Ergänzung zum Beitrag in FA 7/20, S. 586 ff.
„Raspberry Pi als Programmiergerät für AVR-Mikrocontroller“

In [1] wurde zur Programmierung von AVR-Controllern mit dem Raspberry Pi am Beispiel eines ATtiny24 die Programmierung desselben mithilfe der Entwicklungsumgebung *Geany* in *GCC-AVR* erläutert, eine Schaltung zur ISP-Programmierung dargestellt, unter anderem je ein Python-Programm zur Programmierung, zum Auslesen des Programmspeichers sowie eines zum Lesen der Signatur-Bytes vorgestellt.

Hier wird ergänzend eine für den ATtiny24 passende Experimentierschaltung mit zwei Siebensegmentanzeigen dargestellt, die an mehreren anderen AVR-Controllern angeschlossen werden kann.

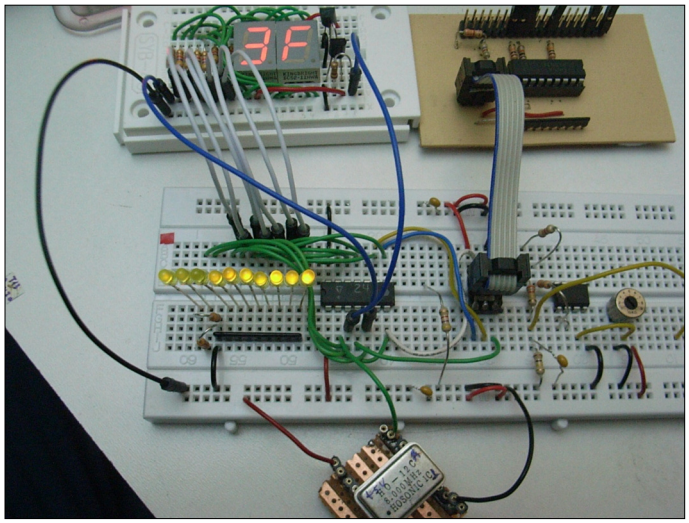


Bild A1: Betrieb eines ATtiny24 mit externem Quarzoszillator

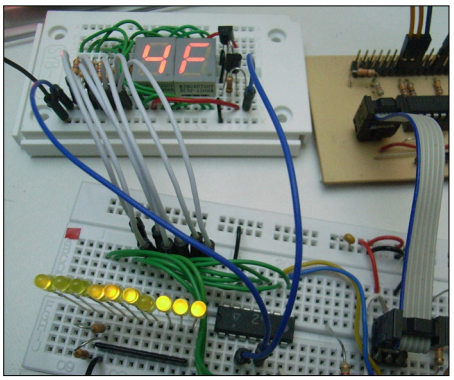


Bild A2: Laboraufbau der Schaltung in Bild 7 auf Steckboards

■ Disassemblierung einer von ARV-GCC erzeugten Hex-Datei

Bei der Compilierung der Datei *T24_blink1.hex* mit AVR-GCC entsteht unter anderem auch die Datei *T24_blink1.elf*. Von dieser kann man sich im LXTerminal ein Disassemblerlisting anzeigen lassen, wenn man in diesem

avr-objdump -d T24_blink1.elf

eingibt. Bild A3 zeigt einen Teil der Bildschirmausgabe im *LXTerminal*, und zwar einen Ausschnitt aus der Funktion *main*. Durch Vergleich mit den Zeilen im Programmlisting *T24_blink.c* lässt sich erkennen, auf welche Art und Weise diese jeweils für den Mikrocontroller in für ihn Befehle übersetzt wurden.

■ Vorbereitung

Damit man die zum Beitrag gehörenden Python-Dateien nach der Übertragung in den Ordner */home/pi/ATtiny24* ausführen kann, ist im *LXTerminal*

*sudo chmod +x *.py*

einzugeben. Erst danach lässt sich das Python-Programm *prog16wd2.py* zum Programmieren eines ATtiny24, ATtiny25, ATtiny26, ATtiny13 und eines ATtiny2313 im *LXTerminal* durch Eingabe von

./prog16wd2.py

starten.

