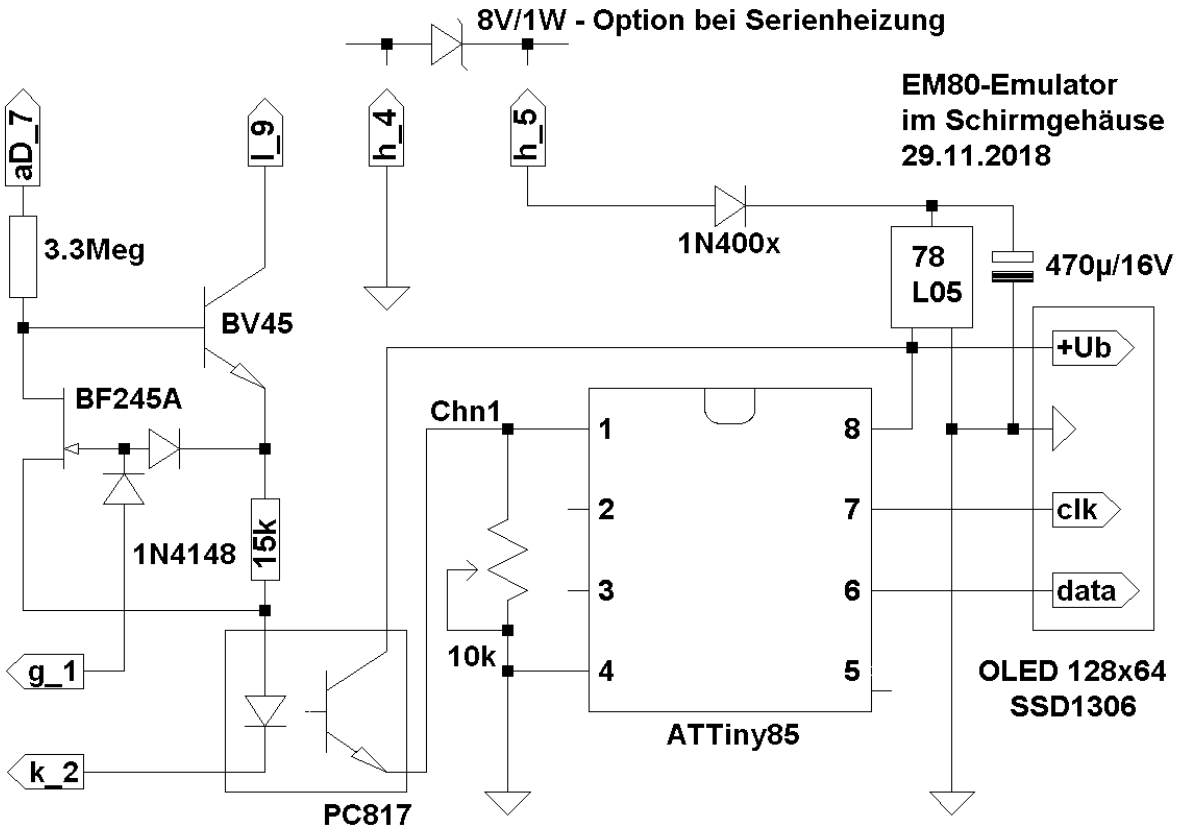


Abbildung 9: Gesamtschaltbild

Der Emulator toleriert Heizspannungen von 4V~ und 12V~. Der 78L05 dient dabei nicht der Stabilisierung sondern der Begrenzung. Das Display verfügt über einen eigenen LDO, der Versorgungsspannungen ab 3,3V akzeptiert. Und der ADC und die anderen Innereien des Tiny arbeiten weitgehend unabhängig von der Versorgungsspannung. Ein Ersatz des 78L05 durch die LDO-Version gestattet auch die Heizung mit 3V~.

Keine Vorteile dagegen bringt die Umrüstung auf eine Schottky-Diode. Die Stromflusswinkel der Diode ist in dieser Schaltung klein. Trotz geringer mittlerer Stromentnahme durch die nachfolgende Elektronik fließen mehrere hundert Milliampere Elko-Ladeströme. Der Spannungsabfall beispielsweise einer Schottky BAT48 beträgt in dieser Schaltung 900mV. Bei gleicher Baugröße ist eine normale pn-Diode dem überlegen. Durch Einbau einer MBRS360 könnte man allerdings 400mV Spannungsabfall einsparen. Aber die Schottky ist viel größer.

