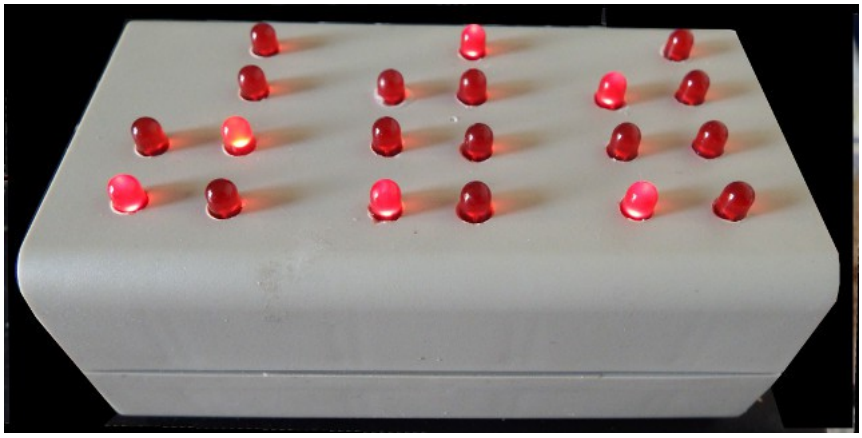
Eine Alternative zu Dicke-Tal-Uhren: die Dünne-Berg-Uhr

Überblick



1 BCD-Anzeige als Alternative zu 7-Segmentanzeigen

Schon die allerersten Computer, die noch Relais- oder Röhrenrechner hießen, hatten kleine Glühlämpchen, mit denen der Programmierer im Einzelschritt überprüfen konnte, was der Rechner so gerade vor sich hin rechnet. Auf diese Weise blinkte es gar lustig, wenn der Rechner arbeitete. Und wenn der Operator den Rechner anhielt, um zu schauen, was er gerade jetzt so macht, konnte er den Lämpchen entnehmen, mit was der Rechner gerade jetzt so beschäftigt war:

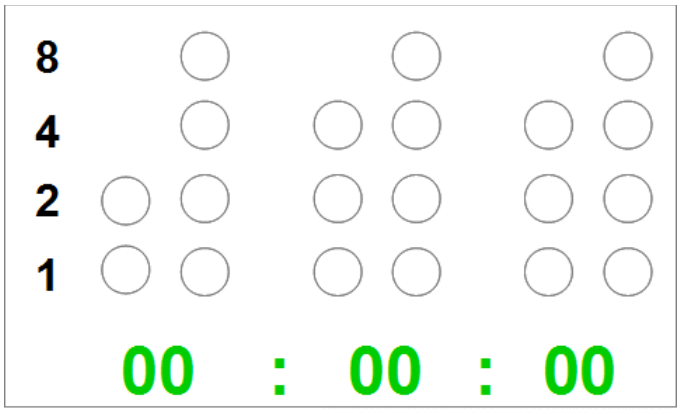
1. 12 oder 16 Lämpchen zeigten die Adresse an, die gerade bearbeitet wurde oder die als nächstes dran war.
2. 8 Lämpchen zeigten den Befehl an, den der Rechner als nächtes ausführen würde, wenn man ihn ließe.
3. 8 Lämpchen zeigten den Akkumulator an, das letzte Resultat der Rechnerei.

Das Einzelschrittverfahren war zwar mühsam, behielt aber auch in komplexen Situationen die Übersicht.

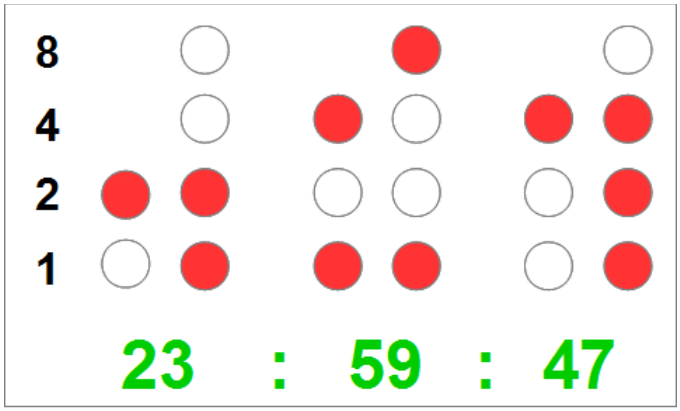
Das Konzept dieser Uhr geht in diese Anfangszeiten zurück: die Zahlen werden binär angezeigt, müssen zusammengezählt werden und werden nicht etwa auf krummen Anzeigen fertig serviert. Das Ablesen erfordert etwas binären Grips.

So sieht die Uhr um 24:00 Uhr aus:

Alle Lampen von der Uhr sind aus. Es ist herrlich dunkel, aber nur für eine Sekunde lang.



Jetzt so eine andere Uhrzeit:



Jetzt kommen die Zahlenwerte links, die Zweierpotenzen, zum Einsatz. Bei den Zehnern der Stunde ist die 2-Lampe an, also sind die Zehner 2. Bei den Einern der Stunde ist die 1-Lampe und die 2-Lampe an, macht zusammen 3. Selbiges bei den weiteren Ziffern.

Fehlende Lampen bei den Zehnerstellen sind auf das bekloppte 60-er-System bei der Uhrzeit zurückzuführen. Die 60 (das "Schock") und die 12 (das "Dutzend") waren mal magische Erfindungen von ein paar abstrusen Menschen, die eigentlich im heutigen Dezimalsystem und auch in der Computerei nix mehr zu suchen haben und nur Ärger machen, und die ansonsten nur noch bei der englischen Währung ihre überkommenen Urständ fristen und eigentlich konsequent abgeschafft gehören. Und halt auch noch bei der Uhrzeit.

Nun vergleichen wir dieses Anzeigesystem mal mit dem herkömmlichen Sieben-Segment-Zirkus bei Uhren. Um die Uhrzeit mit dieser Auflösung darzustellen, bräuchten wir sechs solcher Anzeigen mit 42 LEDs drin. Wir kommen mit weniger als der Hälfte aus: unsere Uhr hat gerade mal 20 davon. Ressourcen und Stromverbrauch mal eben halbiert, wenn das kein Argument für die Dünne-Berg- gegen die Dicke-Tal-Uhr ist.

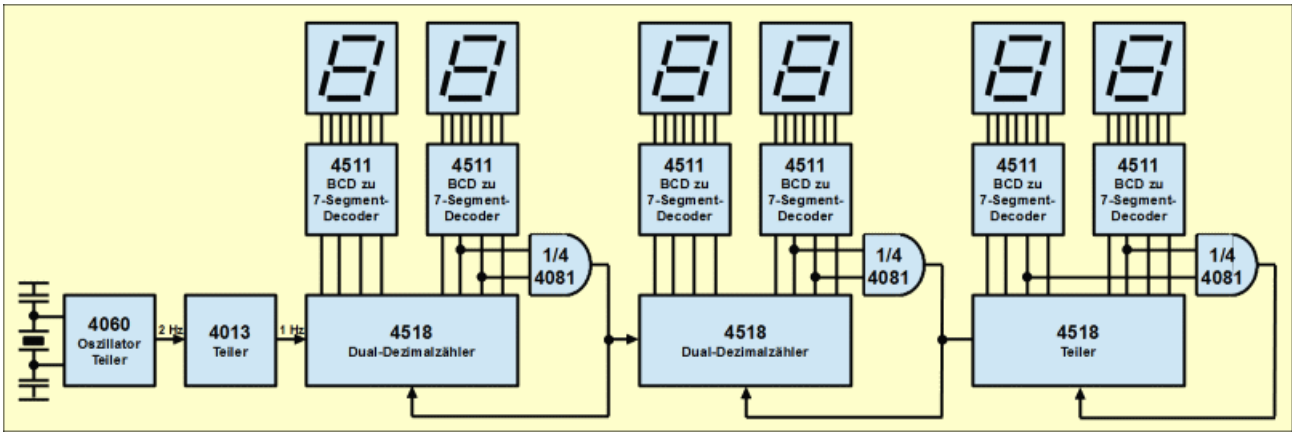
Unsere sechs Ziffern heißen bei Informatikers übrigens BCD (Binary Coded Digit), so was wie binär-kodierte Ziffern. Wir werden im Kapitel Software noch sehen, dass das hervorragendes Futter für Microcontroller ist und dass er den Umgang mit solchen Ziffern glatt von selbst beherrscht.