■ Übertragung mehrerer Bytes in den EEPROM eines ATtiny24

Der EEPROM-Bereich eines ATtiny24 ist 128 Bytes groß. Nach dem Löschen des Mikrocontrollers beinhalten alle Speicherplätze den Wert 255 bzw. hexadezimal FFh. Ich habe für den ATtiny24 das C-Programm *T24_EEP01.c* geschrieben, mit dem sich eine bestimmte Anzahl Bytes in seinen EEPROM übertragen und zur Kontrolle mithilfe des Python-Programms *EEPROM_read.py* auslesen lassen. Bild A4 zeigt den Ausschnitt aus dem C-Programm *T24_EEP01.c*, mit dem ein Teil des EEPROM beschrieben und anschließend wieder gelesen werden kann, hier die Funktionen *EEPROM_write* sowie *EEPROM_read*.

In dem in Bild A6 dargestellten Hauptprogramm werden zunächst nacheinander 16 Bytes zur Darstellung von Ziffern und Zeichen auf einer Siebensegmentanzeige an die EEPROM-Adressen 00h bis 0Fh übertragen. Anschließend wird in einer Endlosschleife von den sechzehn zuvor beschriebenen EEPROM-Adressen fortlaufend mit einer Pause von jeweils1 s gelesen, um die entsprechenden Zeichen abwechselnd auf einer der beiden Siebensegmentanzeigen der Experimentierschaltung in Bild A7 darstellen zu können.

Soll der gesamte EEPROM-Inhalt oder ein Teil desselben im *LXTerminal* angezeigt werden, ist dort

./read_EEPROM.py

einzugeben und die Anzahl der Bytes zu nennen, die angezeigt werden sollen. In Bild A5 werden nach den ersten 16 Bytes

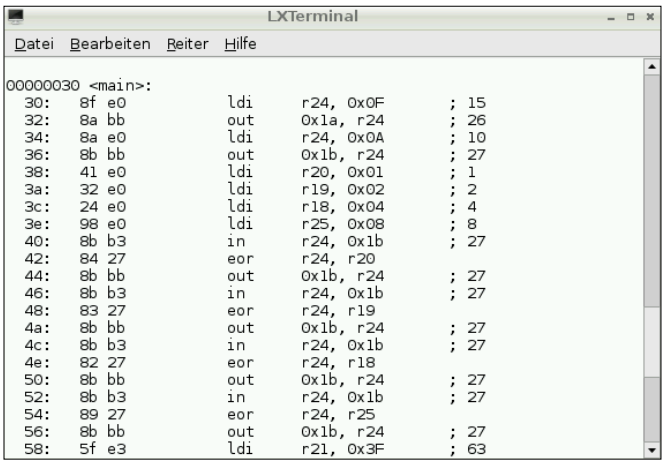


Bild A3: Ausschnitt aus disassemblierter Datei *T24_blink1.elf*

```
//vgl. Datenblaetter ATtiny24/44/84, S. 18, C Code Example
void EEPROM_write(unsigned int ucAddress,unsigned char ucData)
{
    while(EECR & (1<<EEPE));
    EECR=(0<<EEPM1)|(0<<EEPM0);
    EEAR=ucAddress;
    EEDR=ucData;
    EECR |= (1<<EEMPE);
    EECR |= (1<<EEPE);
}
//vgl. Datenblaetter ATtiny24/44/84, S. 18, C Code Example
unsigned char EEPROM_read(unsigned int ucAddress)
{
while(EECR & (1<<EEPE));
EEAR=ucAddress;
EECR |=(1<<EERE);
return EEDR;
}
```

Bild A4: Funktionen zum Beschreiben und Lesen aus dem EEPROM eines ATtiny24

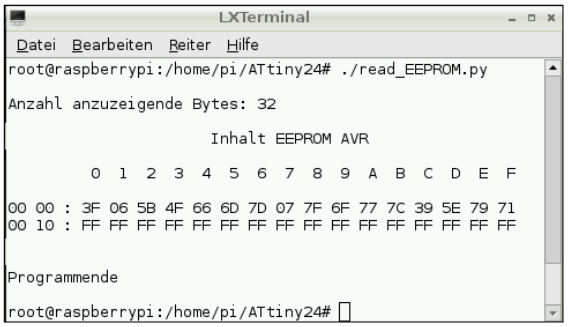


Bild A5: Anzeige von EEPROM-Inhalten eines ATtiny24 mit dem Python-Programm *read_EEPROM.py*

auch die Inhalte der 16 folgenden, noch nicht belegten EEPROM-Speicherplätze angezeigt. Bei der Übertragung einer anderen Hex-Datei in den ATtiny24 wird dessen EEPROM vollständig gelöscht.