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 0.5 mA. Der Feinabgleich wird mit dem Trimmer auf Controllerseite durchgeführt. Innerhalb des Controllers befindet sich eine Kennlinientabelle.

Schirmgehäuse

Die im Analogteil benötigte elektrostatische Schirmung und auch die Schirmung gegen abgestrahlte HF lässt sich am einfachsten mit einem Metallgehäuse erreichen. Es genügt ein Schirmgehäuse, denn die intern erzeugte Störstrahlung kann den eigenen Analogteil nicht beeinflussen. Dieser reagiert nur auf Brummen.

Ideal ist eine Röhre aus dünnem Aluminiumblech. Leider lassen sich die von der Originalröhre vorgegebenen Maße nicht ganz erreichen. Das Displaymodul ist dafür zu breit. Der Querschnitt ist daher oval.

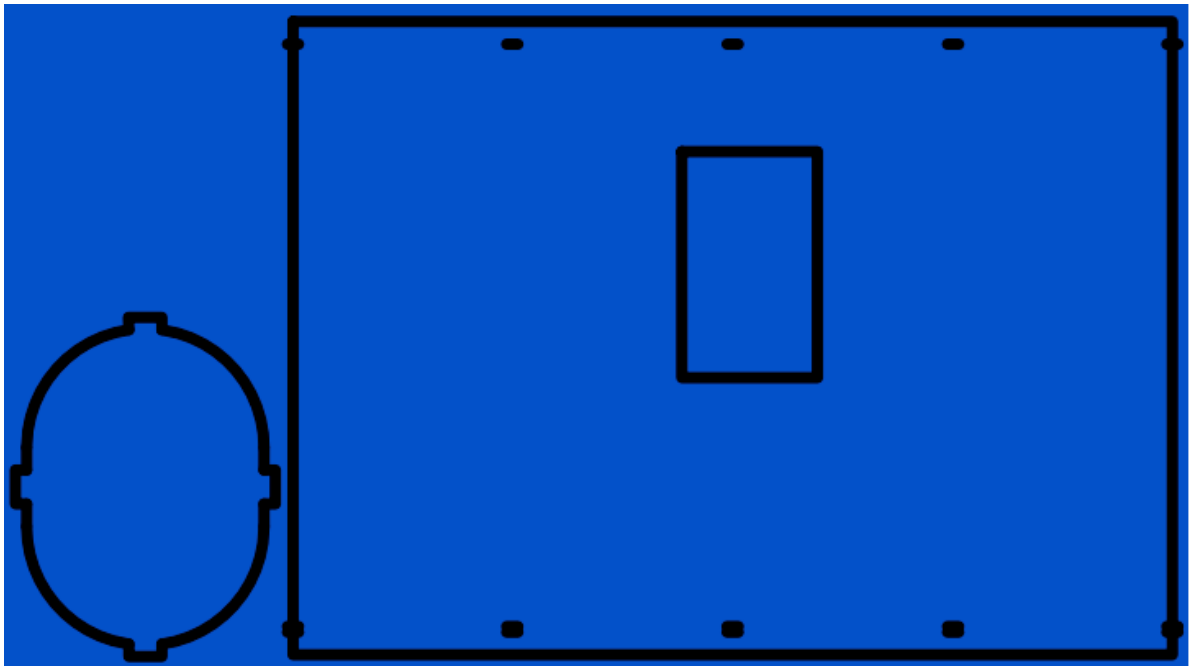


Abbildung 10: Blechteile

Um das Blech zu „walzrunden“, wurde eine 3-Walzen-Biegemaschine gebaut:

Abbildung 11: Prinzip 3-Walzen-Biegemaschine (Wikipedia)

Leider war die Arbeit schwierig und das Ergebnis nicht brauchbar.

