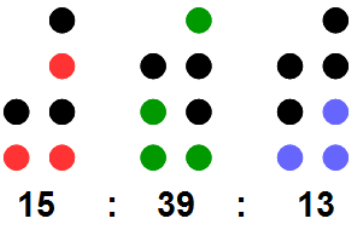


AVR-Anwendungen
RGB-Dünne-Berg-Uhr mit ATmega16
Hardware, Aufbau, Anwendung und Software für die BCD-Uhr

RGB-Dünne-Berg-Uhr mit ATmega16

... als Kontrast zu Dicke-Tal-Uhren, hier mit Vielfarben zum Auswählen.

1 Eigenschaften der RGB-Uhr mit ATmega16

Die Hardware-Eigenschaften der beschriebenen RGB-Uhr:

- BCD-Kodierung: Stunden-/Minuten-/Sekunden als binär-kodierte Einer- und Zehner-Ziffern: Eins, Zwei, Vier, Acht, Zehn, Zwanzig und Vierzig mit je einer LED
- 20 RGB-Leds zeigen die Uhrzeit in Stunden:Minuten:Sekunden als binär-codierte Ziffern in vier, per Software einstellbaren Farben an:
    1. Stunden zwischen 1 bis 5, Minuten und Sekunden zwischen 1 bis 14: blau
    2. Stunden zwischen 6 bis 11, Minuten und Sekunden zwischen 15 bis 29: grün
    3. Stunden zwischen 12 bis 17, Minuten und Sekunden zwischen 30 bis 44: rot
    4. Stunden zwischen 18 bis 23, Minuten und Sekunden zwischen 45 bis 59: violett
- Vorteile: so eine Uhr kann nicht jeder einfach ablesen daher ideal für Digitalelektronik-Freaks mit etwas Hang zur Kryptik, viel buntig daher eine Bereicherung des Nachttisches mit anderem als gewöhnlichem rot, sehr helle Anzeige daher auch draußen und bei hellem Sonnenlicht noch gut ablesbar, ersetzt bei nächtlichen romantischen Stunden das Hitzeflimmern von Verstärkern und ersetzt schummrig-gelbe Hochvolt-Glimmlampen durch mehrfarbige Niederspannungsbeleuchtung
- Mit Poti regelbare Helligkeit zur individuellen Einstellung
- Quarzgenaue Uhr, bequem und schnell einstellbar mit zwei per Software entprellten Tasten und dem Potentiometer
- Geringer Stromverbrauch: weniger als 1 kWh für ein ganzes Jahr
- Gut dokumentierter freier Assembler-Quellcode, leicht modifizierbar für andere Farben, Taktfrequenzen und anpassbar an andere Prozessortypen
- Vielfältige Diagnosemöglichkeiten im Quellcode zum Auffinden von Verdrahtungsfehlern der Hardware
- Freie Dokumentation mit Quellcode und allen Rechen- und Grafikdokumenten
- Einfach in der fertigen Schaltung umprogrammierbar mit der eingebauten ISP6-Schnittstelle, daher schnelle Änderungen der Farben möglich

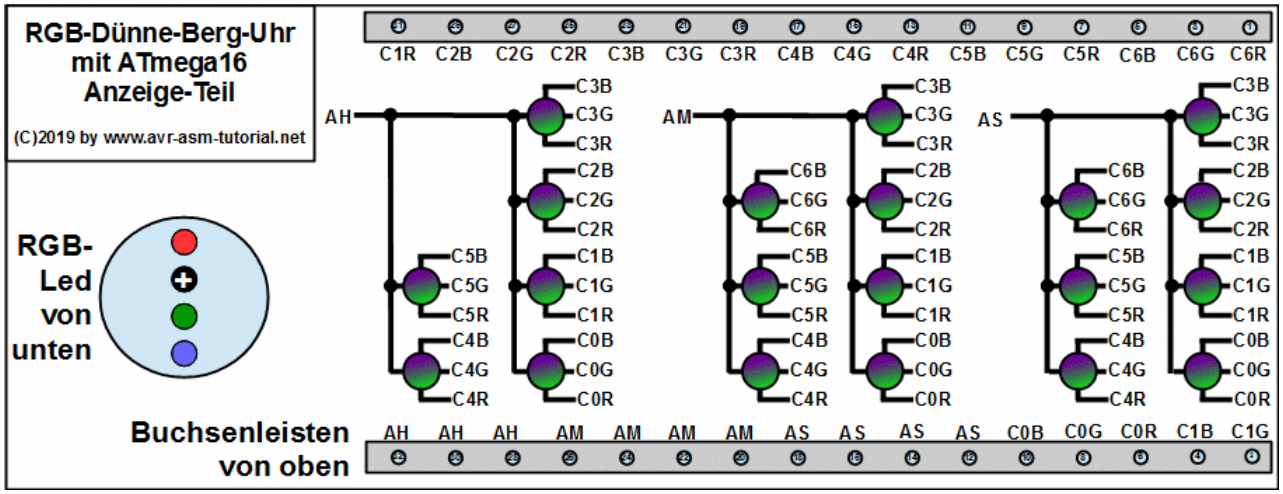
Ein kurzes Video mit der Uhr gibt es im AVI-Format hier, im MP4-Format hier.

Seitenanfang | Eigenschaften | Hardware | Aufbau | Software

2 Hardware

2.1 Anzeigen-Teil

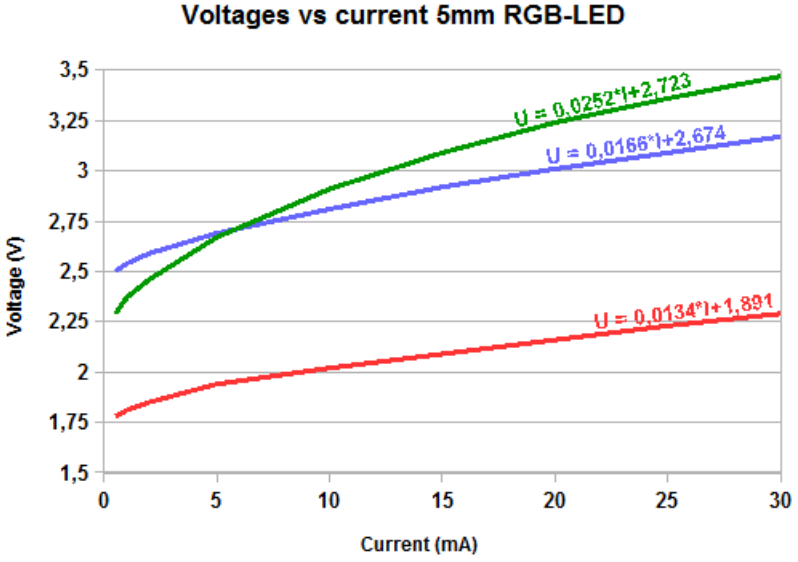
Dies ist der Anzeigenteil der Uhr. Er besteht aus 20 RGB-Leds mit gemeinsamer Anode und zwei 16-poligen Buchsenleisten, die den Anzeigenteil mit dem Controllerteil verbinden.



Die Ansteuerung der Leds erfolgt mit Multiplexen. Je ein Drittel der Zeit sind die Stunden-, Minuten- und Sekunden-Leds aktiviert. Dazu sind die Anoden zusammengeschaltet, sie werden von je einem PNP-Transistor auf hohes Spannungspotenzial gezogen. Jeweils für zwei LEDs ist eine Anodenleitung an der Buchsenleiste vorgesehen, um den Strom an jedem Anodenpin der Buchsenleiste auf maximal 6*20 = 120 mA zu begrenzen. Das ergibt insgesamt 11 Anodenleitungen (drei für die sechs Stunden-Anoden, je vier für die sieben Minuten- und Sekunden-Anoden.

Die drei Leds jeder Leuchtdiode sind in drei gemeinsamen Kathoden-Leitungen zusammengeführt. Die drei niedrigsten Leds der Einer-Anzeige sind miteinander verbunden und bilden die drei Signale C0R, C0G und C0B. Sie werden vom Controller auf niedriges Potenzial gezogen, wenn sie über den aktiven Multiplexzyklus an sein sollen. Sieben Dioden (vier mit den Einern, drei mit den Zehnern bei Minuten und Sekunden, zwei bei Stunden) mit je drei Farben bilden zusammen 21 gemeinsame Kathodenleitungen. Zusammen mit den elf Anodenleitungen sind 32 Pins an den beiden 16-poligen Buchsenleisten belegt.

Das hier sind die Spannungen der drei Leds in Durchlassrichtung bei unterschiedlichen Strömen zwischen 0,5 und 30 mA. Die Kurven verlaufen sehr unterschiedlich. Bei der roten Led ist die Spannung am niedrigsten, sie steigt mit dem Strom auch nur gering an. Die grüne Led hat bei Strömen bis 5 mA eine niedrigere Spannung als die blaue, steigt aber mit dem Strom steiler an und überholt die blaue, die weniger steil ansteigende Spannungen aufweist. Die Werte sind wegen Exemplarstreuungen unge-

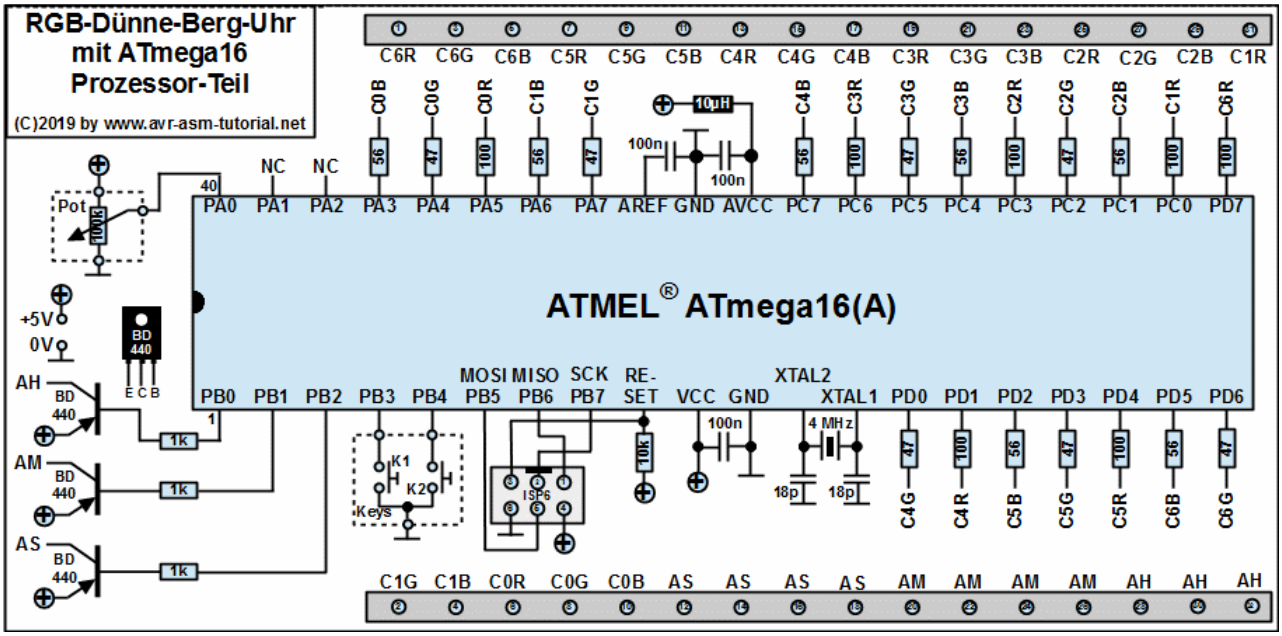


nau.

Die Näherungsgeraden wurden aus den Spannungen bei Strömen ab 15 mA errechnet.

2.2 Prozessorteil

Dies ist der Prozessorteil mit dem ATmega16. Er steuert mit dem I/O-Port A fünf Farbkathoden an, mit I/O-Port C weitere acht und mit I/O-Port D die letzten acht. Die Stromsteuerung der Led-Kathoden erfolgt durch Widerstände, die sich aus den unterschiedlichen Durchlassspannungen der drei Farben ergeben (siehe unten). Die Kathoden-Portpins sind aktiv niedrig.



Die Ströme durch alle Leds bei einer Multiplex-Periode summieren sich im GND-Pin des Prozessors. Um den Strom aus dem GND-Pin des ATmega16 auf die laut Datenblatt maximal zulässigen 200 mA (PDIP-Packung) zu begrenzen, wurde der Strom durch die einzelnen Leds auf 20 mA festgelegt. Bei der Darstellung der Zahl 57 sind jeweils fünf Leds eingeschaltet. Würden zur Farbkombination zwei Farben eingeschaltet, ergeben sich 10 Led-Ströme. Aus dem Maximalstrom durch den GND-Pin ergibt sich die Auslegung auf 20 mA pro Farbe. Würden hingegen alle drei Farben der Leds eingeschaltet (weiß), ergäben sich 15 zu treibende Led-Farben und der Strom müsste auf 200 mA / 15 = 13,3 mA pro Farbe einer Led begrenzt werden. Das geht auch noch, ohne dass die Leds beim Dreier-Multiplexen zu schwach werden. Die Led-Treiber-Widerstände wären dann entsprechend zu vergrößern.