BCD-Anzeige	Prozessor	Netzteil	Aufbau	Software	Bedienanleitung

2 Der Prozessor

2.1 Warum ein Prozessor?

Zuerst mal die klassische Variante, eine CMOS-Uhr:



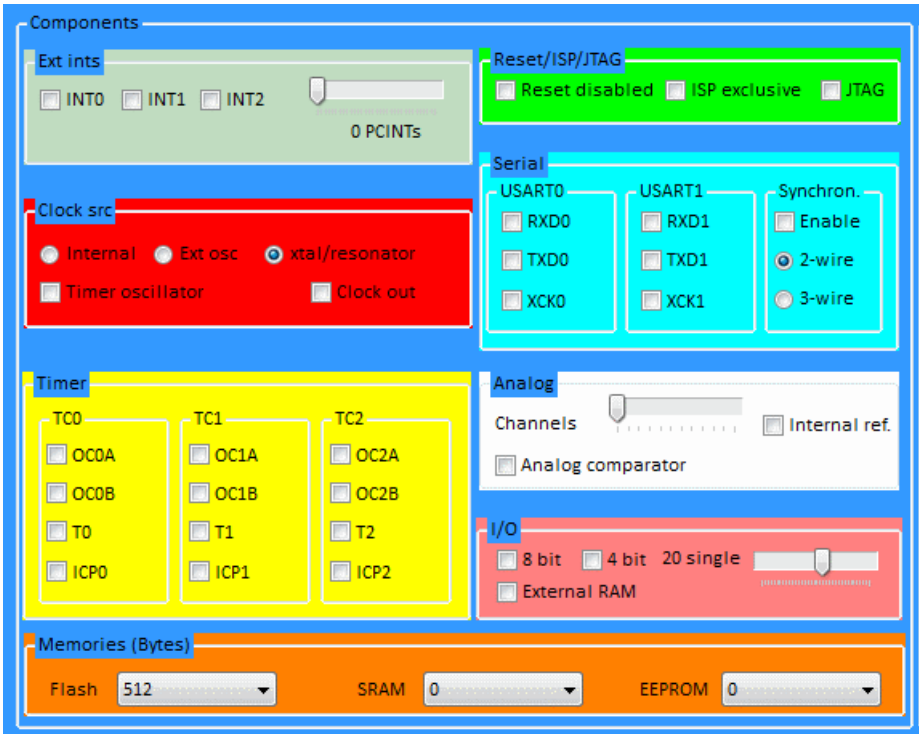
Ganze 12 ICs, 42 Widerstände und sechs Siebensegmentanzeigen dürften eine ganze Europlatine füllen. Plus Dutzende von Leiterbahnen. Ein echtes CMOS-Grab, teuer und kompliziert.

Das alles (bis auf die LEDs und die Widerstände) ist in so einem Mikrocontroller schon enthalten und braucht bloß per Software (Timer) und Fuses (Quarzoszillator) aktiviert wer-

den. Aber halt nicht in einem nackigen ATmega8, der tut ohne eingebranntes Programm rein gar nix.

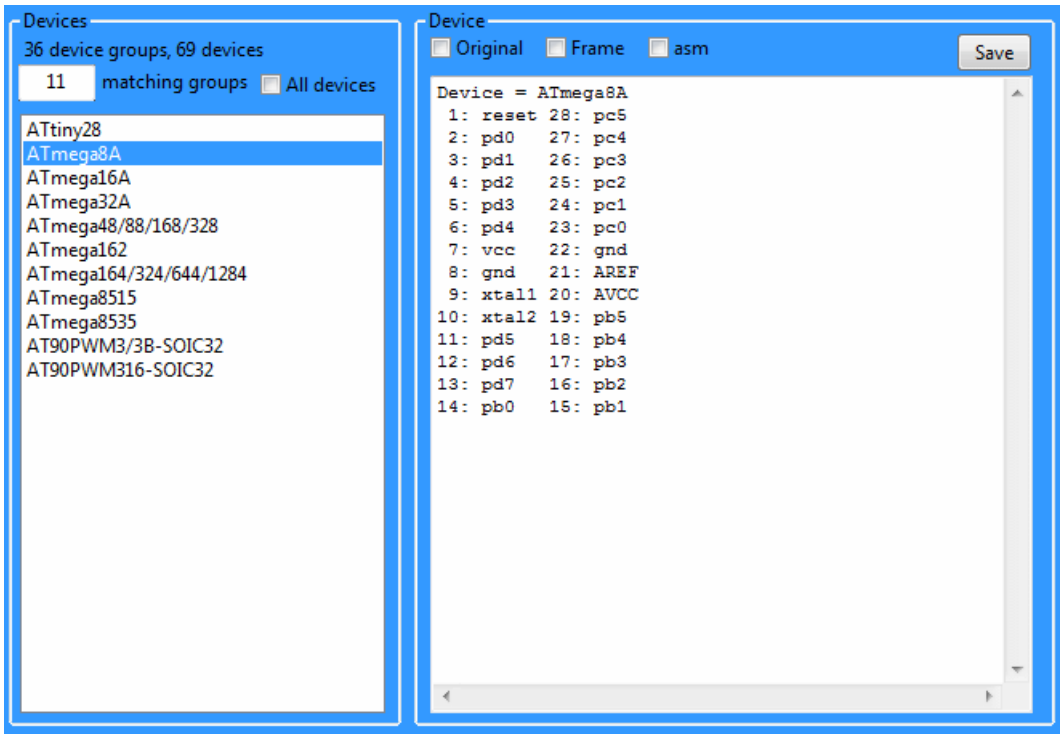
2.2 Prozessorauswahl

Der moderne AVR-Designer greift jetzt zum Arduino, aber nur weil er gar nix anderes kennt und daher an einer seltsamen Fixierung leidet, so Kokain-mäßig. Und taucht voll in die Suche nach passenden Bibliotheken ein, die ihm die eigene Denkarbeit abnehmen und stattdessen nach Lösungen suchen lassen, die dann halt doch nicht wirklich auf sein Problem passen. Aber irgendwie kriegt er das mit seinem eingebauten ATmega324 schon hin, nachdem er schon acht Kompromisse eingegangen ist oder 15 Zeilen Inline-Assembler für die BCD-Mathematik hineingefriemelt hat, die er in keiner Bibliothek findet.



Die anderen, nicht abhängigen Menschen starten den Windows- oder den Linux-AVR-Selektor und wählen in der Sektion "Clock src" einen Quarzoszillator und in der Sektion "I/O" 20 Ein-/Ausgabe-Pins.