- Das größte Problem lag darin, dass man bei dieser Konstruktion nicht zwischen die Walzen hineinsehen kann. Die drei Druckpunkte sind nicht sichtbar, weil die Walzenlagerung die Sicht versperrt.
- Weiterhin kann man bei dieser Konstruktion die Radien nicht messen, da die Kurbel und die beiden Seitenträger Messungen behindern.
- Weiterhin haben die in diesem Projekt benötigten dünnen Walzen einen großen Schlupf zum Blech. Das Blech wird durch die schlupfende Antriebswelle auf der Innenseite verschrammt und durch die vom Blech nicht genügend mitgeführten Außenrollen auch auf der Außenseite.
- Weiterhin war die Höhenverstellung der Andruckrolle mühsam, denn es mussten stets die beiden Enden der oberen Walze einzeln - aber mit gleichem Druck - nachgespannt werden.

Zur Rohrfertigung ist die Vorrichtung geeignet. Aber zum gezielten Radienbiegen einzelner Abschnitte ist die Maschine nicht brauchbar. Es musste eine Biegemaschine erdacht werden, die diese Nachteile nicht hat.

Es wurde folgende Vorrichtung für kleine und dünne Bleche gebaut:

Das Gerät ist eine Presse. Die drei Walzen drehen sich nicht. Somit gibt es auch keinen Schlupf.

Die Presskraft wird von einem Maschinenschraubstock geliefert und wirkt auf drei Rohre. Die zwei Rohre links liegen bündig aneinander und das rechte Rohr kann in die Mitte dieser beiden Rohre eingepresst werden.

Abbildung 12: 3-Walzen-Presse

Von der Seite aus kann man die drei Auflagepunkte sehen und das Biegegut positionieren und unbehindert messen.

Abbildung 13: 3-Walzen-Presse Druckpunkte

Die Presse zentriert sich automatisch.

Zum Einlegen des Werkstücks muss man lediglich die Kurbel zurückziehen, das Blech einschieben und mit der beweglichen Rolle festklemmen.

Abbildung 14: 3-Walzen-Presse Aufnahmeposition

Dann wird das Werkstück mit kleinen Hüben schrittweise gepresst. Man kann die Kraft gefühlvoll dosieren.

Schließlich hat sich das Werkstück um die Arbeitswalze herum gewickelt.

Abbildung 15: 3-Walzen-Presse Werkstück ausfahren

Zum Entnehmen des Werkstücks wird die Arbeitswalze einseitig abgeschraubt und hochgeklappt.