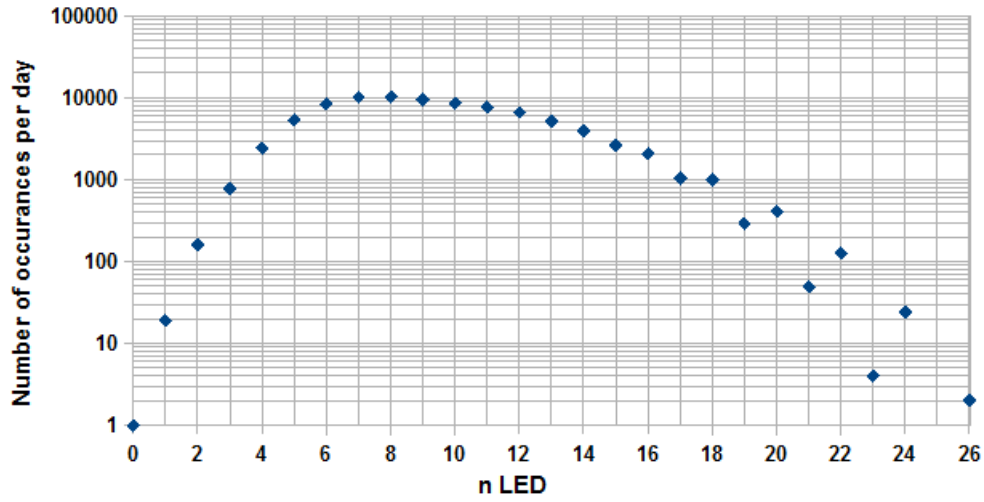
Die drei Anoden werden von den drei I/O-Portpins PB0 (Stunden), PB1 (Minuten) und PB2 (Sekunden) mittels den PNP-Transistoren vom Typ BD440 angesteuert, deren Basisstrom mit Widerständen von 1K begrenzt wird. Die Anoden-Portpins sind ebenfalls aktiv low (Null = An).

Die beiden Tasten zur Zeiteinstellung sind an die Portpins PB3 und PB4 angeschlossen. Beide sind durch interne Zuschaltung der Pull-Up-Widerstände ebenfalls aktiv low.

Das Potentiometer von 100k teilt die Betriebsspannung und ist mit seinem Schleifer an den AD-Wandler-Eingangspin ADC0 angeschlossen.

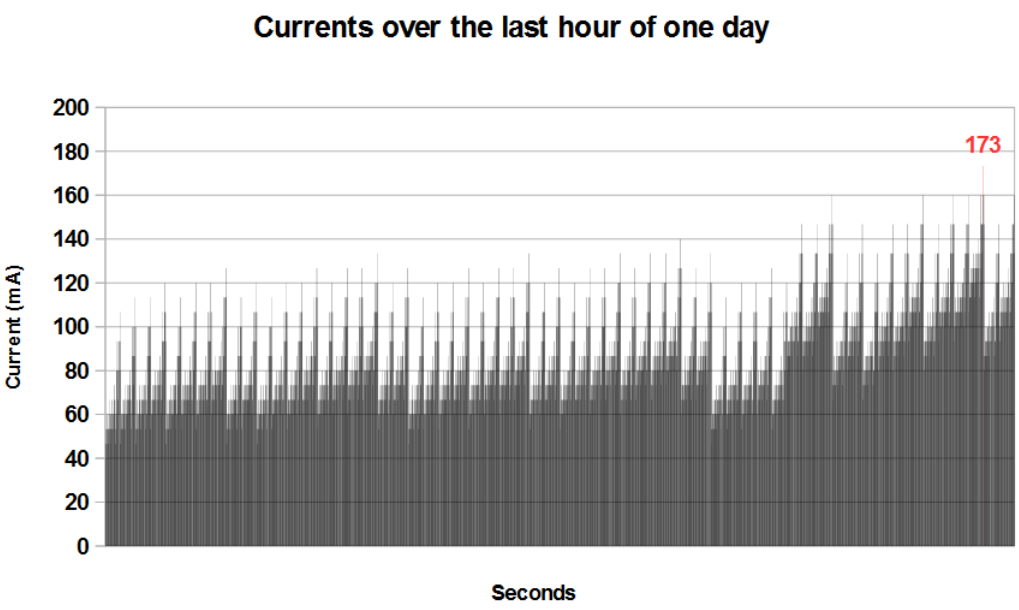
An den XTAL-Pins ist ein 4 MHz-Quarz mit zwei 18 pF-Anschwingkondensatoren angeschlossen. Dieser ist per Fuse-Umstellung als Taktquelle einzuschalten (siehe unten). Es können auch andere Quarze verwendet werden, dann muss die Taktung im Quellcode in der Konstante **clock** entsprechend angepasst werden.

Die Portpins PB5, PB6 und PB7 bedienen die ISP6-Schnittstelle, über die der Prozessor in der Schaltung programmiert werden kann.



Dies ist die Anzahl an Leds, die während des Dauers einer Sekunde über einen ganzen Tag lang angeschaltet sind. Die teilweise Einschaltung durch Multiplexen reduziert diese Anzahl, so dass die maximal gleichzeitig eingeschalteten Leds bei allen Kombinationen auf 10 beschränkt bleibt.

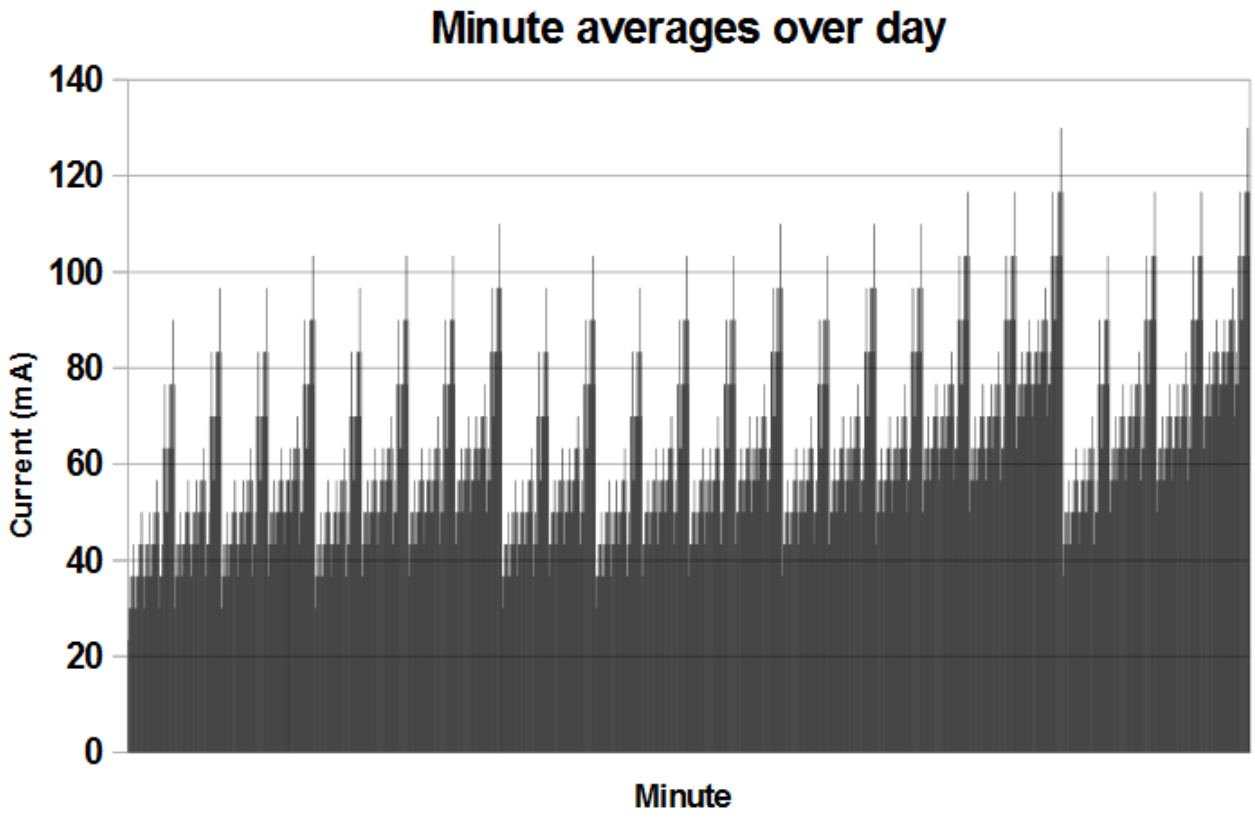
Die Widerstände, wie sie im Schaltbild eingezeichnet sind, legen den Strom auf 20 mA pro Led-Farbe fest. Die Maximalströme durch die Leds treten am Ende des Tages, zwischen 23 und 24 Uhr auf. Das Bild zeigt die sehr unterschiedlichen Strombelastungen in diesem Zeitraum.



Maximal tritt einmalig eine Stromspitze von 173 mA über eine Sekunde lang auf (10 Leds sind gleichzeitig an). Das Datenblatt legt einen Maximalstrom durch den GND-Pin von 200 mA für die PDIP-Packung fest, die 10*20 mA bzw. die durchschnittlichen 173 mA halten diese Grenze ein.

Für die Netzteillast sind neben dem Maximalstrom auch noch die über längere Zeiträume gemittelte Belastung wichtig, da sie die Wärmelast von Trafo und Spannungsregler bestimmen. Das Diagramm zeigt die Minutendurchschnitte über den gesamten Tag.

Im Schnitt bleiben die Minutenlasten bei unter 100 mA, nur am Stundenende treten kurzzeitig Ströme über 100 mA auf. Diese kurzzeitigen Spitzen sind für die thermische Leistung irrelevant.

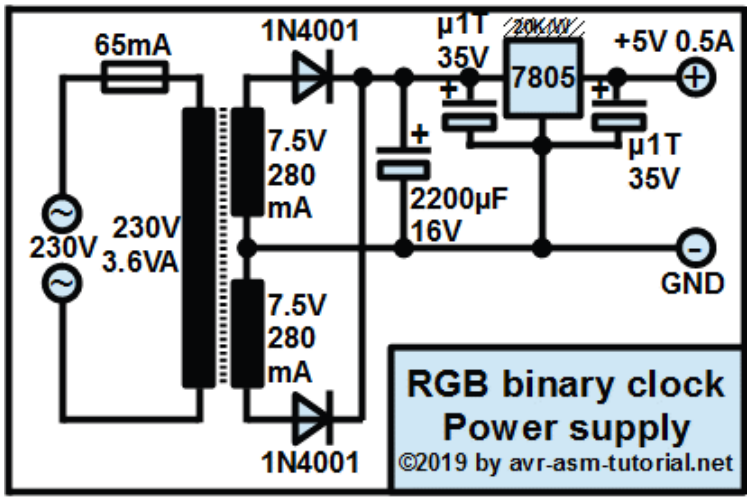


2.3 Netzteil

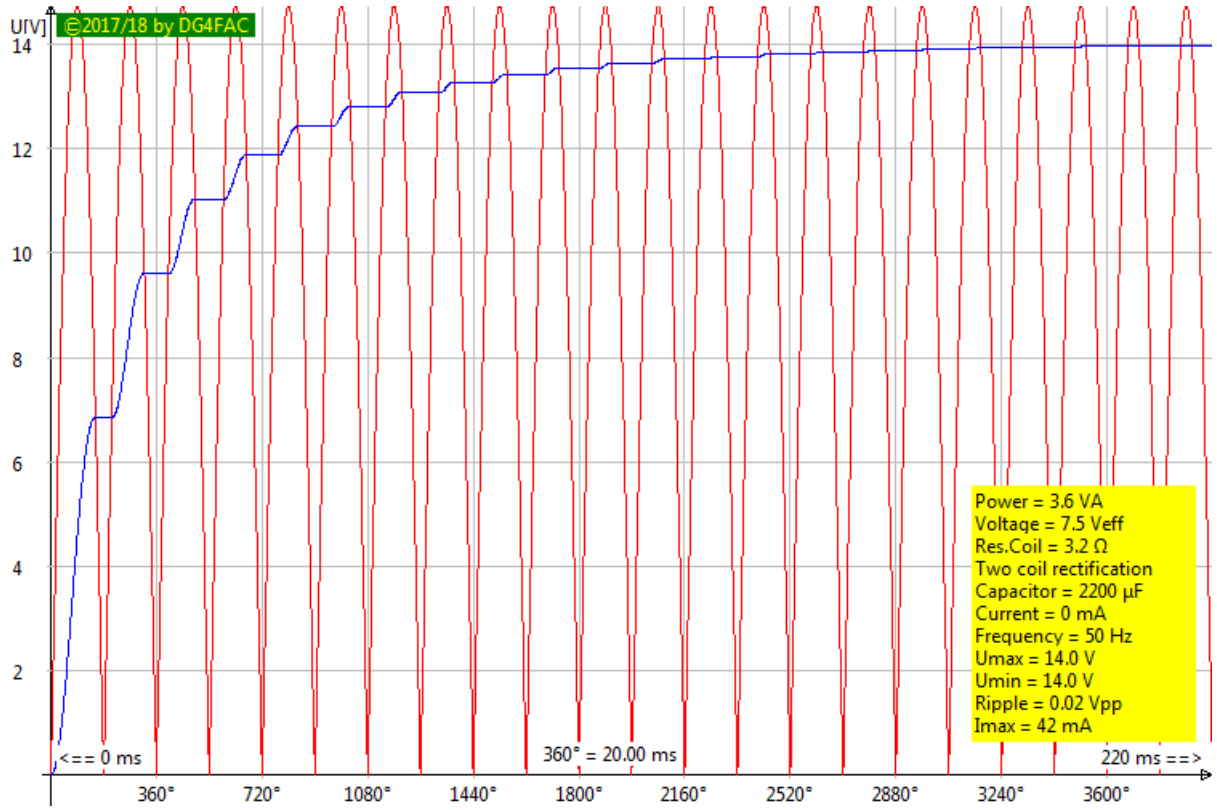
Das Netzteil muss folgende Eigenschaften besitzen:

1. Es muss stabile 5 V liefern, da der Strom durch die LEDs über strombegrenzende Widerstände von dieser Spannung abhängt.
2. Da jeweils maximal 5 Leds mit maximal zwei Farben (10 Einzel-Leds) und maximal 20 mA Strom pro Led an sein können, muss es kurzzeitig einen Strom von maximal 200 mA verkraften. Bei 7,5 V Trafospannung sind das 1,5 VA, daher kommt der nächsthöhere Verfügbare, ein 3,6 VA zum Einsatz.
3. Der Spannungsregler muss ein 7805 sein, mit maximal ca. 1 W ist ein kleiner Kühlkörper von 20 K/W ausreichend.