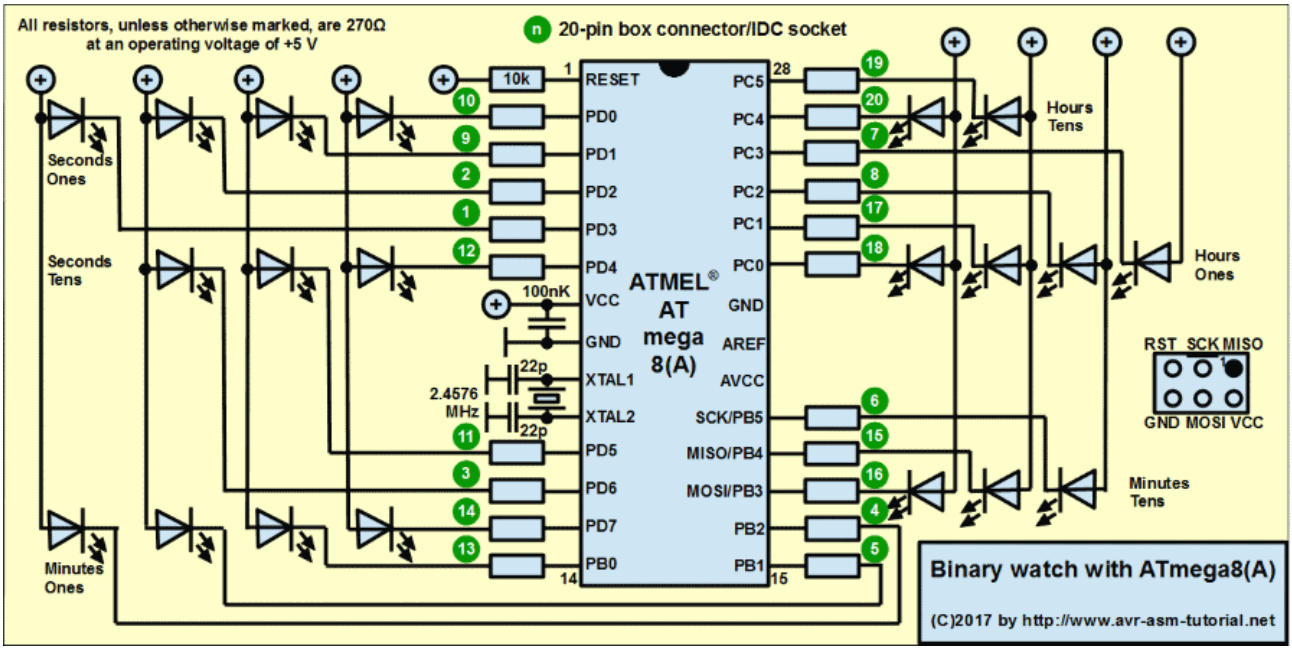
Eine Reihe von AVR-Typen kann das. Wir wählen den bei Reichelt leicht erhältlichen ATmega8 in irgendeiner der erhältlichen Unterarten (mit nix, mit A, mit ...), auch wenn manche Angeber meinen, der sei total out und werde schon gar nicht mehr produziert (es ist manchmal amüsant, in die vom Arduino total verbogene Wahrnehmungswelt mancher Zeitgenossen einzutauchen).



2.3 Schaltbild

So sieht unser Schaltbild mit dem ATmega8 aus, alles easy: aus 12 ICs wird eines mit 28 Pins und aus 42 Widerständen wird glatt die Hälfte. Und noch dazu total easy lieferbar.

Die zusätzliche Anforderung, dass die Software nicht allzu kompliziert sein soll, bringt folgendes Schaltbild zutage.



Für den Quarz mit 2,4576 MHz kann auch jeder andere Typ mit fast beliebiger Frequenz eingesetzt werden, der durch 1.024 geteilt und mit einem ganzzahligen Teiler bis 65.536 eine Eins ohne Nachkommastellen liefert. Er ist mit Keramikkondensatoren von 22 pF zum Schwingen angeregt.

Die sechs BCD-Ziffern sind an die Ports D, B und C angeschlossen. Alle Portanschlüsse passen zu den Anzeigenreihen, nur das Portbit PD7 überlappt ein wenig, was aber per Software mit einfachem Links- und Rechtsschieben leicht handhabbar ist.

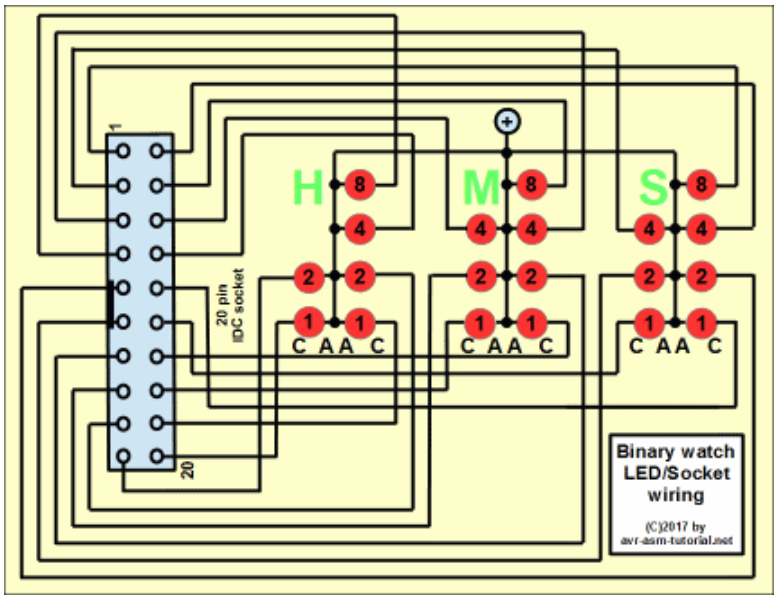
Zum Programmieren in der Zielschaltung ist noch eine ein 6-poliger ISP-Stecker eingebaut.

In dieser Schaltung sind alle LEDs in Sink-Schaltung angeschlossen. Bei 5 V Betriebsspannung beträgt der LED-Strom

$$I_{LED} = (5 - 2 - 0{,}2) / 270 * 1.000$$
$$= 10{,}4 \text{ mA}$$

Wären alle 20 LEDs an, betrüge der Strom 207 mA. Der ATmega8 ist mit maximal 300 mA Strom über den GND-Pin spezifiziert, kann also so betrieben werden.

Um den Aufbau zu vereinfachen, ist noch ein 20-poliger Wannenstecker eingebaut, der die 20 LEDs mit der Prozessorplatine verbindet. Die Anschlussfolge des Steckers orientiert sich an einer vereinfachten LED-Verdrahtung mit den En-



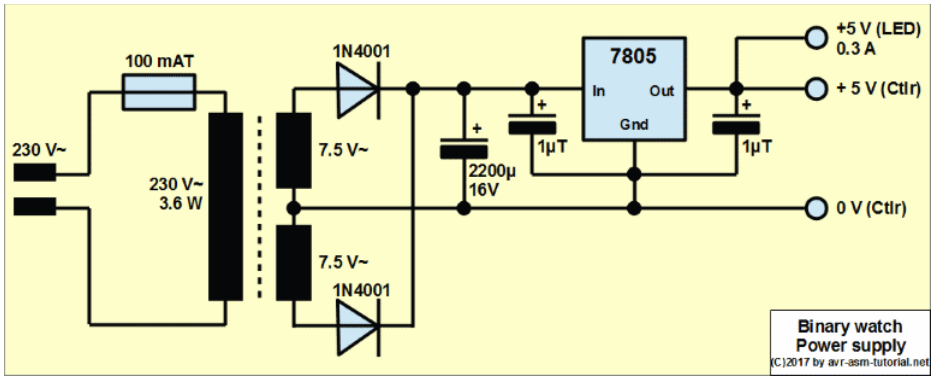
den des eingesteckten Flachkabels.