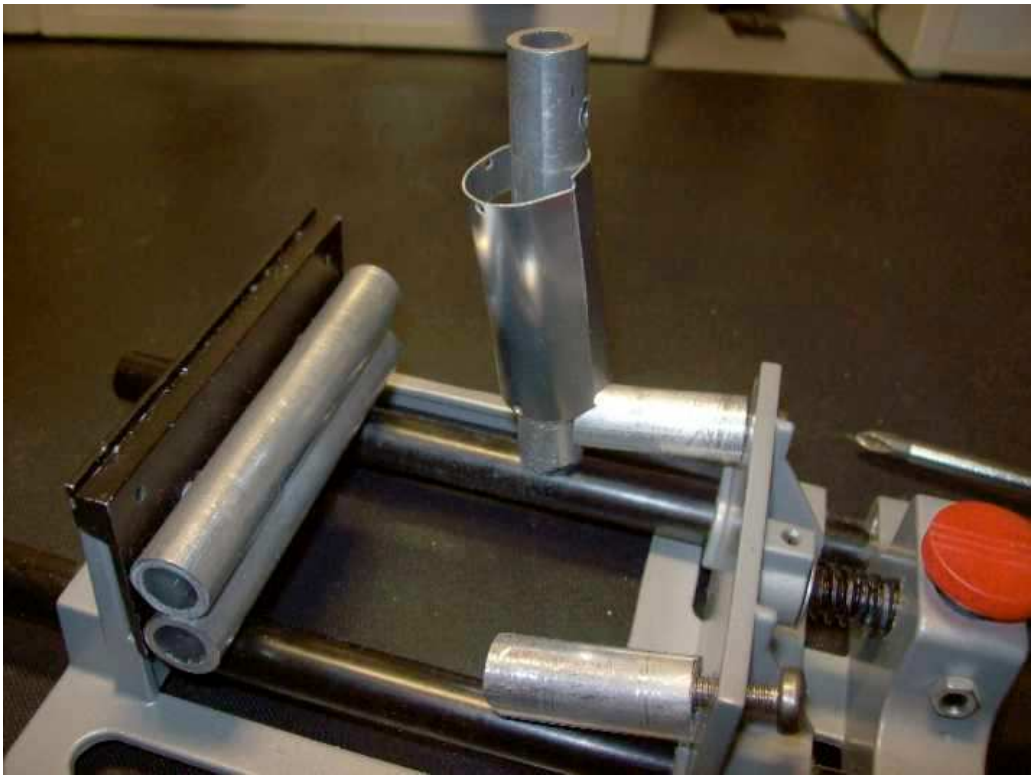


Abbildung 16: 3-Walzen-Presse Werkstück entnehmen

Nach zwei Minuten Arbeit.....

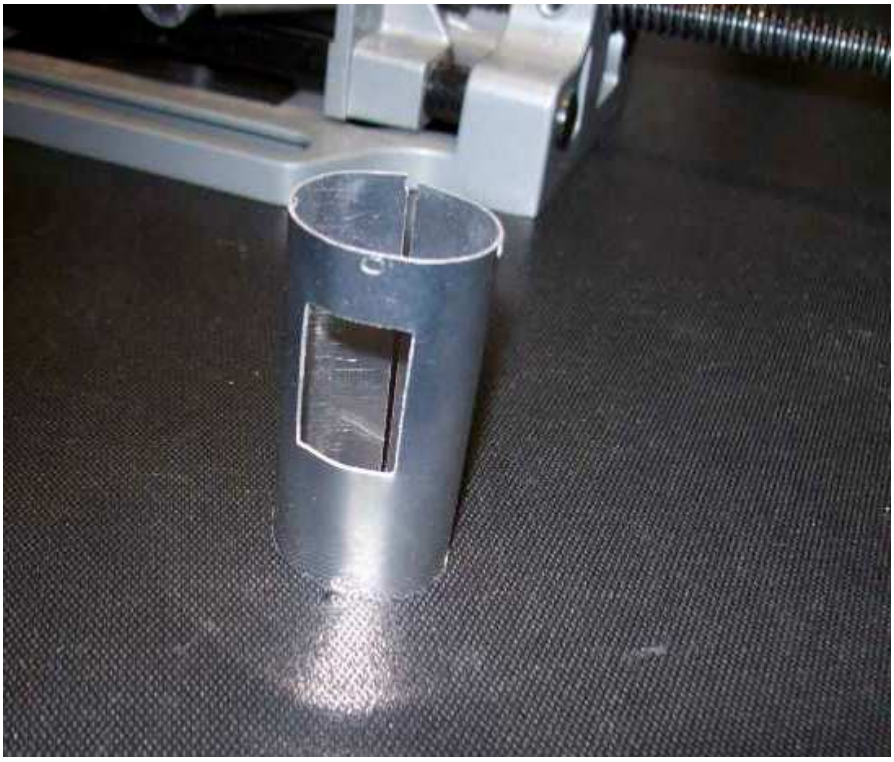


Abbildung 17: 3-Walzen-Presse Aller-erster Biegeversuch

Abbildung 18: Gehäuse mit Deckel und Gravur

Aufbau

Als Vorbild für den Aufbau dient die amerikanische Tinkertoy-Technologie.