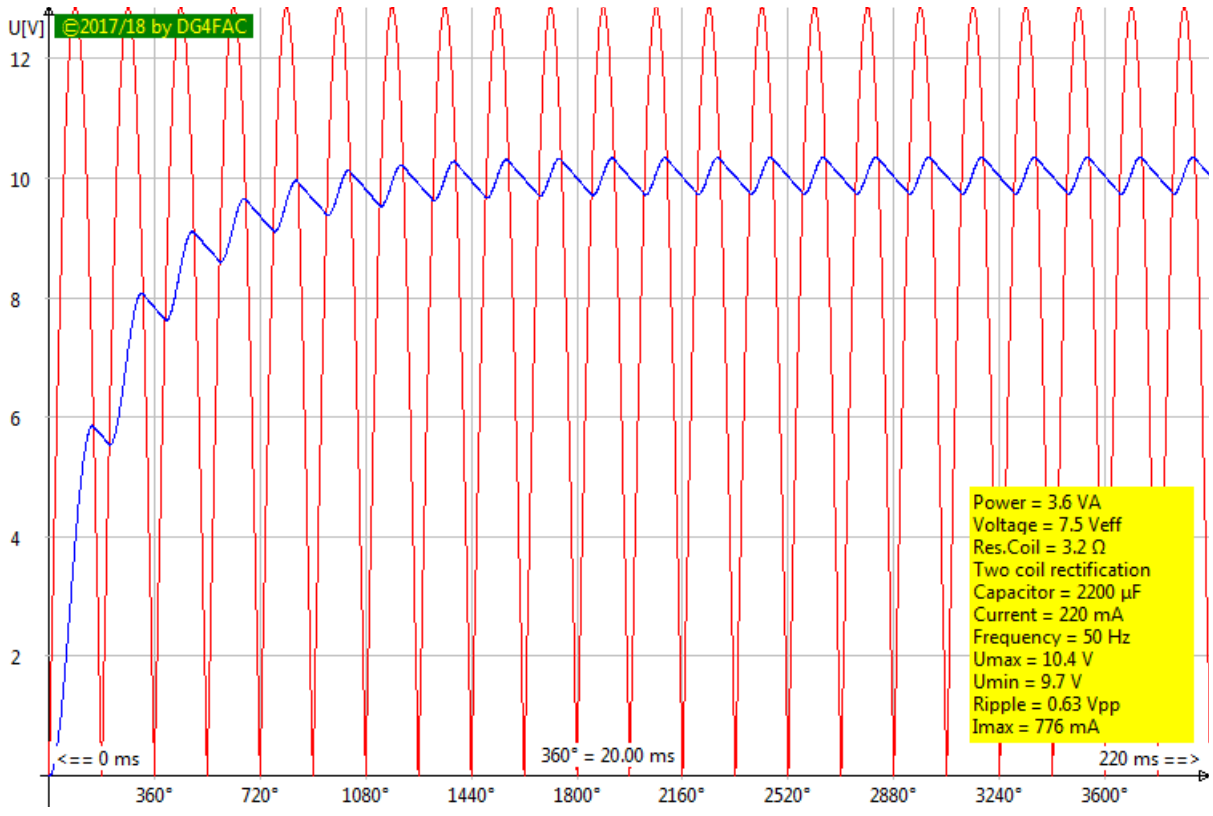
Im Schaltbild ist so ein Standard-Netzteil mit einem 2*7,5 V-Trafo mit seiner Dimensionierung zu sehen. Die beiden Dioden sind 1N4001.



Das nachfolgende Diagramm simuliert das dargestellte Netzteil ohne angeschlossene Last (siehe die Power-Supply-Software hier). Die Spannung am Elko bleibt ohne Last bei deutlich unter 16 V, deshalb kommen die Elkos mit einer Spannungsbelastbarkeit von 16 V aus.



Das hier ist die Netzteilspannung bei 220 mA Last, der Maximallast währenddessen der höchsten Multiplexphase. Es besteht ausreichend Lastreserve und der Elko ist großzügig dimensioniert.



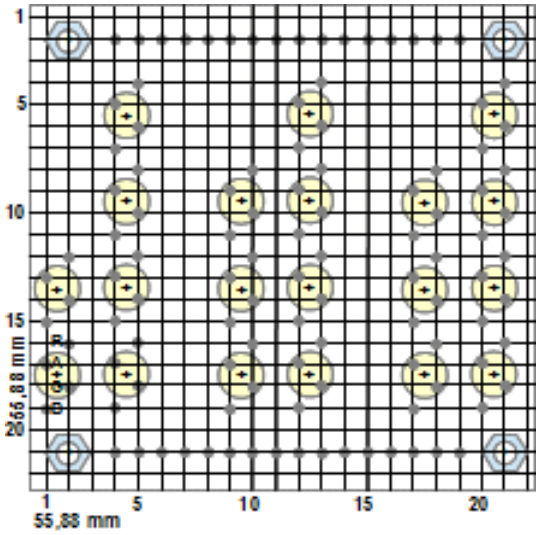
Das Netzteil wurde mit einer Dauerlast von 280 mA getestet. Die 5V-Spannung ist absolut

stabil, die Temperaturen von Trafo und 7805 bleiben im Rahmen. Erst bei größeren Strömen von über 500 mA bricht die geregelte Spannung ein.

3 Aufbau

Sowohl die Prozessor- als auch die Anzeigeplatine sind auf beidseitig kaschiertem Lochraster aufgebaut, damit die beiden mit den zwei Stecker-/Buchsen-Leisten umgekehrt aufeinander gesteckt werden können und die Verdrahtung der Stecker- bzw. Buchsenleiste von der Bestückungsseite her erfolgen kann.

3.1 LED-Platine



Die 20 RGB-Leds sind auf einer 66*66 mm-Platine angebracht, die mit vier Schrauben M2,5 von der Innenseite des Gehäuses her die Leds in den 5 mm-Bohrungen halten.

Die vier Anschlussdrähte der Leds sind abgebogen und auf seitenversetzten Bahnen des Lochrasters angeordnet, damit die Leds seitliche Stabilität erhalten und nicht abknicken können. Die Plus-Zeichen zeigen die Mittelpunkte der 5 mm-Bohrungen auf der Frontplatte an.

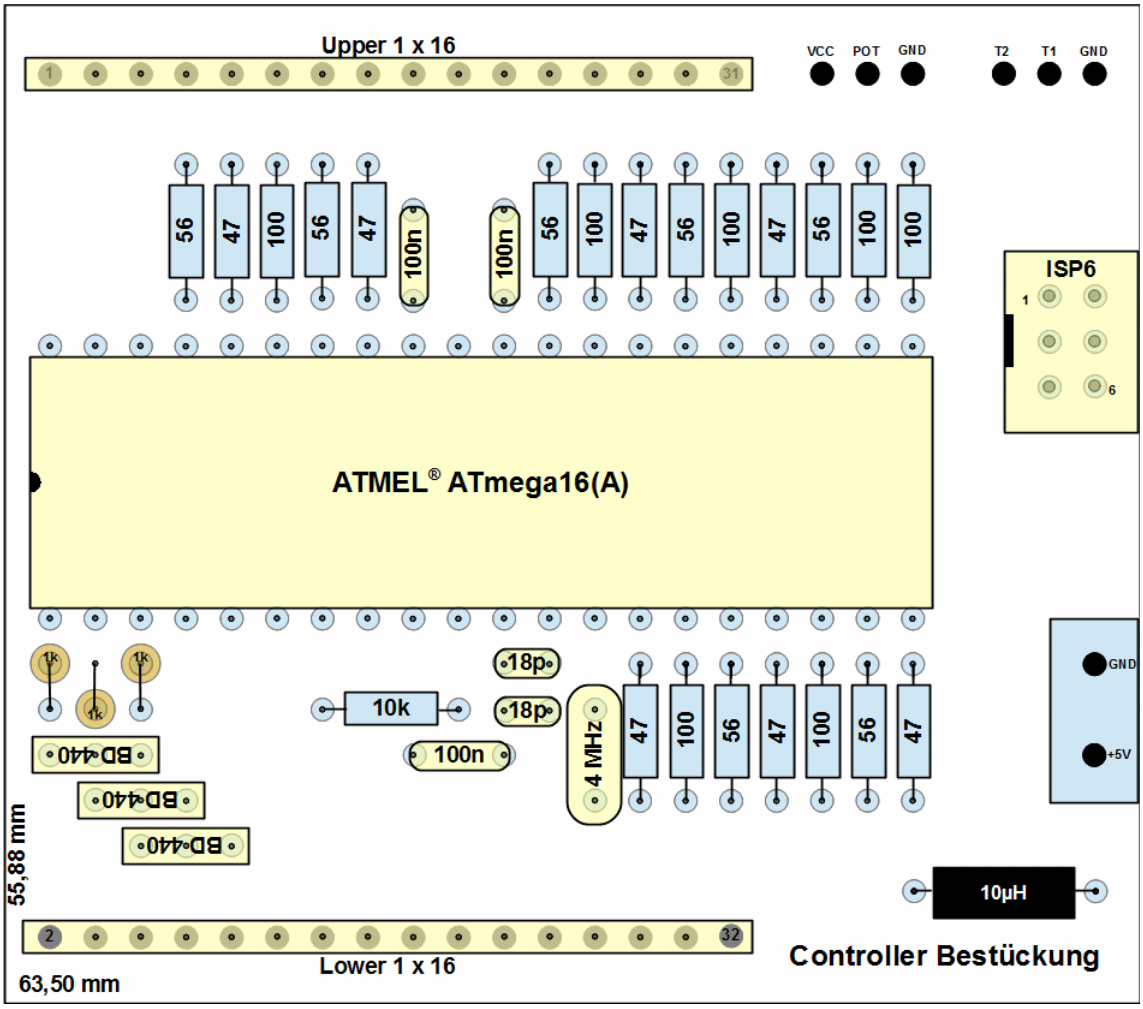
Am besten baut man die Led-Platine so auf. Zunächst bohrt man die 5 mm-Bohrungen, weitet sie um etwa einen halben mm auf, damit die Leds leicht eingeführt werden können und etwas Spiel haben. Dann bohrt man auch die vier M2,5-Befestigungslöcher in die Frontplatte und befestigt darin vier M2,5x20-Schrauben. Dann steckt man eine Reihe mit 6 oder 7 Leds lose auf die Platine und führt sie mit den vier Verschraubungen in die Löcher der Frontplatte ein. Dann entfernt man überstehenden Led-Anschlussdraht und lötet die Lötstellen der sechs bzw. sieben Leds der Anzeigestelle.

Nacheinander bestückt man so Stunden-, Minuten- und Sekunden-Leds und stellt dabei sicher, dass sie in die gebohrten Löcher der Frontplatte passen. Abschließend lötet man die beiden 16-poligen Buchsenleisten von der Bestückungsseite her ein und verdrahtet die Leds gemäß Schaltplan.

Die Verdrahtung der Buchsenleiste erfolgt dabei folgendermaßen. Von der Lötseite her schiebt man einen verzinnten Kupferlackdraht durch das dem Pin benachbarte Rasterloch und knickt diesen mit der Pinzette etwa vier mm lang um 90° ab. Nach dem Einlöten des Pins lötet man den abgeknickten Draht an die Lötstelle.

3.2 Prozessor-Platine

Die Belegung der 65*55 mm großen Prozessorplatine mit den Komponenten zeigt das Bild.



Da die Prozessorplatine wird mit ihren 32 Steckpins auf die Led-Platine von hinten aufgesteckt wird, mit der Komponentenseite nach oben und in umgekehrter Richtung, ändert sich die Belegung und Nummerierung der Steckpins entsprechend.

Die beiden 16-poligen Steckerleisten werden nach unten hin bestückt. Dazu wird, wie bei der Anzeigenplatine auch, jeweils ein Pin festgelötet und ein am Ende verzinnter Kupferlackdraht von unten her eingeführt, die oberen vier mm des Drahts um 90° abgewinkelt und mit der Lötstelle des Pins verlötet.