■ Übertragen einer Datei im Intel-Hex-Format in den EEPROM

Soll eine EEPROM-Datei im Hex-Format in den EEPROM eines ATtiny24 übertragen werden, so kann dies mithilfe des Python-Programms *prog_EEPROM3.py* erfolgen. Bild A8 zeigt einen Screenshot dazu. Man startet das Programm durch Eingabe von

python prog_EEPROM3.py

bzw.

./prog_EEPROM3.py

im *LXTerminal*. Dann gibt man den korrekten Namen der betreffenden Hex-Datei an, die darauf in den EEPROM-Speicher des ATtiny24 übertragen wird. In Bild A9 wurde die Datei *T24_EEP_read02.hex* in den Editor *Leafpad* geladen.

Darauf kann man sich gemäß der Darstellung in Bild A10 den gesamten EEPROM-Inhalt oder einen Teil davon beispielsweise dem Programm *AVR_ISP_contr.py* anzeigen lassen. Man startet dieses im Arbeitsordner im *LXTerminal* durch Eingabe von

sudo python ./AVR_ISP_contr.py

bzw.

sudo ./AVR_ISP_contr.py

Voraussetzung ist, dasss man zuvor alle zum Beitrag gehörenden Python-Programme im *LXTerminal* durch Eingabe von

*sudo chmod +x *.py*

ausführbar gemacht hat. Darauf gibt man, ähnlich wie bei dem bereits genannten Python-Programm *read_EEPROM.py*, hier die Anzahl der anzuzeigenden Words im dafür vorgesehenen Textfeld an und klickt auf die Schaltfläche *Read EEPROM Data Memory*, worauf nach kurzer Zeit eine Darstellung ähnlich wie in Bild A10 erfolgt.

Die Ausgabe erfolgt hier im Intel-Hex-Format. Ein Vergleich der Prüfsummen am Ende der Zeilen in Bild A9 mit denjenigen in Bild A10 zeigt, dass die Übertragung der Hex-Datei in den EEPROM des Mikrocontrollers gelungen ist.

Die erste Zeile unterscheidet sich allerdings von derjenigen in Bild A9, da bei der Programmierung des EEPROMs mittels *prog_EEPROM3.py* das sich gerade im Programmspeicher des Mikrocontrollers befindliche Programm nicht gelöscht wird. Dieses wird unmittelbar nach der Übertragung der Bytes in den EEPROM wieder durch High-Pegel am Reset-Pin des Mikrocontrollers gestartet.

Gemäß den Ausführungen oben überträgt das Programm im ATtiny24 an die ersten 16 EEPROM-Adressen die Bytes für die Darstellung der Ziffern 0 bis 9 sowie der Ziffern A, B, C, D, E und F und liest anschließend diese darauf von dort, um die Inhalte auf einer Experimentierschaltung in Bild A7 mit zwei Siebensegmentanzeigen darzustellen.

Der zuvor mit *AVR_ISP_contr.py* vom ATtiny24 ausgelesene EEPROM-Inhalt lässt sich nach dem Anklicken von *Save as* oben im Fenster des *AVR ISP Control Panel* bei Bedarf speichern, wenn zuvor ein Dateiname, hier *T24_EEP_dt1.hex*, eingegeben und anschließend die Schaltfläche *Speichern* gedrückt wird, wie in Bild A11.