Die Kathoden der Achter- und Vierer-LEDs kommen an Pin 1 bis 8 der Flachkabelbuchse, die der Zweier und Einer an die Pins 9 bis 20. Alle Anoden der LEDs sind verbunden und gehen an ein separates Kabel direkt zum Netzteil.

3 Das Netzteil

3.1 Schaltbild

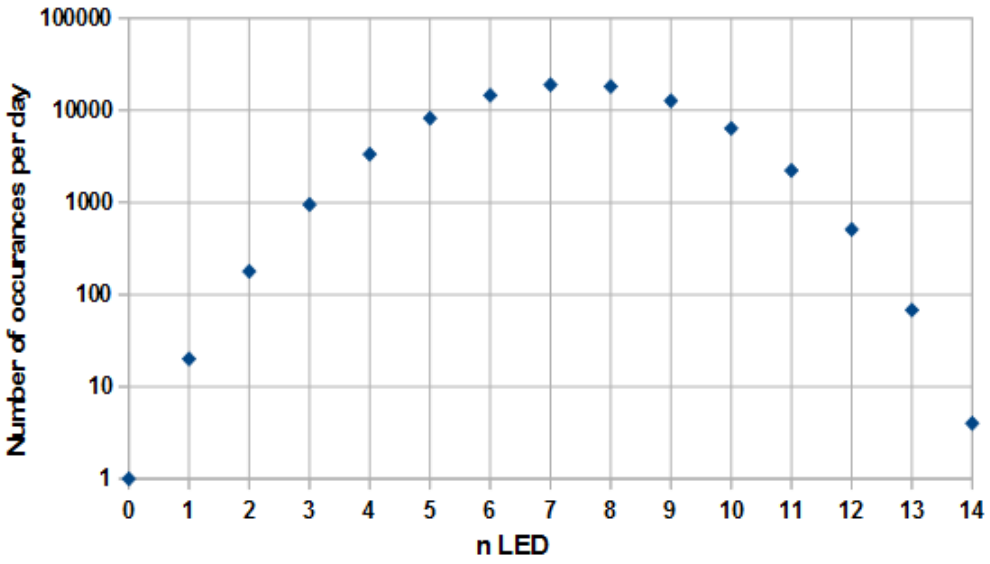
Dies hier ist das Schaltbild des Netzteils.



Das ganze passt auf eine kleine Lochrasterplatine, die perfekt in das gewählte Gehäuse passt.

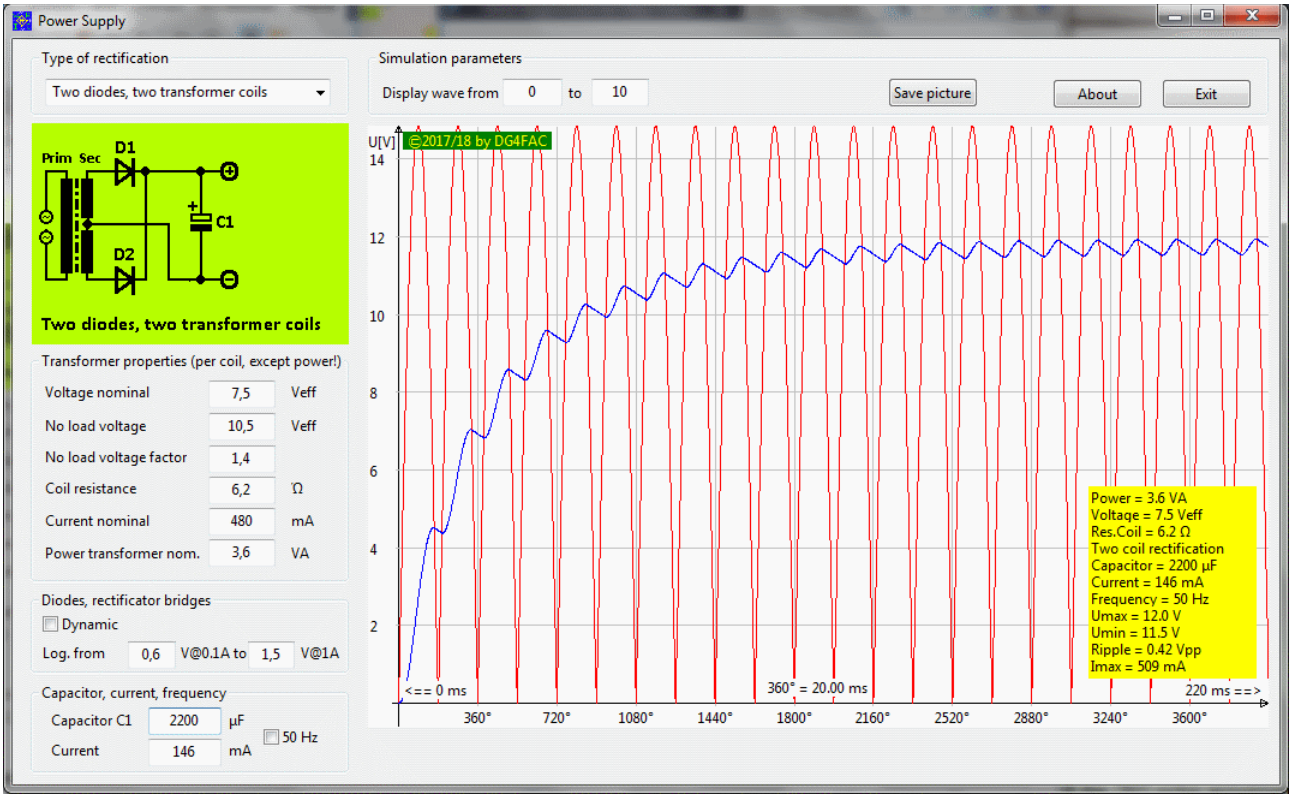
3.2 Dimensionierung

Wie oben beschrieben brauchen wir für die LEDs maximal knapp über 200 mA. Um den realen Wert zu ermitteln, wurden in einer Tabellenkalkulation alle 86.399 Sekunden eines Tages simuliert und die Anzahl LEDs berechnet. Maximal sind gleichzeitig 14 LEDs an (z.B. um 17:37:37 und um 17:57:37). Es ergibt sich die nebenstehende Verteilung.