Sind die Stromversorgungsleitungen, die 40-polige Fassung des ATmega16, die Led-Widerstände, die Anodentreibertransistoren und die beiden Pfostensteckerleisten bestückt und verdrahtet, kann man die Led-Platine anstecken und die korrekte Led-Verdrahtung testen, indem man auf der ATmega16-Fassung nacheinander die Anodentreiberpins 1, 2 und 3 auf GND legt und die entsprechenden Kathodenanschlusspins PA3 bis PA7, PC0 bis PC7 sowie PD0 bis PD7 ebenfalls mit GND verbindet. Die Leds 0 bis 5 (Stunden) bzw. 0 bis 6 (Minuten, Sekunden) sollten dann in der Reihenfolge blau - grün - rot leuchten.

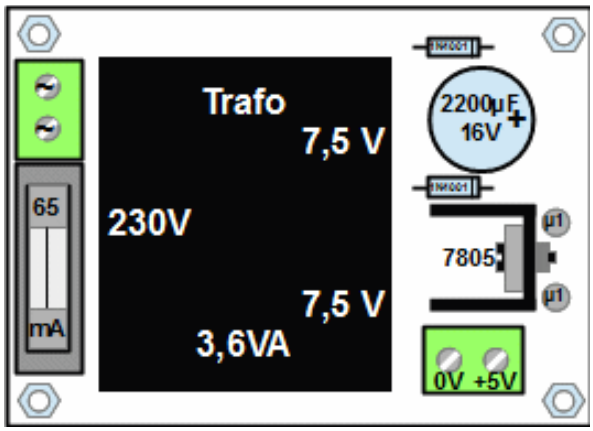
Sind die Stromversorgung, der Quarz mit den beiden Kondensatoren, der Resetwiderstand, die ATmega16-Fassung und der ISP6-Stecker bestückt und verdrahtet, kann mit dem Programmiergerät versucht werden, die Erkennungsbits aus dem ATmega16 auszulesen und die Quarzfuse im Prozessor zu aktivieren (siehe unten). Gelingt beides, kann man den Quellcode mit eingeschaltetem High-Speed-Modus (die entsprechende Quellcode-Zeile

mit **cAccel** auf einen höheren Wert als Eins stellen und neu assemblieren/brennen). Der geht dann mit Vielfach-Speed einen Tag durch, z. B. mit 8 in drei Stunden oder mit 64-fach-Speed in 22,5 Minuten.

Weitere in der Software vorgesehene Fehlerdiagnosen für die Hardware sind im Kapitel Software näher beschrieben.

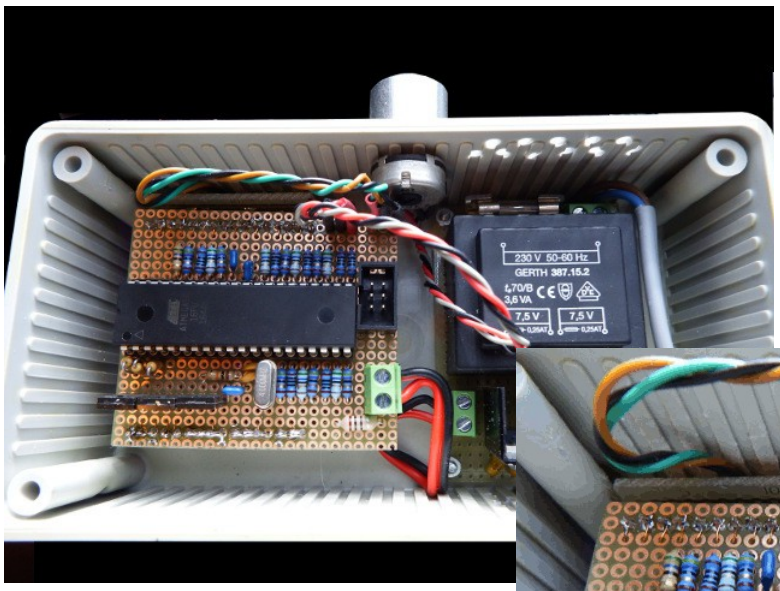
3.3 Netzteil-Platine

Das Netzteil wird auf einer 70x50 mm großen einseitigen Lochrasterplatine aufgebaut.



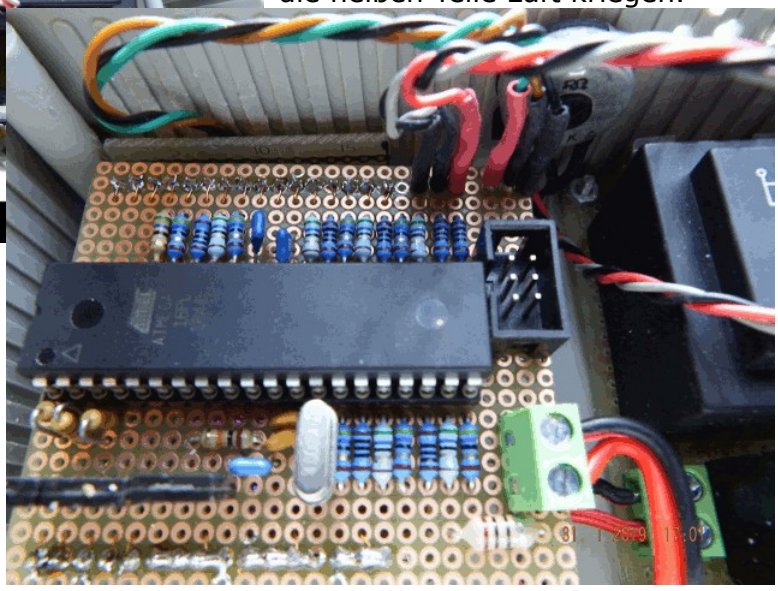
3.4 Gehäuseeinbau

Dem Einbau der Einzelkomponenten in ein 150*75*45 mm großes Standard-Plastikgehäuse steht nun nichts mehr entgegen.



Dank der 32 Pins der Stecker-/Buchsen sitzt die Prozessorplatine huckepack auf der Anzeigenplatine und braucht keine eigene Befestigung.

Nicht vergessen: Unter dem 7805-Kühlkörper und über dem Trafo müssen noch Lüftungslöcher in das Gehäuse gebohrt werden, damit die heißen Teile Luft kriegen.



3.5 Gehäusebeschriftung

Die Vorderseite des Gehäuses kriegt noch eine passende Beschriftung, die Rückseite ein Typenschild. Entsprechende Vorschläge finden sich in der Open-Office-Grafikdatei, die auch alle Schaltbilder und Layouts enthält.

Soll das ein Geschenk werden, muss natürlich noch eine passende Beschreibung dazu.

Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

4 Software

4.1 Downloads

Die Software ist hier im asm-Format verfügbar und im Anhang aufgelistet.

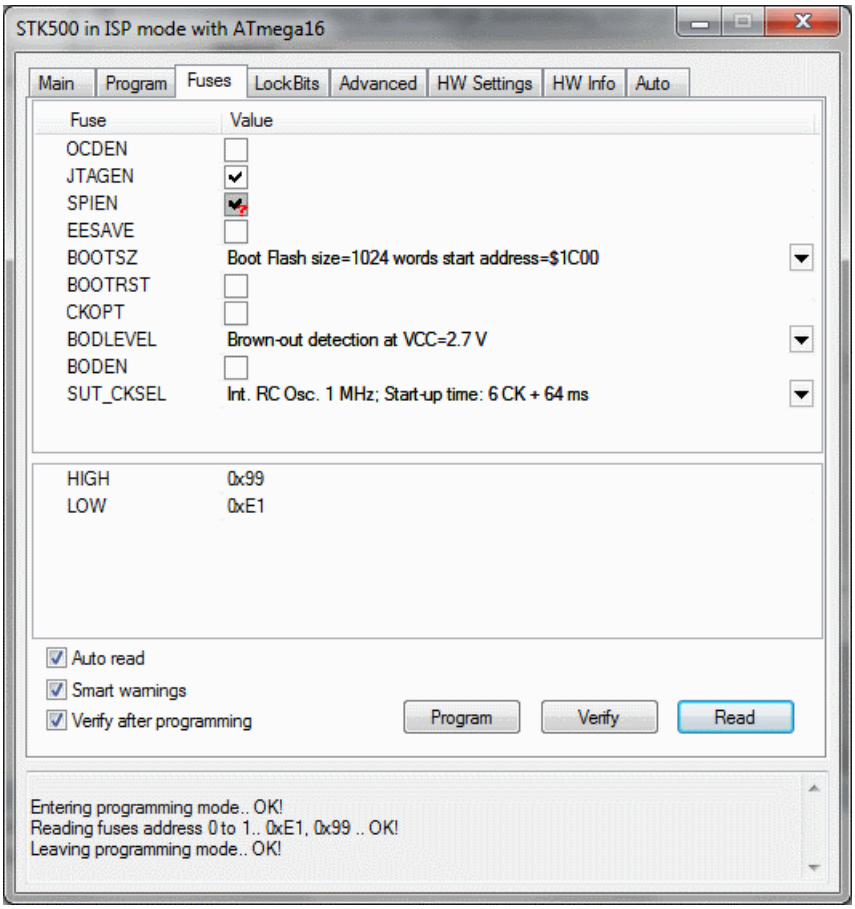
Zu der Uhr gibt es noch folgende zusätzliche Dokumente zum Download:

- Alle Schaltbilder, Zeichnungen und Grafiken in mehreren Zeichenblättern in einem Open-Office-Draw-Dokument hier,
- alle Berechnungen zu der Uhr (Taktraten, RGB-Led-Spannungsanalyse, Messung und Berechnung der U_{CE}-Sättigungsspannung beim BD440, gemessene RGB-Led-Ströme beim gebauten Prototyp) gibt es in vier Tabellenblättern des Open-Office-Dokuments hier,
- die Berechnungen zur Strombelastung jeder Sekunde eines Tages und die Mittelung über Minuten befinden sich in zwei Rechenblättern des Open-Office-Dokuments hier (Achtung! Das Dokument ist eine Riesentabelle und braucht lange Zeit zum Laden und zum Rechnen!),
- die Flussdiagramme der Software sind im Open-Office-Draw-Dokument hier enthalten
- das Handbuch zum Verschenken gibt es hier.

4.2 Assemblieren des Quellcodes

Vor dem Assemblieren sollte noch einmal kontrolliert werden, dass im Quellcode keine Debugging-Schalter gesetzt sind (siehe unten).

Zum Assemblieren wird ein Assembler benötigt, der .IF-Direktiven beherrscht. Der ATMEL-Assembler 2 beherrscht das. Wer sich scheut, 900 MB ATMEL-Studio-Software herunterzuladen und zu installieren oder wer kein Windows-Betriebssystem hat, ist mit meinem Assembler gavrasm besser und einfacher bedient. Wie damit mit einem 64-Bit-Betriebssystem assembliert wird, findet sich für Windows hier und für Linux hier. Wer ein ganz anderes Betriebssystem (32-Bit, Mac-OS, etc.) hat, kann sich den Quellcode von gavrasm herunterladen und mit Free Pascal was Eigenes selbst kompilieren.



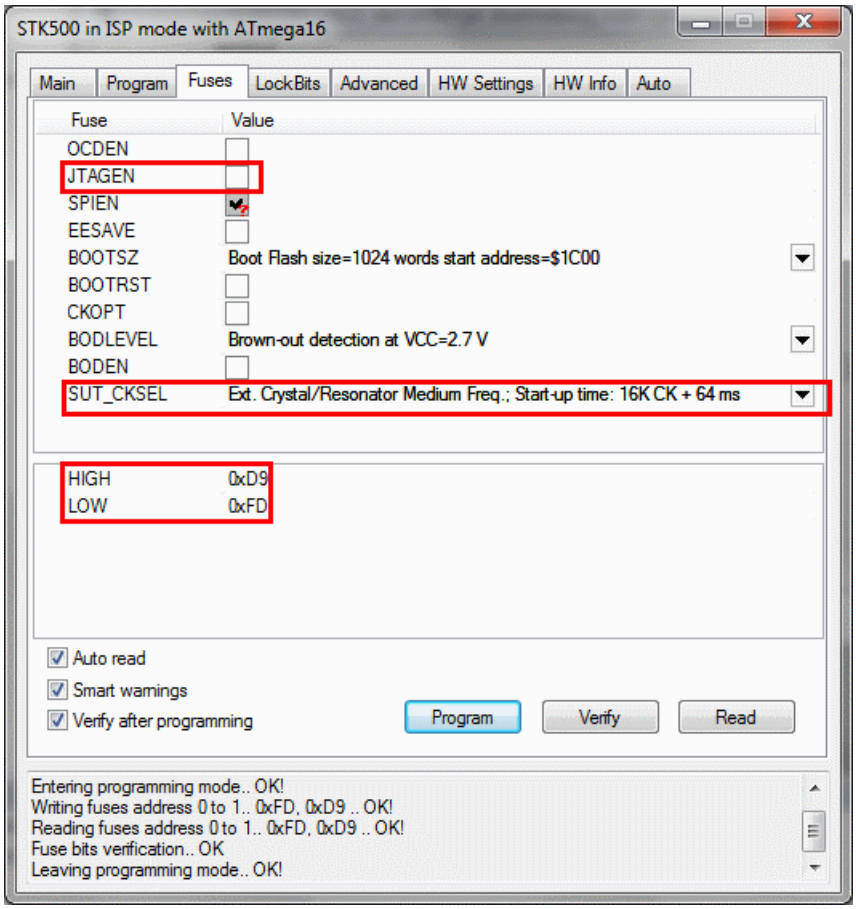
Der assemblierte Maschinencode sollte nach dem Assemblieren als .hex-Datei im gleichen Verzeichnis herumlie-

gen.