Das von mir entworfene Programm *AVR_ISP_contr.py* ist selbstverständlich auch zum Auslesen von EEPROM-Inhalten anderer AVR-Mikrocontroller geeignet. Mit ihm lässt sich auch der Programmspeicher eines solchen Mikrocotrollers teilweise oder komplett auslesen, wenn nach dem

```
int main(void)
 {
DDRA=0xFF;
DDRB=0b00000111;
clearbit(PORTB,PB1);
 //EEPROM-Adresse, Byte //Zeichen
 EEPROM_write(0x00,63); //0
 _delay_ms(10);
 EEPROM_write(0x01,6);  //1
 _delay_ms(10);
 EEPROM_write(0x02,91); //2
 _delay_ms(10);
 EEPROM_write(0x03,79); //3
 _delay_ms(10);
 EEPROM_write(0x04,102);//4
 _delay_ms(10);
 EEPROM_write(0x05,109);//5
 _delay_ms(10);
 EEPROM_write(0x06,125);//6
 _delay_ms(10);
 EEPROM_write(0x07,7);  //7
 _delay_ms(10);
 EEPROM_write(0x08,127);//8
 _delay_ms(10);
 EEPROM_write(0x09,111);//9
 _delay_ms(10);
 EEPROM_write(0x0A,119);//A
 _delay_ms(10);
 EEPROM_write(0x0B,124);//b
 _delay_ms(10);
 EEPROM_write(0x0C,57); //C
 _delay_ms(10);
 EEPROM_write(0x0D,94); //d
 _delay_ms(10);
 EEPROM_write(0x0E,121);//E
 _delay_ms(10);
 EEPROM_write(0x0F,113);//F
 _delay_ms(10);
setbit(PORTB,PB1);
while(1) // Endlosschleife
{
for (i=0; i<16; i++)
{
    PORTA=EEPROM_read(i);
    _delay_ms(1000);
    togglebit(PORTB,PB1);
}
}
}
```

Bild A6: Hauptprogramm zur Übertragung von Bytes in den EEPROM eines ATtiny24 und anschließendes Lesen vom EEPROM

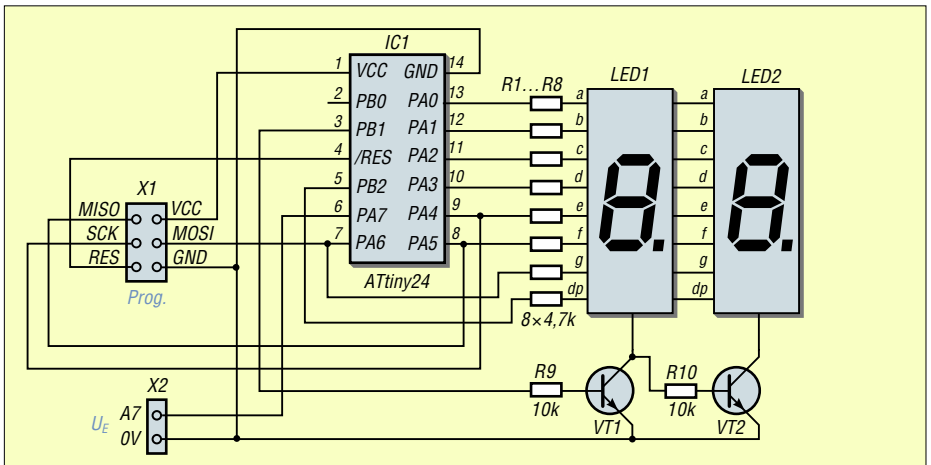
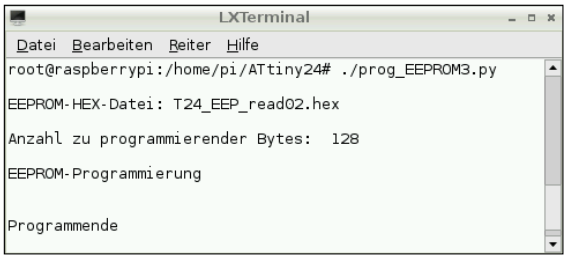


Bild A7: Beispielschaltung mit zwei Siebensegmentanzeigen mit gemeinsamer Katode, die ein ATtiny24 über zwei NPN-Transistoren im Multiplexbetrieb ansteuert

Eingeben der Anzahl von Words, die man im Intel-Hex-Format dargestellt haben möchte, die Schaltfläche *Read Program Memory* angeklickt wird.
Je nach dem Wert der ganzen Zahl, den man im dafür vorgesehenen einzeiligen Textfeld eingetippt hat, dauert es eine bestimmte Zeit, bis die jeweilige Anzeige oben erfolgt.
Zum Löschen des Inhalts im mehrzeiligen Textfeld klickt man die Schaltfläche *New Text*. Das Programm lässt sich durch Betätigen der Schaltfläche *End* beenden, wenn man nicht die zusätzlichen Möglichkeiten, die es bietet, nutzen möchte, indem man die Schaltfläche *Expand Panel* anklickt Diese wurden im Beitrag erläutert.