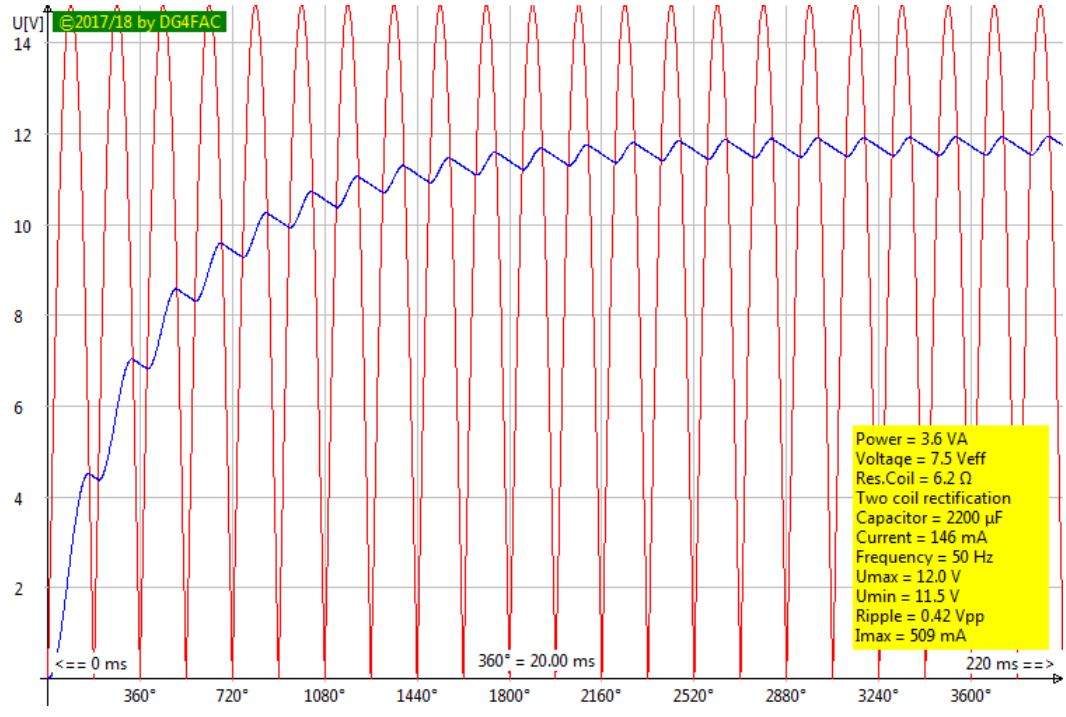


Im Schnitt sind 7,3 LEDs gleichzeitig an, der maximale Minutenschnitt beträgt 11,7 LEDs. Das sind bei 10,4 mA LED-Strom im Schnitt 76 mA, maximal für eine Minute lang 122 mA und maximal für eine Sekunde lang 146 mA. Also deutlich weniger als der unrealistische 20-LED-Fall.

In der Bastelkiste fand sich ein 2*7,5V-Trafo mit 3,6 VA. Um zu testen, ob der geeignet wäre, wurde die Power-Supply-Software hier angeworfen und das Trafonetzteil gerechnet.

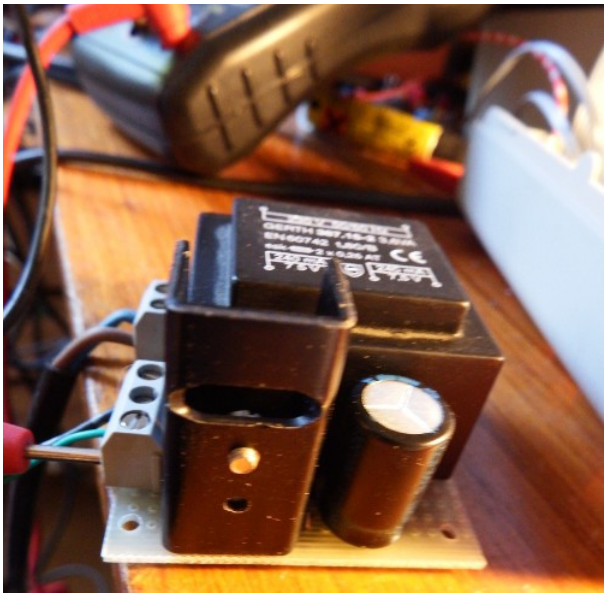


Es wird Zwei-Dioden-zwei-Spulen-Gleichrichtung gewählt, in die Felder mit dem Transformer 7,5 V, 3,6 VA und 6,2 Ω in die Felder mit dem Elko 2.200 µF und in das Feld mit dem Strom 146 mA eingetragen. Und schon produziert die Software die Kurve mit was das Netzteil so arbeitet.



Das Netzteil produziert aus dem 7,5-V-Trafo 12 V Spitzenspannung bei 0,42 V_{pp} Brummspannung. Das ist vollkommen ausreichend, um den 5V-Regler korrekt auszusteuern. Da bei 146 mA 7 V am Spannungsregler hängen bleiben und das mit P = U * I etwas mehr als 1 W Wärmelast ist, kriegt der 7805 einen kleinen Kühlkörper mit 20 K/W.

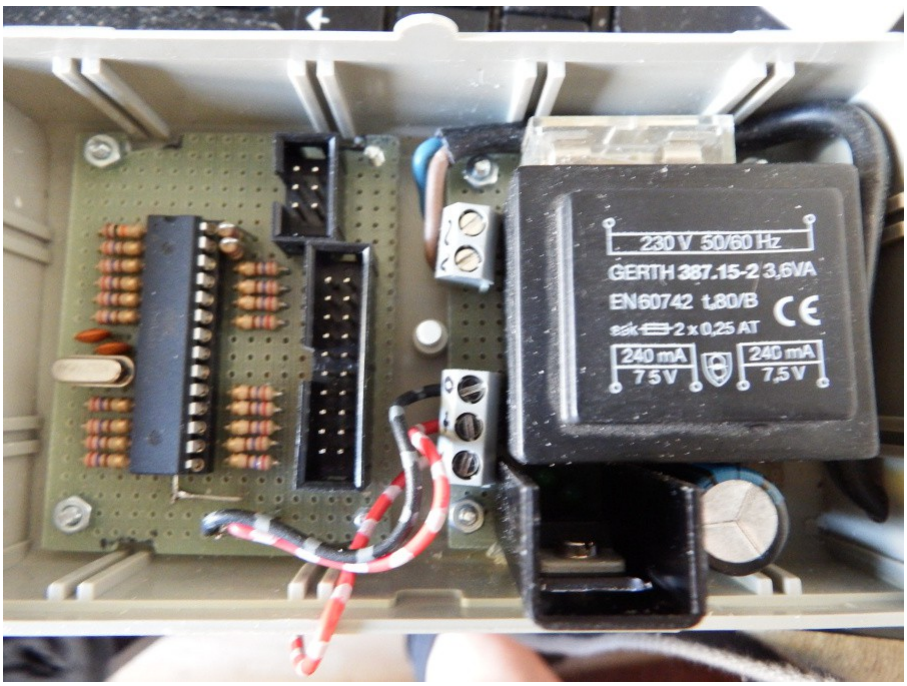
3.3 Lasttest



Um sicherzustellen, dass das Netzteil auch extreme Ausnahmezustände übersteht (z.B. Hitzeperioden), habe ich es mit 300 mA Dauerlast (2 Widerstände mit 33Ω-2W parallel) mehrere Stunden gequält. Der Trafo und der 7805-Kühlkörper werden dabei recht heiß, aber noch erträglich.

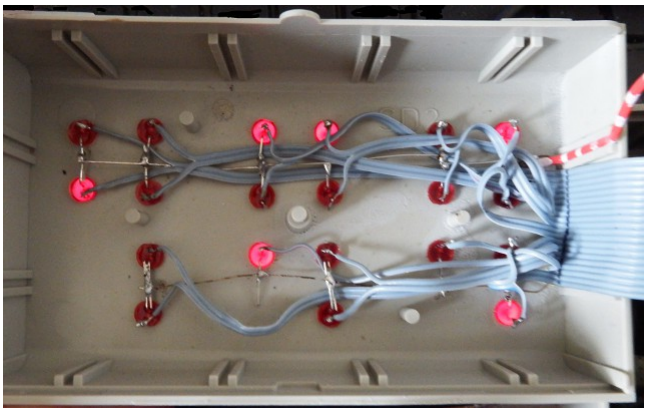
Auch bei 400 mA ist das Ribble noch erträglich (1,04 V_{pp}) und auf der geregelten 5V-Spannung tut sich nix.