4.3 Flashen, Fuses

Der Hexcode muss dann in den Flashspeicher des ATmega16 geschrieben werden. Das geht mit einem AVR-Brenner und der entsprechenden Software.

Vor oder nach dem Flashen müssen noch die Fuses des ATmega16 geändert werden, sonst läuft die Uhr im 96-Stunden-Modus und ein Led-Bit geht nicht. Die entsprechenden Fuses sind im rechten Bild markiert.



4.4 Funktionsweise der Software

Die folgenden Einzelkapitel stellen die grundlegenden Funktionen der Software dar.

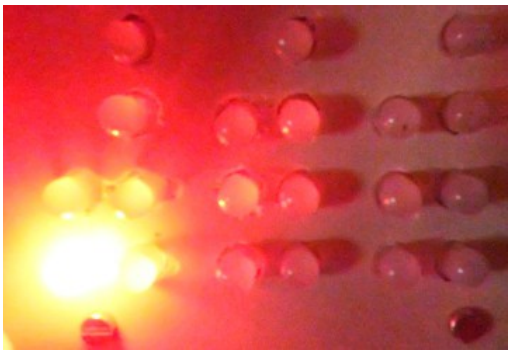
4.4.1 Hardware-Diagnosen

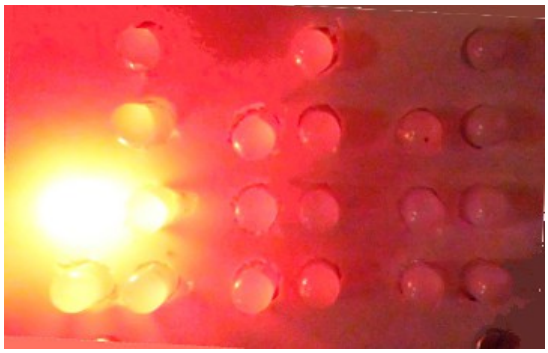
Als Hilfe bei der Inbetriebnahme der Uhr können die folgenden Diagnosehilfen benutzt werden. Sie werden eingestellt, indem im Kopf der Quelldatei die entsprechenden Konstanten mit **Yes** versehen werden. Dabei macht es wenig Sinn, gleichzeitig mehrere Optionen anzuschalten, da sich diese teilweise gegenseitig ausschließen. Nach der Änderung jeweils neu assemblieren und die Flashdatei brennen.

1. "cAccel": Mit diesem Parameter kann die Uhr um den Faktor N beschleunigt werden. Mit N = 8 achtfach, mit N = 100 hundertfach, mit 144 der Tag im Schnelldurchlauf in 10 Minuten.
2. "dbg_leds": Nacheinander werden alle Leds in der Reihenfolge
   - Sekunden - Minuten - Stunden,
   - Einer - Zweier - Vierer - Achter - Zehner - Zwanziger und, bei Sekunden/Minuten, die Vierziger,
   - blau - grün - rot

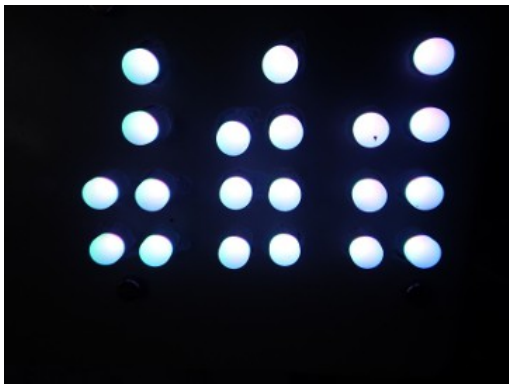
   jeweils für eine Sekunde lang eingeschaltet. Damit können falsch verdrahtete Leds festgestellt werden.
3. "dbg_adc": Der gemessene ADC-Wert wird als Stunde zwischen 0 und 23 dargestellt. Das zeigt an, ob das Poti korrekt funktioniert und richtig herum angeschlossen ist.
4. "dbg_key": Wenn Tasten gedrückt sind, gehen die roten Leds Stunden-Zehner (Taste 1) bzw. Stunden-Zwanziger (Taste 2) an. Ist keine der Tasten gedrückt, bleiben

alle Leds dunkel.





5. „dbg_moni_xxx“: Diese Debug-Schalter stellen die angegebenen Register xxx auf der Sekundenposition dar.
6. "dbg_colbal": Schnell nacheinander werden alle Leds auf weiß eingestellt, jeweils nur eine der 20 Lampen ist eingeschaltet. Die Lampen sollten alle etwa gleich weiß leuchten.



Für die endgültige Version müssen alle Schalter wieder auf **No** und **cAccel=1** umgestellt und neu assembliert werden.

4.4.2 Zeitsteuerung

Die Zeitsteuerung funktioniert folgendermaßen. Der quarzgesteuerte Takt von 4 MHz wird im Timer/Counter 1 durch 16.000 geteilt (Prescaler = 1, 15.999 im Compare-Port A). Daraus ergibt sich eine Frequenz von 250 Hz, mit der der Compare-Match-A-Interrupt ausgelöst wird. Das Register **rDivSec** zählt von 250 an abwärts. Erreicht es Null, wird die Flagge **bSec** gesetzt, ihre Behandlung erfolgt außerhalb des Interrupts.

Ferner werden mit jedem Compare-Match-A-Interrupt die Stunden-, Minuten- und Sekunden-Bits der Led-Kathoden an den Multiplex-Ports ausgegeben und der entsprechende Anodentreiber eingeschaltet. Ist die Uhr im Zeit-Einstellmodus, wird statt der Uhrzeit die eingestellte Zeit ausgegeben und die aktive Stelle (Stunde, Minute oder Sekunde) blinkt im Sekundenrhythmus mit voreingestellter Impulsdauer (Konstante **BlinkPeriod**).

Außerhalb des Interrupts bewirkt die gesetzte Sekundenflagge **bSec**, dass die Zeit um eine Sekunde erhöht wird. Die Zeit wird in binärer Form in den Registern **rHour**, **rMinute** und **rSecond** gehandhabt. Mehr zum Handling von Datum und Uhrzeit mit AVRs findet sich übrigens hier.

Die Werte für die drei Kathodenports und dem Anodenport liegen in einem Puffer ab **sRgb** im SRAM. Die Ausgabe erfolgt reihum mit dem Zeigerregister Y (YH:YL), ist das Ende des Puffers erreicht beginnt Y wieder am Anfang. Die Umwandlung der binären Zeit in Led-Kathoden-Bits ist hier beschrieben.

4.4.3 AD-Wandler-Steuerung als weitere Taktquelle

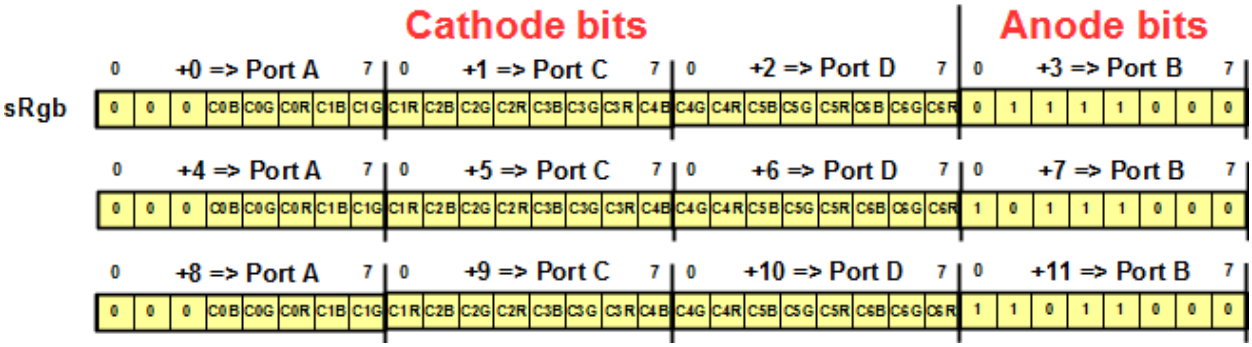
Unabhängig von diesem Timer erfolgt die Ermittlung des Potentiometerwertes durch den AD-Wandler. Dieser wird mit 4 MHz / 32 = 125 kHz getaktet. Er benötigt 13 Takte pro Wandlung. Der gemessene Wert wird im Registerpaar rAdcH:rAdcL aufsummiert, bis 64 Messungen erfolgt sind (Zähler in rAdcCnt). Dann wird das MSB der Messwertsumme in das Register **rAdcRes** kopiert und die Flagge **bAdc** gesetzt, deren weitere Behandlung außerhalb des ADC-Complete-Interrupts erfolgt. Dies ist mit einer Frequenz von 4.000.000 / 32 / 13 / 64 = 150,24 Hz der Fall, so dass alle 6,656 ms die Flagge erscheint.

Die Taktung ist allerdings etwas ungenauer als beim TC1-CTC, da zwischen dem AD-Complete-Interrupt und dem Neustart des AD-Wandlers durch die zwischenzeitlich ausgeführten Instruktionen der Interrupt-Service-Routine etwas Verzögerung eintritt.

Die Flagge **bAdc** bewirkt die Helligkeitsregelung und die Tastensteuerung zur Einstellung der Zeit und ist im Detail hier näher beschrieben.

4.4.4 Multiplexen

Damit der Multiplextakt immer die drei darzustellenden RGB-Kombinationen der Multiplex-Bits darstellen kann, sind diese im SRAM in einem 12 Byte umfassenden Puffer abgelegt.

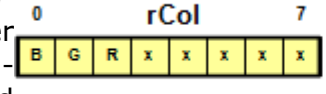


In der Reihenfolge Stunden - Minuten - Sekunden sind dazu die drei Bytes der Kathodenbits (Null = Kathode angeschaltet, Eins = Kathode ausgeschaltet) für die Ports A, C und D abgelegt. Die Leds 0 bis 3 stellen die Einer, 4 bis 6 (Stunden: 5) die Zehner-BCD-Bits dar. Die Farbzuordnung der einzelnen Bits zu den Kathoden ist jeweils in der Reihenfolge Blau - Grün - Rot angeordnet, so wie die Portbits jeweils an die RGB-Leds angeschlossen sind.

Den drei Kathodenbytes folgt jeweils das Anodenbyte für Port B. Die drei Anoden für Stunden, Minuten und Sekunden sind ebenfalls invertiert (Anode an = Null, Anode aus = Eins). Da sich diese drei Bytes nicht ändern werden sie beim Initiieren einmalig an die drei Positionen im SRAM abgelegt.

4.4.5 Umwandlung der binären Zeit in Multiplex-Bits

Dies ist die Farbkodierung im Register rCol. Die drei untersten Bits sind alle Eins, wenn die Farbansteuerung abgeschaltet ist (Stunde/Minute/Sekunde = 0) oder wenn ein darzustellendes BCD-Bit Null ist. In den Stufenfolgen 6/12/18 (bei Stunden) bzw. 15/30/45 (bei Minuten und Sekunden) wechseln jeweils die Farben der Leds. Die Farbzuordnung gilt jeweils für alle sechs (Stunden) bzw. sieben (Minuten, Sekunden) Leds in gleicher Weise.
Die Farbe wird festgestellt, indem das eingelesene Byte jeweils mit dem Vergleichswert im Register rCmp verglichen wird. Dies wird zu Beginn der Umwandlung auf sechs eingestellt. Als Grundfarbe wird in rCol blau eingestellt (0b0000.0110 = 6). Ist das Zeitbyte kleiner als dieser Wert, wird diese Farbe verwendet. Wenn nicht, wird die nächste Farbe eingestellt

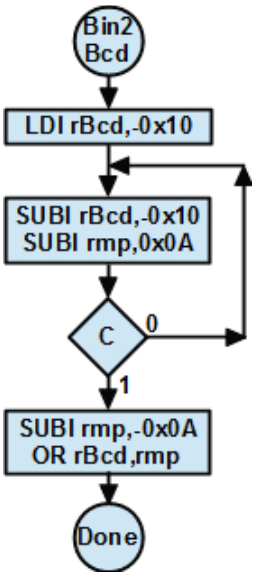


m/s	0	1 .. 14	15 .. 29	30 .. 44	45 .. 59
h	0	1 .. 5	6 .. 11	12 .. 17	18 .. 23
rCol	1 1 1	0 1 1	1 0 1	1 1 0	0 0 0
Farbe	aus	blau	grün	rot	weiss

(grün). Der Vergleichswert wird verdoppelt. Ist der Zeitwert auch größer oder gleich zu diesem Vergleichswert, wird die Farbe auf rot umgestellt. Dann wird durch Addieren der Hälfte des verdoppelten Vergleichswertes mit drei Viertel verglichen. Erreicht oder Überschreitet der Zeitwert auch diese Vergleichsschwelle, wird violett eingestellt (rot plus blau).

Die Vergleichsschwelle in rCmp wird danach für Minuten- und Sekunden-Vergleiche auf 15 eingestellt.