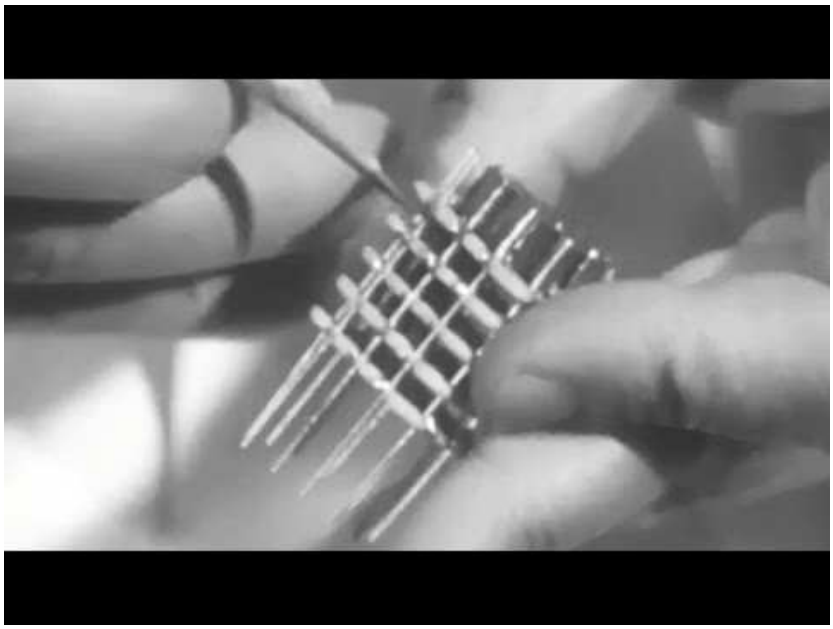


Abbildung 19: Tinkertoy-Aufbau

Es werden drei Platinen im Querschnitt der Röhre übereinander gestapelt. Die untere Platine trägt die Sockelstifte und überträgt die Kraft beim Stecken und Ziehen der Röhre mit vier kleinen Nasen auf das Schirmblech.

Bündig darüber befindet sich die Analogplatine, deren bedrahtete Bauteile wie SMD auf der Leiterbahnseite liegend bestückt werden.

Den Abschluss des 25mm dicken Stapels bildet die konventionell bestückte Digitalplatine mit Controller und Stromversorgung.

Darüber befindet sich das hochkant stehende Display und als Deckel wird ein Alublech mit Nasen verwendet.