4 Der Aufbau



Der Aufbau erfolgt auf Lochrasterplatinen, die passgerecht in das Gehäuse passen.

Der Vorteil des 20-poligen LED-Anschlusses ist, dass Netzteil und Prozessorteil im unteren Gehäuseteil platziert werden können. Der LED-Teil im Deckel sieht dann so aus.



Die Verdrahtung der LEDs ist durch die Steckerbelegung sehr vereinfacht. Dafür ist dann die Verdrahtung des Steckers auf der Prozessorplatine etwas komplizierter. Die erste Version hatte zwei Verdrahtungsfehler.

5 Die Software

Die Software ist in Assembler-Format hier, im Anhang hier.

5.1 Aufbau und Struktur

Die Software ist in folgenden Schritten aufgebaut:

1. Initiierung der Hardware,
2. Timer, Interrupts und Schlafmodus,
3. Sekundeninterrupt und Uhrzeit,
4. Umwandeln in das Anzeigeformat und Portausgabe,
5. Quarzoszillator mit Fuse einschalten.

5.2 Initiierung

Da das Programm den Interruptmodus verwendet, muss der Stapel initiiert werden. Da der ATmega8 über mehr als 256 Byte SRAM verfügt, sind Low- und High-Byte des Stapelzeigers zu setzen.

Alle Portpins sind dann als Ausgänge umzuschalten, indem deren Richtungsports lauter Einsen erhalten. Alle Ports sind beim Init auf High zu schalten, damit der Fall mit den annenen 20 LEDs gar nicht erst eintritt.

5.3 Timer, Interrupts und Schlafmodus

Der Timer arbeitet mit einem Vorteiler von 1024 und im CTC-Modus (Rücksetzen bei Erreichen des Compare-Match-A-Wertes). Die Compare-Match-Werte sind von der Quarzfrequenz abhängig. Die Werte sollten mit Ganzzahlenwerten Eins ergeben. Geeignete handelsübliche Quarze sind (Quarze oberhalb 10 MHz sind für die V/L-Versionen des ATmega8 oder bei Betriebsspannungen unter 5 Volt ungeeignet):

Passende Quarze (MHz):		
< 5 MHz	5 – 10 MHz	> 10 MHz
2,048	5,0688	10,24
2,097152	5,12	11,0592
2,4576	6,144	12,288
3,072	6,4	14,7456
3,2768	6,5536	16
3,93216	7,3728	18,432
4,096	7,68	19,6608
4,194304	8,388608	
4,9152	9,216	
	9,8304	

Wird ein anderer Quarz als 2,4576 MHz verwendet, ist die Quarzfrequenz in der Konstante clock zu ändern.

In jedem Fall muss die Fuse geändert werden (siehe unten).

Nach dem Starten des Timers im Port TCCR1B ist noch der Compare-Match-Interrupt einzuschalten. Damit der Controller beim Zählen alle anderen Arbeiten einstellt, wird der

Sleep-Mode Idle eingestellt, nur der Interrupt weckt den Prozessor wieder auf. Die Ausführung von Interrupts sind durch Setzen des Interrupt-Flags im Statusregister einzuschalten.

Nach dem Aufwecken wird der Controller einfach wieder schlafen gelegt.

5.4 Sekundeninterrupt und Uhrzeit

Da kein anderer Interrupt zu bearbeiten und auch sonst nichts weiteres zu tun ist, findet der gesamte Ablauf im Sekundeninterrupt statt. Die Software enthält dafür zwei verschiedene Versionen, die mit dem Schalter pbcd ausgewählt werden. Ist dieser auf 1 gesetzt, findet die Berechnung der Uhrzeit mit gepackten BCD-Zahlen statt, Wenn nicht mit binären Zahlen.

Bei gepackten BCD-Zahlen enthalten die Register rSek, rMin und rStd in den unteren vier Bits die Einer, in den oberen vier Bits die Zehner. Um zu erkennen, ob die nächste Sekunde im unteren Nibble mehr als 9 Sekunden erreicht hat, wird 6 zum unteren Nibble dazu addiert. Tritt dabei ein Übertrag vom unteren in das obere Nibble auf, ist die H-Flagge (Halbübertrag) im Statusregister gesetzt. Falls nicht, werden die 6 wieder abgezogen. Sekunden und Minuten werden mit 0x60, Stunden mit 0x24 verglichen. Sind diese erreicht, wird das betreffende Register auf Null rückgesetzt und die nächste Minute oder Stunde angestoßen.

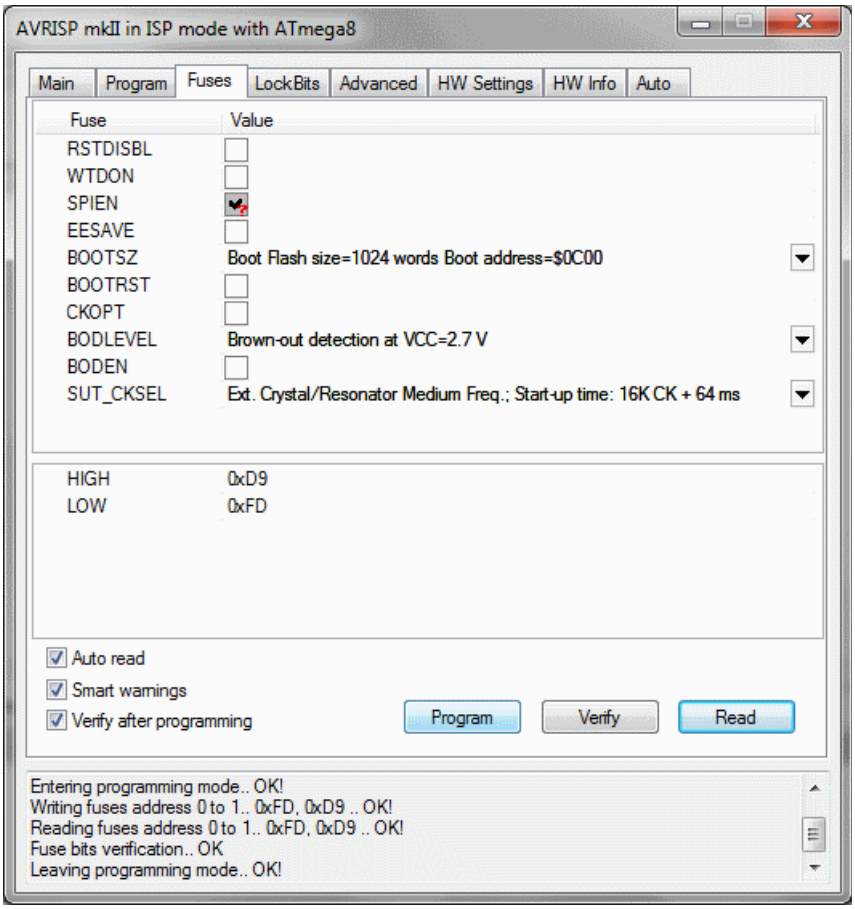
Im binären Modus werden die Register rSek, rMin und rStd so lange erhöht, bis ihr Inhalt 60 bzw. 24 ist und das Rücksetzen erfolgt (bei Sekunden und Minuten mit Übertrag auf das nächste Register).