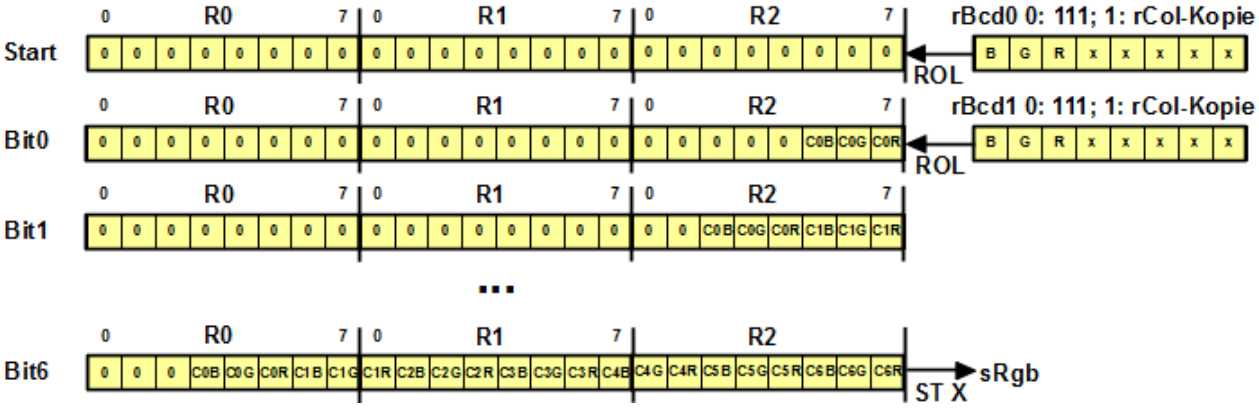
Um die Zeit jeweils in BCD dazustellen, müssen binäre Stunden, Minuten und Sekunden im Vielzweckregister rmp in BCD im Register rBcd umgewandelt werden. Dazu kommt der nebenstehend abgebildete Algorithmus zur Anwendung.



1. Zunächst wird rBcd auf Minus 0x10 eingestellt, so dass sich beim ersten Addieren von -0x10 Null ergibt.
2. Im nächsten Schritt wird -0x10 zu rBcd addiert und die höhere BCD-Ziffer um Eins erhöht (beim ersten Mal zu Null).
3. Dann werden von der Eingangszahl in rmp 10 abgezogen. Führt dies nicht zu einem gesetzten Übertragsbit, wird der Schritt so lange wiederholt, bis es gesetzt ist.
4. Durch Addieren von 10 zu rmp (Subtrahieren von -10) wird die letzte Subtraktion wieder rückgängig gemacht.
5. Die BCD-Einer in rmp werden mit den BCD-Zehnern in rBcd verodert und die Umwandlung ist beendet.

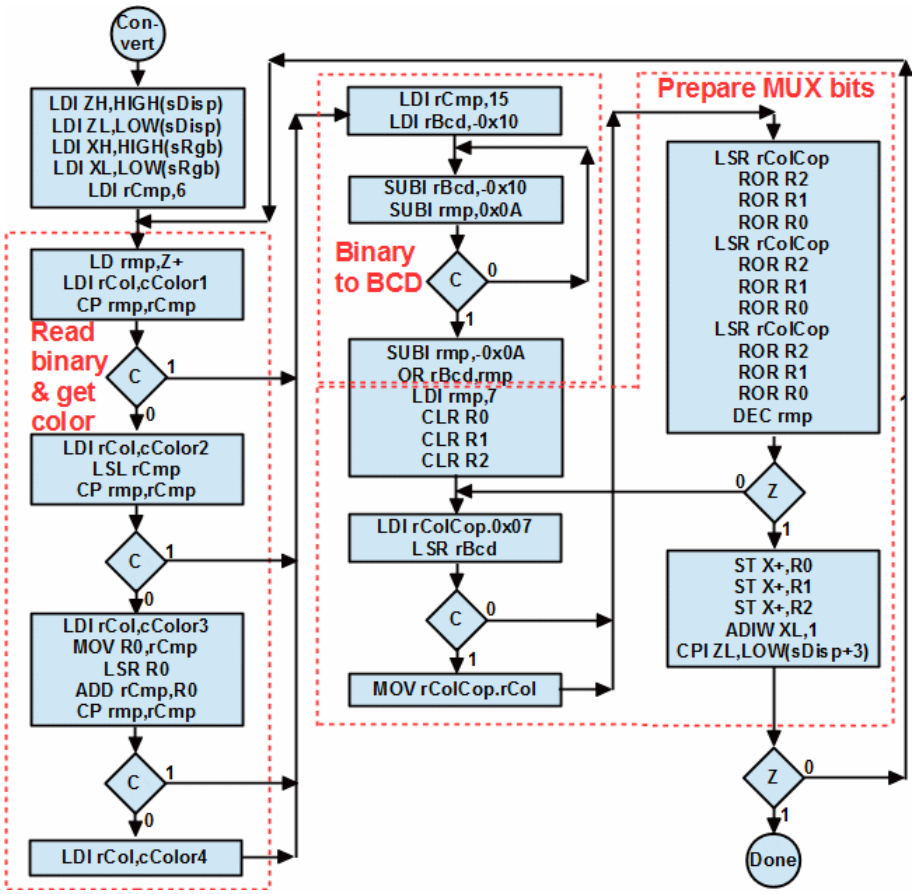
So funktioniert das Zusammenstellen der Multiplex-Bits aus den Farben in rCol. Dazu wird ein Puffer in den Registern R2:R1:R0 angelegt und mit Nullen befüllt. Ein Bit-Zähler in rmp wird auf 7 eingestellt.

Dann beginnt das Einschieben der Farbbits. Dazu wird eine Kopie von rCol im Register rColCopy angelegt und drei mal links geschoben. Das in der Carry-Flagge befindliche Farbbit wird dann jeweils in die Register R2:R1:R0 von rechts her eingerollt.



In der Darstellung sind die ersten beiden Einrollvorgänge dargestellt. Sind alle sieben Einrollvorgänge absolviert, wird der Puffer in R2:R1:R0 an der aktuellen Position im Register XH:XL in das SRAM geschrieben.

Das ist der gesamte Algorithmus der Umwandlung der binären Zeitinformation ab dem Register ZH:ZL (zeigt zu Beginn auf die Position sDisp, kann sowohl auf eine SRAM-Position als auch ein Register mit Stunden/Minuten/Sekunden zeigen).



1. Es beginnt mit dem Einlesen des Bytes, auf das ZH:ZL zeigt.
2. Durch Vergleich mit dem Vergleichswert in rCmp wird wie oben dargestellt der Farbwert ermittelt.
3. Der eingelesene Zeitwert wird dann wie oben dargestellt in BCD im Regiter rBcd umgewandelt.
4. Dann werden die sieben BCD-Bits ermittelt, als Farbbits in den Puffer R2:R1:R0 geschoben und in das SRAM kopiert. Das Addieren von 1 im Zeiger XH:XL überspringt das Anodenbyte.
5. Sind alle drei Zeitbytes behandelt, ist der Algorithmus beendet.

Der gesamte Algorithmus benötigt bei 4 MHz Takt ca. 150 µs, also deutlich kürzer als ein Multplex-Durchgang oder auch ein AD-Wandler-Durchgang andauert.

4.4.5 Einstellung der Zeit

Ist die Flagge **bAdc** gesetzt, sind ca. 6,6 ms vergangen.

Bedingt durch die vielfältigen Funktionen, die in die AD-Wandler-Flaggenbehandlung eingebunden sind, ergibt sich ein umfangreicher Algorhythmus. Noch einfach gestaltet sich der Ablauf solange die Zeiteinstellung nicht aktiv ist: das MSB der AD-Wandler-Summe (0..255) wird mit 250 multipliziert und das 16-Bit-Resultat durch vier geteilt (0..15.937). Dieser Wert wird in den Compare-B-Port des Timers TC1 geschrieben. Der Compare-B-Interrupt bewirkt bei jedem Eintreten, dass die Anodentreiber-Ausgänge vorzeitig abgeschaltet werden. Ist der Wert groß, erfolgt die Abschaltung spät im CTC-Zyklus und die Leds sind hell (z. B. 15.937 / 16.000 = 99,6% Helligkeit). Ist der Wert niedrig, sind die Leds die meiste Zeit aus (z. B. 0 = 1 / 16.000 = 0,006% Helligkeit).

Abschließend wird noch abgefragt, ob die Taste 1 gedrückt ist und die Zeiteinstellung eingeleitet werden soll. Falls ja (Eingangspin ist aktiv low), wird die Flagge **bSetA** aktiviert und die beiden Flaggen **bSetM**, **bSetS** und **bSetO** gelöscht und die Eingabe von Stunden erfolgt. Das MSB der AD-Wert-Summe wird mit 24 multipliziert und dieser Stundenwert in das Eingaberegister für Stunden geschrieben. Danach wird das Prellschutz-Register **rBounce** auf seinen Anfangswert gesetzt. Es bewirkt, dass jeder weitere Tastendruck ignoriert wird, solange dieses nicht auf Null heruntergezählt worden ist.

Ist beim Vorliegen der MSB-Summe das Bit **bSetA** gesetzt, wird zunächst geprüft, ob der Prellschutzzähler auf Null steht. Ist dies nicht der Fall, werden die Tasten 1 und 2 abgefragt. Sind beide inaktiv (11, inaktiv high), wird der Zähler abwärts gezählt. Ist eine der beiden Tasten noch immer gedrückt, beginnt die Prellschutzperiode erneut von vorne.

Ist die Prellschutzperiode erfolgreich absolviert, wird das Bit **rTSetO** abgefragt. Dieses ist entweder gesetzt, wenn die letzte Stelle (Sekunden) fertig eingegeben ist und Taste 2 gedrückt war oder wenn die Rückwärtstaste 2 während der Eingabe der Stunden gedrückt war. In beiden Fällen war abzuwarten, bis die Tasten lange genug losgelassen wurden. Erst danach wird der Eingabemodus durch Rücksetzen des **rTSet**-Bits verlassen und wieder die aktuelle Zeit angezeigt.

War beim Vorliegen der AD-Wert-Summe das Bit **bSetA** gesetzt, der Prellschutzzähler **rBounce** auf Null und **bSetO** nicht gesetzt, wird die Taste 1 abgefragt. Ist diese gedrückt (Eingang = 0), dann liegt ein Rückwärtsbefehl vor. Je nachdem,

- ob gerade Sekunden eingegeben werden (**bSetS**=1): Falls ja, wird dieses Bit rückgesetzt, der MSB des ADC-Werts mit 60 multipliziert und in die Eingabeminuten geschrieben,
- ob gerade Minuten eingegeben werden (**bSetM**=1): Falls ja, wird dieses Bit rückgesetzt, der MSB des ADC-Werts mit 24 multipliziert und in die Eingabestunden geschrieben,
- ob gerade Stunden eingegeben werden (**bSetM**=0): Das **bSetO**-Bit wird gesetzt, um nach Beenden des Tastendrucks die Zeiteinstellung zu beenden und wieder zur Anzeige der (inzwischen weiter gelaufenen) Zeit zurückzukehren,

In jedem Fall wird nach diesem Tastendruck der Prellzähler **rBounce** mit seinem Startwert geladen.

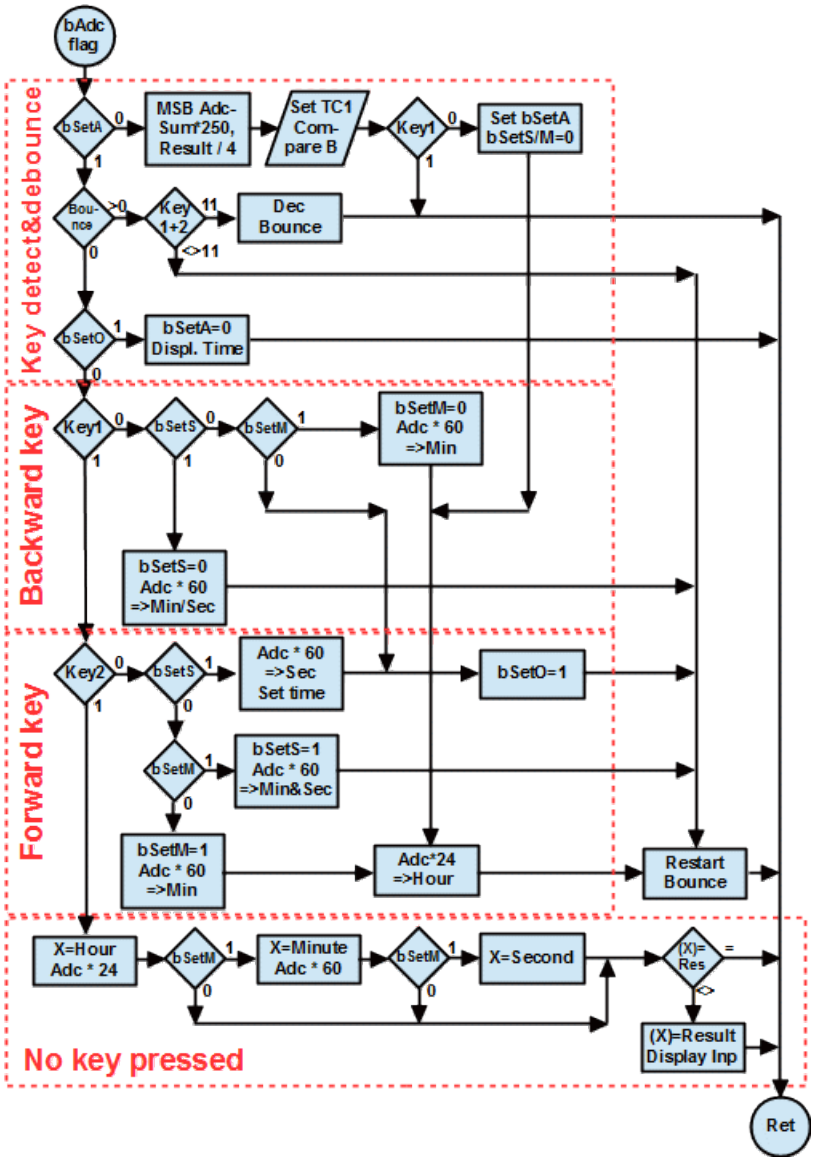
War die Taste 1 nicht gedrückt, wird die Taste 2 abgefragt. Ist diese gedrückt, liegt ein Vorwärtsbefehl vor. Je nachdem,