Bild A8: Screenshot des Python-Programms *prog_EEPROM3.py*



hältnisse um. VT1 sperrt und die Basis von VT2 liegt über die Segmente der LED1 an den Pins PA0 bis PA6 sowie PB2. Wird an mindestens einem dieser Ausgänge High-Pegel abgegebenen, steuert VT2 durch. Der dabei fließende geringe Basis-Strom reicht jedoch nicht, um das entsprechende Segment der LED1 zum Leuchten zu bringen. Die Segmente der LED2 leuchten hingegen entsprechend den an PA0 bis PA6 und PB2 ausgegebenen Pegeln.
Der Umschaltvorgang über PB1 muss so schnell erfolgen, dass das Auge des Betrachters dies nicht erkennt und eine gleichzeitige Darstellung von Zeichen auf beiden Anzeigeelementen wahrnimmt.
In einigen anderen Projekten nutze ich den an PA7 des ATtiny24 vorhandenen A/D-Umsetzer für eine Spannungsanzeige. Die dort angelegte Spannung darf jedoch den Betriebsspannungsbereich des ATtiny24 in beiden Richtungen nicht überschreiten. Für Versuche kann man an X2 den Schleifkontakt eines zwischen VCC und Masse (GND) liegenden 10-kΩ-Potenziometers anschließen.
Die Bilder A1 und A2 zeigt den Probeaufbau der Schaltungen in Bild A7 auf Steckboards. Als Verbindungen zwischen dem Steckboard mit dem ATtiny24 und demjenigen mit den beiden Siebensegmentanzeigen dienen konfektionierte Leitungen. Sie ermöglichen einen schnellen Aufbau.
Die Schaltung in Bild A7 ist auch an anderen AVR-Mikrocontroller, wie ATtiny26, ATmega8, ATmega88 oder ATmega16 anschließbar, wenn man die Zurodnung der Portpins zu den Anschlüssen der Siebensegmentanzeigen anpasst.

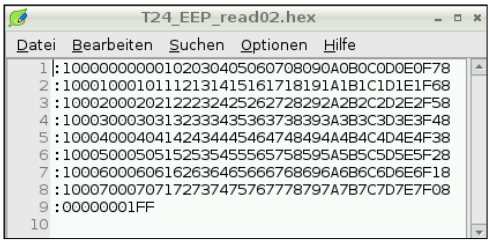


Bild A9: Inhalt der Datei *T24_EEP_read02.hex*

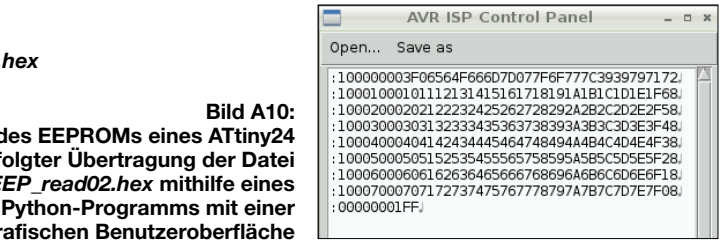


Bild A10: Lesen des EEPROMs eines ATtiny24 nach erfolgter Übertragung der Datei *T24_EEP_read02.hex* mithilfe eines Python-Programms mit einer grafischen Benutzeroberfläche

■ Experimentierschaltung mit zwei Siebensegmentanzeigen

Bild A7 zeigt eine Experimentierschaltung für einen ATtiny24 mit zwei Siebensegment-Anzeigeelementen, die jeweils einen gemeinsamen Katodenanschluss besitzen.

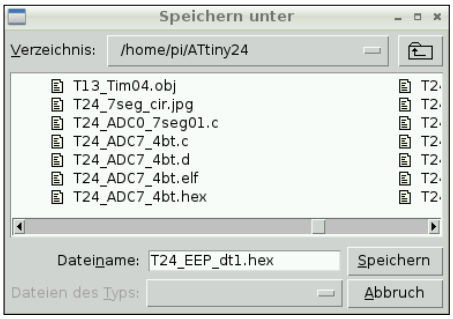


Bild A11: Speichern eines EEPROM-Inhalts nach dem Auslesen mithilfe von *AVR_ISP_contr.py*