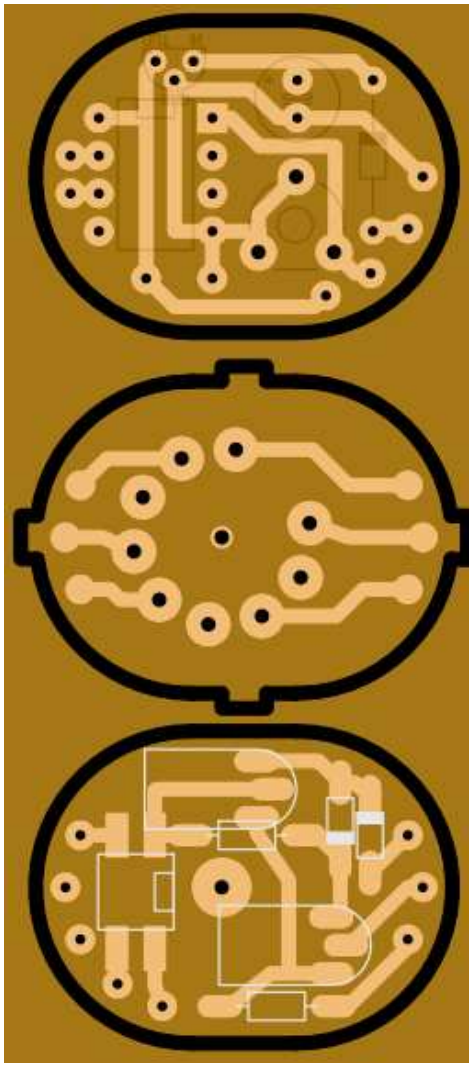


Abbildung 20: Platinenlayout

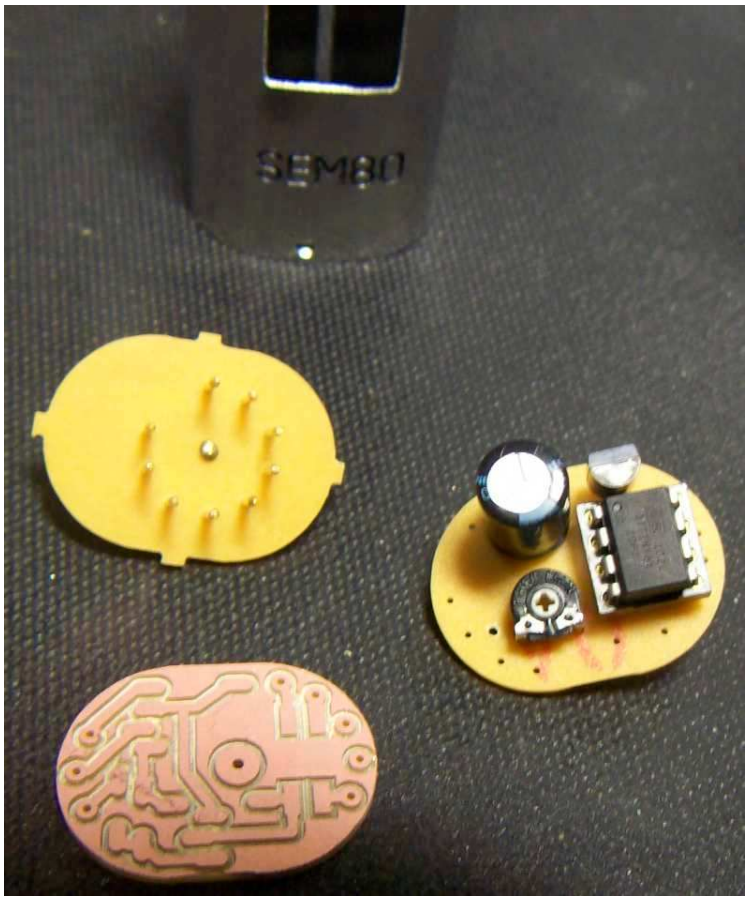


Abbildung 21: gefräste Platinen

Abbildung 22: Gesamt-Elektronik

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.