- ob gerade Sekunden eingegeben werden (**bSetS**=1): Falls ja, ist die Zeit fertig eingestellt. Dazu wird der aktuelle AD-Wandlerwert mit 60 multipliziert und in die Sekunden der aktuellen Zeit geschrieben. Die Stunden und Minuten der Einstellungswerte werden in diejenigen der aktuellen Zeit kopiert und das **rTSetO**-Bit gesetzt, damit nach Beenden des Tastendrucks die Beendigung des Zeiteinstellmodus erfolgen wird.
- ob gerade Minuten eingegeben werden (**bSetM**=1): Falls ja, wird der MSB der AD-Wert-Summe mit 60 multipliziert und sowohl in das Minuten- als auch das Sekunden-Register geschrieben und das Sekundenbit **bSetS** wird gesetzt.
- ob gerade Stunden eingegeben werden (**bSetM**=0): Falls ja, wird das Minutenbit gesetzt und der MSB-Wert der AD-Summe mit 60 multipliziert und in das Minutenregister geschrieben. Ferner wird dieses mit 24 multipliziert und in das Stundenregister geschrieben.

In allen Fällen wird wieder das Prellschutzregister neu gestartet.

Wenn weder Taste 1 noch Taste 2 gedrückt waren, wird im Registerpaar X die aktuelle Eingabeposition (Stunden, Minuten oder Sekunden) eingestellt und die Messwertsumme mit 24 oder 60 multipliziert. Hat sich der Wert geändert, wird er neu geschrieben, in Kathodenbits umgewandelt und angezeigt.

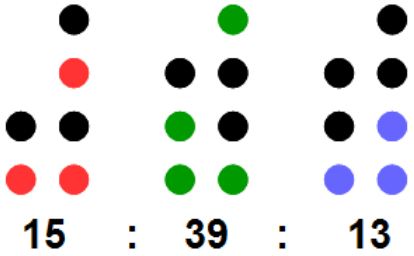
Nicht eingezeichnet ist der Algorhythmus, der das Time-Out der Zeiteinstellung feststellt. Dieser erfolgt mit dem Registerpaar **rInpToH:rInpToL**, das bei jedem Durchlauf um Eins erhöht wird. Erreicht es durch 16-Bit-Überlauf Null, wird der Eingabemodus direkt und unmittelbar beendet. Jeder Tastendruck setzt diese beiden Register auf ihren Anfangswert. Maximal kann das Timeout auf bis zu sieben Minuten eingestellt werden (Konstante **InputTimeOutMinutes**).



Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

Lob, Tadel, Fehlermeldungen, Genöle und Geschimpfe oder Spam bitte über das Kommentarformular an mich.