5.5 Umwandeln in das Anzeigeformat und Portausgabe

Im gepackten BCD-Modus ist die Ausgabe am einfachsten: die Ports D, B und C kriegen direkt die gepackte BCD-Zahl. Nur das unterste Bit von B wird aus dem Port heraus- und in den Port D als oberstes Bit hereingeschoben.

Beim binären Uhrzeitmodus muss die Binärzahl zuerst durch zehn dividiert werden. Das Divisionsergebnis geht dann in das obere Nibble, der Divisionsrest in das untere Nibble. Damit hat man dann gepacktes BCD. Bit 0 aus Port B wird dann wie oben beschrieben in Port D kopiert.

Bei der Ausgabe werden die drei Portergebnisse mit COM invertiert, da die Anoden der LEDs an +5V angeschlossen sind und leuchten, wenn der Portausgang auf Null geht.



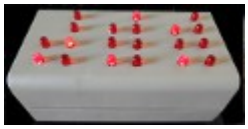
5.6 Quarzoszillator einschalten

Um den Quarzoszillator als Taktquelle einzuschalten, ist die Fuse wie im Bild gezeigt zu setzen.

6 Die Bedienanleitung

Wenn die Dünne-Berg-Uhr als Geschenk zu verschenken ist, muss eine ordentliche Bedienungsanleitung her. Ich habe da mal einen Entwurf für OpenOffice in maschinell aus Chinesisch in Deutsch übersetzter Version (Kennzeichen: chinesische Rechner haben keine Umlaute) gemacht und daraus ein PDF produziert.