Ihre Anschlüsse *a* bis *f* und *dp* sind miteinander und jeweils gemeinsame Anschlüsse über Vorwiderstände von je 4,7 kΩ mit den Portpins PA0 bis PA6 sowie PB2 verbunden. Da nicht genügend Anschlüsse zum direkten Anschluss zweier Siebensegment-Anzeigeelemente vorhanden sind, werden sie über die NPN-Transistoren im Wechsel, also im Multiplexbetrieb, angesteuert.
Liegt High-Pegel an PB1, steuert VT1 durch und zieht den Katoden-Anschluss der LED1 nach Masse. Dann leuchten die Segmente der LED1 entsprechend den an PA0 bis PA6 und PB2 ausgegebenen Pegeln. Während dessen ist der Transistor VT2 gesperrt, da sein Basis-Anschluss ebenfalls auf Massepegel liegt. Liegt Low-Pegel an PB1, kehren sich die Verhältnisse um. VT1 sperrt und die Basis von VT2 liegt über die Segmente der LED1 an den Pins PA0 bis PA6 sowie PB2. Wird an mindestens einem dieser Ausgänge High-Pegel abgegebenen, steuert VT2 durch. Der dabei fließende geringe Basis-Strom reicht jedoch nicht, um das entsprechende Segment der LED1 zum Leuchten zu bringen. Die Segmente der LED2 leuchten hingegen entsprechend den an PA0 bis PA6 und PB2 ausgegebenen Pegeln.
Der Umschaltvorgang über PB1 muss so schnell erfolgen, dass das Auge des Betrachters dies nicht erkennt und eine gleichzeitige Darstellung von Zeichen auf beiden Anzeigeelementen wahrnimmt.
In einigen anderen Projekten nutze ich den an PA7 des ATtiny24 vorhandenen A/D-Umsetzer für eine Spannungsanzeige. Die dort angelegte Spannung darf jedoch den Betriebsspannungsbereich des ATtiny24 in beiden Richtungen nicht überschreiten. Für Versuche kann man an X2 den Schleifkontakt eines zwischen VCC und Masse (GND) liegenden 10-kΩ-Potenziometers anschließen.
Die Bilder A1 und A2 zeigt den Probeaufbau der Schaltungen in Bild A7 auf Steckboards. Als Verbindungen zwischen dem Steckboard mit dem ATtiny24 und demjenigen mit den beiden Siebensegmentanzeigen dienen konfektionierte Leitungen. Sie ermöglichen einen schnellen Aufbau.
Die Schaltung in Bild A7 ist auch an anderen AVR-Mikrocontroller, wie ATtiny26, ATmega8, ATmega88 oder ATmega16 anschließbar, wenn man die Zurodnung der Portpins zu den Anschlüssen der Siebensegmentanzeigen anpasst.

Bild A12 zeigt einen Ausschnitt aus dem Hauptprogramm *T24_EEP02.c*. Nach der Übertragung der von GCC-AVR erstellten Hex-Datei in den ATtiny24 werden die 16 Bytes für die Darstellung der Zeichen 0 bis 9 sowie A bis F in den EEPROM des Mikrocontrollers übertragen und darauf in einer Endlosschleife die Inhalte der EEPROM-Speicherplätze 0 bis 15 nacheinander mit zwischengeschalteten Pausen in sehr schneller Folge auf den beiden Siebensegmentanzeigen in Bild A7 im Hex-Format dargestellt. Für das Auge des Betrachters erfolgt die Ausgabe gleichzeitig.

h_nieder@arcor.de

Literatur

[1] Nieder, H., DL6PH: Programmierung von AVR-Controllern mit dem Raspberry Pi. FUNKAMATEUR 65 (2016) H. 2, S. 132–135

```
unsigned char EE_byte,H_nib,L_nib;
while(1)
{
for (i=0; i<16; i++)// for 1, 16 in das EEPROM zuvor
                    // uebertragene Bytes als zwei
                    // HEX-Zeichen anzeigen
{
    EE_byte=EEPROM_read(i);
    L_nib=(EE_byte & 0x0F);
    H_nib=((EE_byte & 0xF0)>>4);
    for (j=0;j<100;j++)// for 2, 100 mal beide  Anzeigen
{                      // abwechselnd einschalten
    clearbit(PORTB,PB1);//Anzeige rechts ein
    w=L_nib;
    PORTA=anz_werte[w];
    _delay_ms(10);
    setbit(PORTB,PB1);  //Anzeige links ein
    w=H_nib;
    PORTA=anz_werte[w];
    _delay_ms(10);
}//Ende for 1
}//Ende for 2
}//Ende while(1)
```

Bild A12: Ausschnitt aus dem Programmlisting *T24_EEP02.c*

Fotos, Screenshots: DL6PH