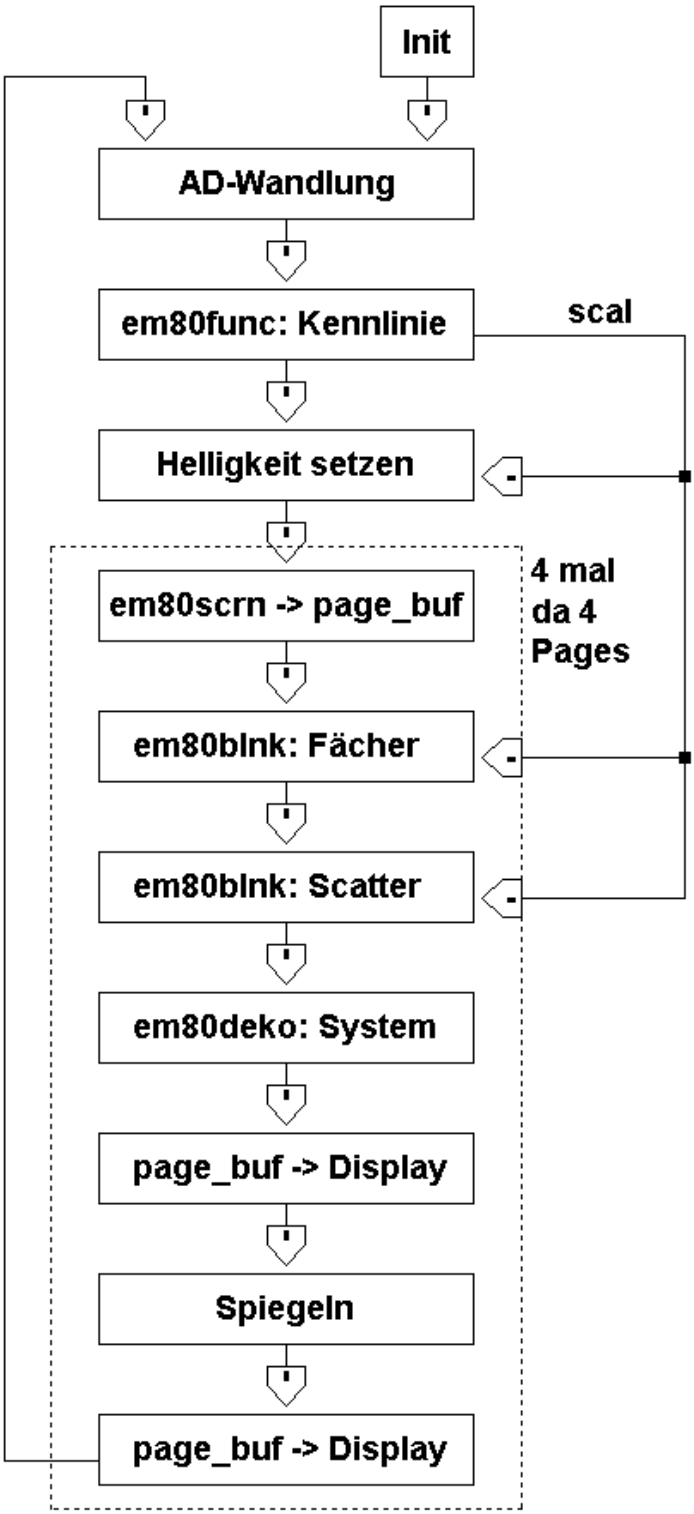


Abbildung 23: Softwareübersicht

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

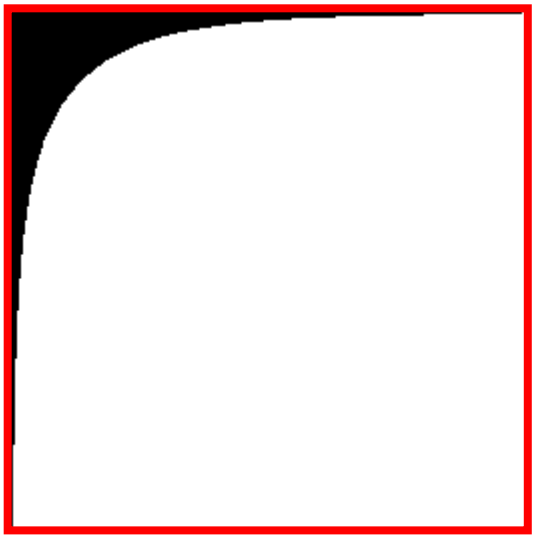


Abbildung 24: Röhrenkennlinie

Abbildung 25: Fächer mit Leuchtschirmkrümmung

Abbildung 26: Leuchtschirm

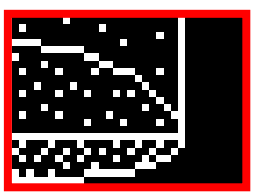


Abbildung 27: Elektrodensystem

So entsteht ein relativ realistisches Aussehen:

.

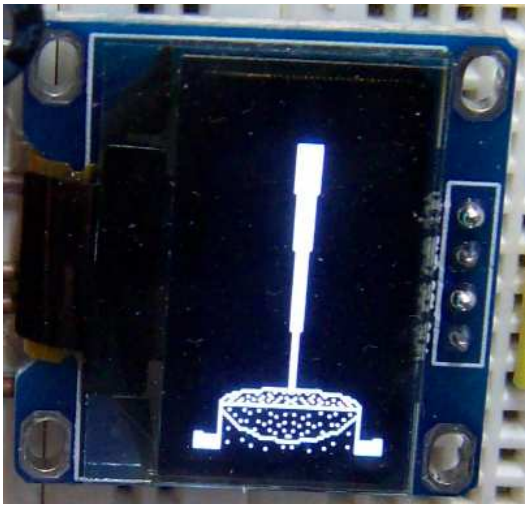


Abbildung 28: Fächer geschlossen

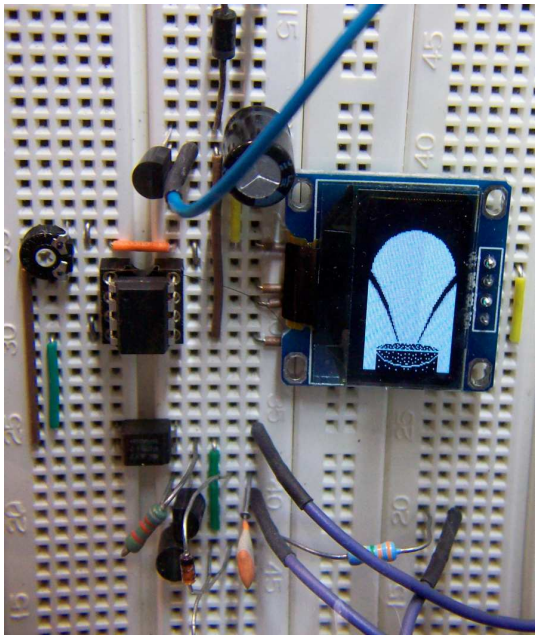


Abbildung 29: Fächer geöffnet

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

Auch die Controllerspeicher werden nur wenig genutzt:

Flash:	2414 bytes (29.5% Full)
RAM:	128 bytes (25.0% Full)
EEPROM:	0 Bytes (0.0% Full)

Abbildung 30: Controller-Ressourcen

Display-Treiber

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

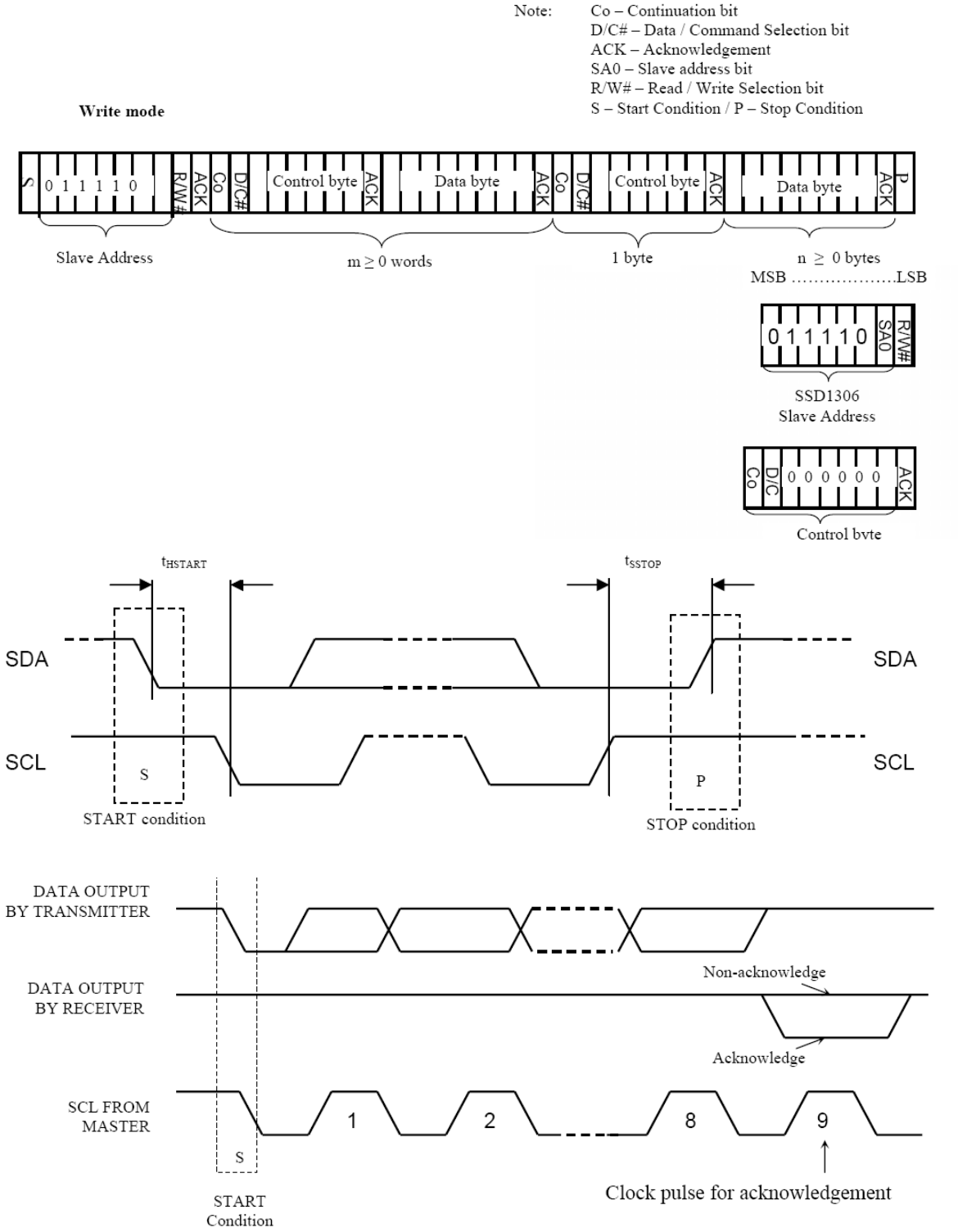


Abbildung 31: I²C-Protokoll

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 32: Listing des I²C-Treibers „usi_drv.h“