BCD-Anzeige	Prozessor	Netzteil	Aufbau	Software	Bedienanleitung



Quellcode für die Dünne-Berg-Uhr

Originalcode in Assembler-Format ist hier.

```
;
; ******************************************
; * Binaere Uhr mit 20 LEDs und ATmega8A   *
; * Version 1, April 2017                  *
; * (C)2017 by http://www.avr-asm-tutorial.net *
; ******************************************
;
.NOLIST
.INCLUDE "m8Adef.inc" ; Headerdatei fuer ATMEGA8A
.LIST
;
; ====================================
;  A S S E M B L E R - S C H A L T E R
; ====================================
;
; Diese Assemblerschalter fuer normalen Ablauf = 0
.equ speed = 0 ; 0 = normal, 1 = Minuten testen, 2 = Stunden testen
.equ fast = 0 ; Schneller Ablauf fuer Debug (Prescaler=1)
;
; Dieser Schalter ist Geschmackssache, pbcd=1 ist schneller
.equ pbcd = 1 ; 1 = Uhrzeit in Packed BCD, 0 = Binaer
;
; ==========================================
;   H A R D W A R E I N F O R M A T I O N E N
; ==========================================
;
; ATmega8A
;                  _________________
;  ISP6+   1 /                 |28
;  RESET o--|reset          PC5|--o StdZ2
;          2|                  |27
;  SekE1 o--|PD0            PC4|--o StdZ1
;          3|                  |26
;  SekE2 o--|PD1            PC3|--o StdE8
;          4|                  |25
;  SekE4 o--|PD2            PC2|--o StdE4
;          5|                  |24
;  SekE8 o--|PD3            PC1|--o StdE2
;          6|                  |23
;  SekZ1 o--|PD4            PC0|--o StdE1
;          7|                  |22
;   +5 V o--|vcc            gnd|--o NC
;          8|                  |21
;    0 V o--|gnd           AREF|--o NC
;          9|                  |20
;  Quarz o--|XTAL1         AVCC|--o NC
; 2,4576  10|                  |19
;   MHz  o--|XTAL2      PB5/SCK|--o MinZ4+ISP6-SCK
;         11|                  |18
;  SekZ2 o--|PD5       PB4/MISO|--o MinZ2+ISP6-MISO
;         12|                  |17
;  SekZ4 o--|PD6       PB3/MOSI|--o MinZ1+ISP6-MOSI
;         13|                  |16
;  MinE1 o--|PD7            PB2|--o MinE8
;         14|                  |15
;  MinE2 o--|PB0            PB1|--o MinE4
;           |__________________|
;
```

```
;
; ================================================
; K O N S T A N T E N   Z U M  E I N S T E L L E N
; ================================================
;
.equ clock = 2457600 ; Taktfrequenz, externer Quarz
;
; ======================================================
;  F E S T E + A B G E L E I T E T E  K O N S T A N T E N
; ======================================================
;
.if speed == 0 ; normale Geschwindigkeit
  .equ cPresc = 1024 ; Teiler fuer Timer 1
  .equ cCtc = clock/cPresc - 1 ; Teiler Timer 1
  .elif speed == 1 ; Minutentest
  .equ cPresc = 64
  .equ cCtc = clock/cPresc/60 - 1
  .else ; Stundentest
  .equ cPresc = 1
  .equ cCtc = (clock/cPresc+1800)/3600 - 1
  .endif
;
; ============================================
;   R E G I S T E R D E F I N I T I O N E N
; ============================================
;
; frei: R0 bis R15
.def rmp1 = R16 ; Vielzweckregister 1
.def rmp2 = R17 ; Vielzweckregister 2
.def rSek = R18 ; Sekunden
.def rMin = R19 ; Minuten
.def rStd = R20 ; Stunden
.def rPD = R21  ; Portausgabe Register Port D
.def rPB = R22  ; dto., Port B
.def rPC = R23  ; dto., Port C
; frei: R24 bis R31
;
; ============================================
;       S R A M   D E F I N I T I O N E N
; ============================================
;
.DSEG
.ORG  0X0060
; (SRAM nur fuer Interrupt und Call-Stack verwendet)
;
; ============================================
;   R E S E T   U N D   I N T   V E K T O R E N
; ============================================
;
.CSEG
.ORG $0000
        rjmp Main ; Reset-Vektor
        reti ; INT0
        reti ; INT1
        reti ; TC2COMP
        reti ; TC2OVF
        reti ; TC1CAPT
        rjmp SekInt ; TC1COMPA, Sekundeninterrupt
        reti ; TC1COMPB
        reti ; TC1OVF
        reti ; TC0OVF
        reti ; SPI_STC
        reti ; UART-RX
        reti ; UART-UDRE
        reti ; UART-TXC
        reti ; ADC
        reti ; EERDY
        reti ; ANACOMP
        reti ; TWI
        reti ; SPMRDY
;
```

```
; ========================================
;    I N T E R R U P T   S E R V I C E
; ========================================
;
; Sekunden-Interrupt
; Da keine anderen Interrupts und keine
;   weiteren Ablaeufe zu bearbeiten sind,
;   erfolgt der gesamte Ablauf innerhalb
;   der Interrupt-Service-Routine
SekInt: ; Sekunden-Interrupt
        inc rSek ; naechste Sekunde
.if pbcd == 1
; Packed-BCD
        ldi rmp1,0x06 ; fuer Packed-BCD-Halbuebertrag
        add rSek,rmp1 ; Halbuebertrag?
        brhs SekInt1 ; Ja, teste Sekunden=60
        sub rSek,rmp1 ; kein Halbuebertrag, ziehe wieder ab
        rjmp SekInt4 ; fertig zur Ausgabe
SekInt1:
        cpi rSek,0x60 ; 60 Sekunden?
        brcs SekInt4 ; Nein, zur Ausgabe
        clr rSek ; Sekunden = 0
        inc rMin ; naechste Minute
        add rMin,rmp1 ; Halbuebertrag?
        brhs SekInt2 ; ja, teste 60 Minuten
        sub rMin,rmp1 ; Nein, wieder abziehen
        rjmp SekInt4 ; Zur Ausgabe
SekInt2:
        cpi rMin,0x60 ; 60 Minuten?
        brcs SekInt4 ; Nein, zur Ausgabe
        clr rMin ; Minuten = 0
        inc rStd ; naechste Stunde
        add rStd,rmp1 ; Halbuebertrag?
        brhs SekInt3 ; Ja, teste Stunden=24
        sub rStd,rmp1 ; nein, ziehe wieder ab
        rjmp SekInt4 ; zur Ausgabe
SekInt3:
        cpi rStd,0x24 ; 24 Stunden?
        brcs SekInt4 ; nein, zur Ausgabe
        clr rStd ; Stunden auf Null
SekInt4: ; Ausgabe vorbereiten
        mov rPC,rStd ; Port C sind die Stunden
        mov rPD,rSek ; Port D sind die Sekunden
        lsl rPD ; oberstes Bit herausrollen
        mov rPB,rMin ; Port B sind die Minuten
        lsr rPB ; unterstes Bit herausrollen
        ror rPD ; und in oberstes Bit Port D hinein
        com rPD ; Portbits umkehren wegen LED-Polaritaet
        com rPB
        andi rPB,0x3F ; oberste Bits maskieren
        com rPC
        andi rPB,0x3F ; oberste Bits maskieren
        ; Ausgabe an Ports
        out PORTD,rPD
        out PORTB,rPB
        out PORTC,rPC
        reti ; fertig
        .else
        ; Binaere Uhrzeit in drei Registern
        cpi rSek,60 ; 60 Sekunden?
        brcs SekInt1 ; nein, Ausgabe
        clr rSek ; ja, Sekunden auf Null
        inc rMin ; naechste Minute
        cpi rMin,60 ; 60 Minuten?
        brcs SekInt1 ; nein, Ausgabe
        clr rMin ; ja, Minuten auf Null
        inc rStd ; naechste Stunde
        cpi rStd,24 ; 24 Stunden?
        brcs SekInt1 ; nein, Ausgabe
        clr rStd ; ja, Stunden auf Null
SekInt1: ; Ausgabe vorbereiten
```

```
        mov rmp1,rSek ; Sekunden in Register
        rcall Div10 ; durch 10 dividieren
        mov rPD,rmp1 ; Zehnerrest in Portregister D
        swap rmp2 ; Zehner in oberes Nibble
        or rPD,rmp2 ; Oberes und unteres Nibble kombinieren
        ; Minuten
        mov rmp1,rMin ; Minuten in Register
        rcall Div10 ; Division durch 10
        lsr rmp1 ; Einer rechts schieben
        brcc SekInt2 ; Kein Carry, Sekundenregister ok
        sbr rPD,0x80 ; setze oberstes Bit Sekundenregister
SekInt2:
        lsl rmp2 ; Schiebe Zehner drei Mal links
        lsl rmp2
        lsl rmp2
        or rmp1,rmp2 ; Kombiniere mit unteren Bits
        mov rPB,rmp1 ; und in Portregister B
        ; Stunden
        mov rmp1,rStd ; Stunden in Register
        rcall Div10 ; Division durch 10
        swap rmp2 ; Zehner in oberes Nibble
        or rmp1,rmp2 ; oberes und unteres Nibble kombinieren
        mov rPC,rmp1 ; und in Portregister C
        ; Invertieren vor der Ausgabe wegen LED-Polaritaet
        com rPD
        com rPB
        andi rPB,0x3F ; obere Bits ausblenden
        com rPC
        andi rPC,0x3F ; obere Bits ausblenden
        ; Portregister auf Ports ausgeben
        out PORTD,rPD
        out PORTB,rPB
        out PORTC,rPC
        reti
;
; Dividiere rmp1 durch 10
; Ergebnis in rmp2, Rest in rmp1
Div10:
        clr rmp2 ; Divisionsergebnis leeren
Div10a:
        subi rmp1,10 ; Zehn abziehen
        brcs Div10b ; Carry, fertig
        inc rmp2 ; Ergebnis erhoehen
        rjmp Div10a ; und weiter dividieren
Div10b:
        subi rmp1,-10 ; 10 zum Rest wieder addieren
        ret ; fertig
        .endif
;
; =========================================
;    H A U P T P R O G R A M M    I N I T
; =========================================
;
Main:
; Initiiere Stapel
        ldi rmp1, HIGH(RAMEND) ; Initiiere MSB Stapel
        out SPH,rmp1
        ldi rmp1, LOW(RAMEND) ; Initiiere LSB Stapel
        out SPL,rmp1
; Init PORT D
        ldi rmp1,0xFF ; alle als Ausgang
        out DDRD,rmp1
        out PORTD,rmp1
; Init Port B
        ldi rmp1,0x3F ; alle als Ausgang
        out DDRB,rmp1
        out PORTB,rmp1
; Init Port C
        out DDRC,rmp1 ; alle als Ausgang
        out PORTC,rmp1
; Init Uhrzeit, Stellen der Uhr auf 12:00:00
```

```
.if pbcd == 1
        ldi rStd,0x12 ; Packed-BCD-Uhrzeit
        ldi rMin,0x00
        ldi rSek,0x00
        .else
        ldi rStd,12 ; Binaer-Uhrzeit
        ldi rMin,0
        ldi rSek,0
        .endif
; Init Timer 1
        ldi rmp1,High(cCtc) ; CTC-A-Wert setzen
        out OCR1AH,rmp1
        ldi rmp1,Low(cCtc)
        out OCR1AL,rmp1
        ldi rmp1,0 ; Timer-Mode Normal/CTC-A
        out TCCR1A,rmp1
.if (fast == 1) | (speed == 2) ; bei Schnell und bei Stundentest
        ldi rmp1,(1<<WGM12)|(1<<CS10) ; Prescaler = 1
        .elif speed == 0 ; Normalbetrieb
        ldi rmp1,(1<<WGM12)|(1<<CS12)|(1<<CS10) ; normal, Prescaler=1024
        .elif speed == 1
        ldi rmp1,(1<<WGM12)|(1<<CS11)|(1<<CS10) ; Prescaler=64
        .endif
        out TCCR1B,rmp1 ; Timer Mode CTC-A, Prescaler
        ldi rmp1,1<<OCIE1A ; Compare-A-Interrupt
        out TIMSK,rmp1 ; in Timer-Interrupt Maske
;
; Sleep und Interrupts
        ldi rmp1,1<<SE ; Enable sleep mode idle
        out MCUCR,rmp1 ; in MCU-Kontrollport
        sei ; Enable Interrupts
;
; =============================================
;     P R O G R A M M - S C H L E I F E
; =============================================
;
Loop:
        sleep ; Schlafen legen
        nop ; Dummy fuer Aufwecken
        rjmp loop ; Zurueck nach Loop
;
; Ende Quellcode
; Copyright
.db "(C)2017 by Gerhard Schmidt  " ; menschenlesbar
.db "C(2)10 7ybG ehrra dcSmhdi t " ; wortgerecht
;
```