Quellcode



AVR-Anwendungen

RGB-Dünne-Berg-Uhr mit ATmega16

Software für die RGB-BCD-Uhr

Assembler-Quellcode für die RGB-Dünne-Berg-Uhr

Der Original-Quellcode im Assembler-Format ist hier.

```
;
; ***********************************
; * RGB-Duenne-Berg-Uhr ATmega16    *
; * Version 1.0 Januar 2019         *
; * (C)2019 by avr-asm-tutorial.net *
; ***********************************
;
.nolist
.include "m16adef.inc" ; Define device ATmega16A
.list
;
; ************************************
;  D E B U G G I N G   S W I T C H E S
; ************************************
;
; Accelerate watch by a factor
.equ cAccel = 1 ; Acceleration of clock, 1..1000
;    (should be 1 in final version)
;
.equ Yes = 1 ; Yes for all switches
.equ No = 0 ; No for all switches
;
; Light all LEDs in all colors one by one in the rows:
;   1. blue - green - red for delay seconds each
;   2. ones, twos, fours, eights, tens, twenties, fourties
;   3. seconds - minutes - hours
.equ dbg_leds = No ; Should be no in final version
.equ dbg_leds_delay = 10 ; On-Time in 50 ms multiples
;
; Debug the ADC: Set hours to MSB result
.equ dbg_adc = No ; Should be no in final version
;
; Debug the keys:
;   Leds dark if no key pressed
;   Set hour tens red on key 1
;   Set hour twenties on key 2
;   (Disables all normal operation!)
.equ dbg_key = No
;
```

```
; Debug the color balance:
;   Lights all three LEDs around and around
;   (Disables all normal operation!)
.equ dbg_colbal = No
;
; Monitoring of registers during input phase
.equ dbg_moni_flags = No ; Monitor the flags
.equ dbg_moni_bounce = No ; Monitor the bounce counter
;
;
; *********************************
;        H A R D W A R E
; *********************************
;
; Device: ATmega16A, Package: 40-pin-PDIP
;              __________
;        1 /          |40
;   AH o--|PB0     PA0|--o POT
;   AM o--|PB1     PA1|--o NC
;   AS o--|PB2     PA2|--o NC
;   K1 o--|PB3     PA3|--o C0B
;   K2 o--|PB4     PA4|--o C0G
; MOSI o--|PB5     PA5|--o C0R
; MISO o--|PB6     PA6|--o C1B
;  SCK o--|PB7     PA7|--o C1G
;  RES o--|RESET  AREF|--o AREF
;  +5V o--|VCC     GND|--o 0V
;   0V o--|GND    AVCC|--o AVCC
; 4MHz o--|XTAL2   PC7|--o C4B
; XTAL o--|XTAL1   PC6|--o C3R
;  C4G o--|PD0     PC5|--o C3G
;  C4R o--|PD1     PC4|--o C3B
;  C5B o--|PD2     PC3|--o C2R
;  C5G o--|PD3     PC2|--o C2G
;  C5R o--|PD4     PC1|--o C2B
;  C6B o--|PD5     PC0|--o C1R
;  C6G o--|PD6     PD7|--o C6R
;       20|_____________|21
;
;
; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
.equ pRgbC1O = PORTA ; RGB1 Cathode Output port
.equ pRgbC1D = DDRA ; RGB1 Cathode Direction port
.equ bRgbC1M = 0xF8 ; RGB1 Cathode Output Mask
.equ pRgbC2O = PORTC ; RGB2 Cathode Output port
.equ pRgbC2D = DDRC ; RGB2 Cathode Direction port
.equ pRgbC2M = 0xFF ; RGB2 Cathode Output Mask
.equ pRgbC3O = PORTD ; RGB3 Cathode Output port
.equ pRgbC3D = DDRD ; RGB3 Cathode Direction port
.equ pRgbC3M = 0xFF ; RGB3 Cathode Output Mask
.equ pRgbAO = PORTB ; RGB Anode Output port
.equ pRgbAD = DDRB ; RGB Anode Direction port
.equ pRgbAM = 0x07 ; RGB Anode Mask
.equ pKeyI = PINB ; Key input port
.equ bKey1 = PINB3 ; First key
.equ bKey2 = PINB4 ; Second key
;
; ***************************************
```

```
;   A D J U S T A B L E   C O N S T A N T S
; ****************************************
;
.equ clock=4000000 ; Define clock frequency
;
; Constants for color composition
.equ bBlue = 0 ; Blue led bit
.equ bGreen = 1 ; Green led bit
.equ bRed = 2 ; Red led bit
;
; Colors for the four quarters
.equ cColor1 = 255-(1<<bBlue) ; Color quarter 1
.equ cColor2 = 255-(1<<bGreen) ; Color quarter 2
.equ cColor3 = 255-(1<<bRed) ; Color quarter 3
.equ cColor4 = 255-((1<<bRed)|(1<<bBlue)) ; Color quarter 4
; Error checking of colors
.if ((cColor1&0x07)==0)||((cColor2&0x07)==0)||((cColor3&0x07)==0)||
((cColor4&0x07)==0)
  .error "One of the colors has three color bits at zero!"
  .endif
;
; Start time of watch: time to be set at start-up
.equ starthour = 20 ; Start at 20 hours
.equ startminute = 0 ; Start at zero minute
.equ startsecond = 0 ; Start at zero seconds
;
; When key input active: blink hour/minute/second
.equ BlinkPeriod = 85 ; 85% on, 15% off
;
; Key input bouncing parameter
.equ BouncePeriod = 50 ; Debouncing period for keys in ms
;
; Input active time-out
.equ InputTimeOutMinutes = 5 ; Time in minutes to skip input period
.if InputTimeOutMinutes > 7
  .error "Input time-out minutes too large!"
  .endif
;
; ************************************************
;  F I X E D  &  D E R I V I V E D  C O N S T A N T S
; ************************************************
;
; Clock signals for second increase and MUX
;   Clock > Presc > CTC > SecDiv > Seconds
;   4 MHz    1     16000     250
.equ cTc1Presc = 1 ; TC1 Prescaler
.equ cTc1Clk = clock / cTc1Presc ; Frequency TC1
.equ cTc1CompA = 15999 ; Int for Mux
.equ cSecDiv = cTc1Clk/(cTc1CompA+1) ; TC0 Second divider
;
; Switch leds off after blink period
.equ cBlinkOff = ((100-BlinkPeriod)*cSecDiv+50) / 100 ; Second divider
;
; ADC constants
.equ cAdcCnt = 64 ; Sum up 64 ADC results
.equ cAdcPs = 32 ; ADC prescaler
.equ cAdcN = 13 ; Number of cycles
.equ cAdcTime = (cAdcCnt*cAdcPs*cAdcN*1000)/clock ; Time in ms
.equ cInpTO = 65536-(clock/cAdcCnt/cAdcPs/cAdcN)*60*InputTimeOutMinutes
.if cInpTO < 0
  .error "Input time-out too long!"
```

```
  .endif
.equ cBounce = (BouncePeriod+cAdcTime/2)/cAdcTime+1 ; Bounce counter
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; used: R2:R1:R0 for binary conversion to color
.def rInpToL = R3 ; Time out input, LSB
.def rInpToH = R4 ; dto., MSB
.def rHrInp = R5 ; Input hour
  .equ sInput = 5
.def rMinInp = R6 ; Input minute
.def rSecInp = R7 ; Input second
.def rCmp = R8 ; Compare value for conversion
.def rConvCnt = R9 ; Counter for time conversion
.def rAdcRes = R10 ; MSB of ADC sum
.def rAdcL = R11 ; ADC sum, LSB
.def rAdcH = R12 ; dto., MSB
.def rAdcCnt = R13 ; ADC counter
.def rSecDiv = R14 ; Seconds divider
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
;  Flags for flow control
.def rFlag = R18 ; Flag register
  ; Those four bits control key input
  ;   OSM_ = xxx0 = Normal operation, no time setting, no monitoring
  ;        = 0001 = Time setting hours, monitoring = Led one green
  ;        = 0011 = dto., minutes, monitoring = Led two blue
  ;        = 0111 = dto., seconds, monitoring = Led four green
  ;        = 1xx1 = dto., end time setting, monitoring = Leds eight/ten/twenty
  .equ bSetA = 0 ; Key flag, for hour setting
  .equ bSetM = 1 ; Flag for minute setting
  .equ bSetS = 2 ; Flag for second setting
  .equ bSetO = 3 ; Flag for end of time setting
  ; bBlink set when bSetA=1 and rSecDiv cycle reaches switching value
  .equ bBlink = 4 ; Switch active leds off
  ; bNoUpd set when bBlink is set, cleared at second start
  .equ bNoUpd = 5 ; Do not update during off-period
  ; Set by TC1A-ISR when a second is over
  .equ bSec = 6 ; Flag second over
  ; Set by ADC-Complete ISR when 64 measurements completed
  .equ bAdc = 7 ; ADC flag
.def rHr = R19 ; Time hours
  .equ sTime = 19 ; Pointer to time info for displaying time
.def rMin = R20 ; Time minutes
.def rSec = R21 ; Time seconds
.def rCol = R22 ; Color
.def rColCop = R23 ; Copy of rCol
.def rBcd = R24 ; BCD coded time
.def rBounce = R25 ; Debouncing counter for key inputs
; used: R27:R26 = X for conversion to cathode bits outside ints
; used: R29:R28 = Y for pointer for multiplexing cycle
; used: R31:R30 = Z for diverse purposes outside ints
;
; *********************************
;           S R A M
; *********************************
;
.dseg
```

```
.org SRAM_START
;
; MUX buffer
sRgb:
.byte 4 ; Four bytes: RGBC1, RGBC2, RGBC3,AGBA, Hours
.byte 4 ; Four bytes: RGBC1, RGBC2, RGBC3,AGBA, Minutes
.byte 4 ; Four bytes: RGBC1, RGBC2, RGBC3,AGBA, Seconds
sRgbEnd:
;
; *********************************
;           C O D E
; *********************************
;
.cseg
.org 000000
;
; ***********************************
;  R E S E T  &  I N T - V E C T O R S
; ***********************************
        rjmp Main ; Reset vector
        nop
        reti ; INT0, unused
        nop
        reti ; INT1, unused
        nop
        reti ; OC2, unused
        nop
        reti ; OVF2, unused
        nop
        reti ; ICP1, unused
        nop
        rjmp Oc1AIsr ; OC1A, MUX and second counting
        nop
        rjmp Oc1BIsr ; OC1B, anode driver off for dimming
        nop
        reti ; OVF1, unused
        nop
        reti ; OVF0, unused
        nop
        reti ; SPI, unused
        nop
        reti ; URXC, unused
        nop
        reti ; UDRE, unused
        nop
        reti ; UTXC, unused
        nop
        rjmp AdccIsr ; ADCC, ADC cycle
        nop
        reti ; ERDY, unused
        nop
        reti ; ACI, unused
        nop
        reti ; TWI, unused
        nop
        reti ; INT2, unused
        nop
        reti ; OC0, unused
        nop
        reti ; SPMR, unused
        nop
```

```
;
; ****************************************
;  I N T - S E R V I C E   R O U T I N E S
; ****************************************
;
; TC1 Compare Match A Interrupt Service Routine
Oc1AIsr:
  in rSreg,SREG ; Save SREG
  dec rSecDiv ; Decrease second divider
  brne Oc1AIsr1 ; Not zero, perform MUX
  ldi rimp,cSecDiv ; Restart seconds divider
  mov rSecDiv,rimp
  sbr rFlag,1<<bSec ; Set seconds flag
  cbr rFlag,1<<bNoUpd ; Updates on
Oc1AIsr1:
  ldi rimp,0x1F ; Clear anode drivers
  out pRgbAO,rimp
  ld rimp,Y+ ; Read first cathodes
  out pRgbC1O,rimp ; Write first RGB port
  ld rimp,Y+ ; Read second cathodes
  out pRgbC2O,rimp ; Write second RGB port
  ld rimp,Y+ ; Read third cathodes
  out pRgbC3O,rimp ; Write third RGB port
  ld rimp,Y+ ; Read anodes
  out pRgbAO,rimp ; Write anode
  cpi YL,sRgbEnd ; End of buffer?
  brne Oc1AIsr2 ; No
  ldi YH,High(sRgb) ; Restart Mux
  ldi YL,Low(sRgb)
Oc1AIsr2:
  sbrs rFlag,bSetA ; Key input active?
  rjmp Oc1AIsr3 ; No
  ldi rimp,cBlinkOff ; Seconds divider reached off?
  cp rimp,rSecDiv ; Blink cycle reached
  brne Oc1AIsr3 ; No, skip flag setting
  sbr rFlag,(1<<bBlink)|(1<<bNoUpd) ; Set blink flag to turn input off
Oc1AIsr3:
  out SREG,rSreg ; Restore SREG
  reti
;
; OC1B Interrupt Service Routine
;   switches anode driver off for dimming
Oc1BIsr:
  ldi rimp,0b00011111 ; Anode driver bits to one
  out pRgbAO,rimp ; in anode driver port
  reti
;
; ADC Conversion Complete Interrupt Service Routine
;   sums up 64 results, if complete: copy MSB, clear sum, set flag
;   restart ADC conversion
AdccIsr:
  in rSreg,SREG ; Save SREG
  in rimp,ADCL ; Read ADC result LSB
  add rAdcL,rimp ; add to sum LSB
  in rimp,ADCH ; Read MSB
  adc rAdcH,rimp ; add to MSB
  dec rAdcCnt ; Decrease counter
  brne AdccIsr1 ; Not zero, restart ADC
  mov rAdcRes,rAdcH ; Copy MSB result
  clr rAdcL ; Restart sum LSB
  clr rAdcH ; dto., MSB
```

```
  ldi rimp,cAdcCnt ; Restart counter
  mov rAdcCnt,rimp ; in counter register
  sbr rFlag,1<<bAdc ; Set ADC flag
AdccIsr1:
  ; Restart ADC
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS0)
  out ADCSRA,rimp ; in ADC control port
  out SREG,rSreg ; Restore SREG
  reti
;
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
Main:
        ldi rmp,High(RAMEND) ; End of SRAM, MSB
        out SPH,rmp ; Init MSB stack pointer
        ldi rmp,Low(RAMEND) ; End of SRAM, LSB
        out SPL,rmp ; Init LSB stack pointer
  ; Init SRAM MUX buffer
  ldi XH,High(sRgb) ; Point to SRAM buffer, MSB
  ldi XL,Low(sRgb) ; dto., LSB
Main1:
  ldi rmp,0b11111000 ; First buffer byte
  st X+,rmp ; Cathodes 1 off
  ldi rmp,0xFF ; all others FF
  st X+,rmp ; Cathodes 2 off
  st X+,rmp ; Cathodes 3 off
  adiw XL,1 ; Jump over anode byte
  cpi XL,Low(sRgbEnd) ; End of buffer?
  brne Main1 ; No, go on
  ldi rmp,0b00011110 ; Anode driver hours
  sts sRgb+3,rmp ; Hour anode active
  ldi rmp,0b00011101 ; Anode driver miinutes
  sts sRgb+7,rmp ; Minute anode active
  ldi rmp,0b00011011 ; Anode driver seconds
  sts sRgb+11,rmp ; Second anode active
  ; Init the I/O ports
  ldi rmp,pRgbAM|(1<<bKey1)|(1<<bKey2) ; RGB Anode Mask plus key pull-ups
  out pRgbAO,rmp ; RGB Anode Output port
  ldi rmp,pRgbAM ; RGB Anode Mask
  out pRgbAD,rmp ; RGB Anode Direction port
  ldi rmp,bRgbC1M ; RGB1 Cathode Output Mask
  out pRgbC1O,rmp ; RGB1 Cathode Output port
  out pRgbC1D,rmp ; RGB1 Cathode Direction port
  ldi rmp,pRgbC2M ; RGB2 Cathode Output Mask
  out pRgbC2O,rmp ; RGB2 Cathode Output port
  out pRgbC2D,rmp ; RGB2 Cathode Direction port
  ldi rmp,pRgbC3M ; RGB3 Cathode Output Mask
  out pRgbC3O,rmp ; RGB3 Cathode Output port
  out pRgbC3D,rmp ; RGB3 Cathode Direction port
;
; ***********************************
;  H A R D W A R E   D E B U G G I N G
; ***********************************
;
; Routines for hardware debugging
;   Those do not perform normal counting but
;   end in indefinite loops!
.if dbg_key == Yes
  ; Debug the attached keys
```

```
  ;   Set hour tens red if key 1 pressed
  ;   Set hour twenties red if key 2 pressed
          ldi rmp,0b00011110 ; Anode hours on
          out pRgbAO,rmp
        dbg_key_loop:
          sbic pKeyI,bKey1 ; Skip if key 1 low
          sbi pRgbC3O,PORTD1 ; Red LED off
          sbis pKeyI,bKey1 ; Skip if key 1 high
          cbi pRgbC3O,PORTD1 ; Red LED on
          sbic pKeyI,bKey2 ; Skip if key 2 low
          sbi pRgbC3O,PORTD4 ; Red LED off
          sbis pKeyI,bKey2 ; Skip if key 2 high
          cbi pRgbC3O,PORTD4 ; Red Led on
          rjmp dbg_key_loop ; Repeat on and on again
  .endif
.if dbg_colbal == Yes
  ; Debug the color balance of all leds
  ;   Switch all leds to white one by one
  ;   and fastly change to the next led
  ;   Colors should be clean white, but
  ;   only 1/20th brightness
  ColBal_loop:
    ldi rMin,0xFF ; Start with upper leds off
    ldi rHr,0xFF
  ColBal_loop1:
    ldi rBcd,0b00011011 ; Start anodes with seconds
  ColBal_loop2:
    ldi rSec,0b11000000 ; Start with seconds ones
    ldi rmp,20 ; 20 leds to cycle
  ColBal_loop3:
    out pRgbC1O,rSec ; Set cathodes, seconds
    out pRgbC2O,rMin ; dto., minutes
    out pRgbC3O,rHr ; dto., hours
    out pRgbAO,rBcd ; Set anode driver
    ori rSec,0b00000111 ; Set all three lower bits
    lsl rSec ; Shift to next led, seconds
    rol rMin ; dto., minutes
    rol rHr ; dto., hours
    lsl rSec ; and to overnext led
    rol rMin
    rol rHr
    lsl rSec ; and to third led
    rol rMin
    rol rHr
    dec rmp
    brne ColBal_loop3 ; Continue led out and shift
    ori rBcd,0b00100000 ; Set next key to one
    lsr rBcd ; Next anode
    brcs ColBal_loop2 ; Not at the end
    rjmp ColBal_loop1 ; Restart all new
  .endif
.if dbg_leds == Yes
  ; Debug the leds: Set the leds
  ;   a) blue-green-red,
  ;   b) ones, twos, fours, eights, tens, twentys, fourtys
  ;   c) seconds, minutes, hours
  Led_loop1:
    ldi rMin,0xFF ; Upper leds off
    ldi rHr,0xFF
    ldi rBcd,0b00011011 ; Start anodes seconds
  Led_loop2:
```

```
    ldi rSec,0b11110000 ; Start with seconds ones
    ldi rmp,21 ; 21 leds to cycle
    sbrs rBcd,0 ; Hour cycle?
    ldi rmp,18 ; 18 leds only
  Led_loop3:
    out pRgbC1O,rSec ; Set cathodes, second
    out pRgbC2O,rMin ; dto., minute
    out pRgbC3O,rHr ; dto., hour
    out pRgbAO,rBcd ; Set anode driver
    ldi rCol,dbg_leds_delay ; Multiple 50 ms delay
  Led_loop4:
    ldi ZH,High((5*clock)/400) ; Delay loop 50 ms, MSB
    ldi ZL,Low((5*clock)/400) ; dto., LSB
  Led_loop5:
    sbiw ZL,1 ; Count down, 2 clock cycles
    brne Led_loop5 ; 2 clock cycles for branching
    ; 4 clock cycles = 1 us @4 MHz
    ; 50.000 us per loop
    dec rCol ; Count delay down
    brne Led_loop4 ; Next delay cycle
    ori rSec,0b00000100 ; Set blue cathode led 0 off
    lsl rSec ; Next cathode, second
    rol rMin ; Next cathode, minute
    rol rHr ; Next cathode, hour
    dec rmp ; Downcount led
    brne Led_loop3 ; Next led output
    lsr rBcd ; Next anode
    brcs Led_loop2 ; Restart with first led
    rjmp Led_loop1 ; Restart at the beginning
  .endif
; ***************************************
;  C O N T I N U E   N O R M A L   I N I T
; ***************************************
  ; Init the start time of the watch
  ldi rHr,starthour
  ldi rMin,startminute
  ldi rSec,startsecond
  ; Init the ADC
  ldi rmp,1<<REFS0 ; VREF=AVCC, ADC channel 0
  out ADMUX,rmp ; to MUX port
  clr rAdcL ; Restart sum LSB
  clr rAdcH ; dto., MSB
  ldi rmp,cAdcCnt ; Restart counter
  mov rAdcCnt,rmp
  ; Start the ADC: Int Enable, clock prescaler = 32
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS0)
  out ADCSRA,rmp ; in ADC control port
  ; Init TC1 as MUX/second timer/anode driver off
  ldi rmp,High(cTC1CompA/cAccel) ; CTC compare value, MSB
  out OCR1AH,rmp ; to compare port MSB
  ldi rmp,Low(cTC1CompA/cAccel) ; dto., LSB
  out OCR1AL,rmp ; dto., LSB
  clr rmp ; Switch LEDs to lowest brightness
  out OCR1BH,rmp ; MSB
  out OCR1BL,rmp ; LSB
  ldi rmp,0 ; WGM10 and WGM11 to 0
  out TCCR1A,rmp
  ldi rmp,(1<<WGM12)|(1<<CS10) ; CTC on compare A, Prescaler=1
  out TCCR1B,rmp ; Start TC1
  ldi rmp,(1<<OCIE1A)|(1<<OCIE1B) ; Enable compare match interrupts
  out TIMSK,rmp ; in timer interrupt mask
```

```
  ; Init flags
  clr rFlag ; All flags off
  ; Init sleep
  ldi rmp,1<<SE ; Enable sleep idle
  out MCUCR,rmp ; in master control port
        sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
;
Loop:
  sleep ; Go to sleep
  nop ; Dummy for wake-up
  sbrc rFlag,bSec ; Second flag set?
  rcall Second ; Perform second increase
  sbrc rFlag,bAdc ; ADC conversion sum complete?
  rcall AdcRdy ; Perform ADC service
  sbrc rFlag,bBlink ; Off-period reached?
  rcall Blink ; Switch leds off
  rjmp Loop
;
; *****************************************
;  T I M E   I N C R E A S E  &  D I S P L A Y
; *****************************************
;
Second:
  cbr rFlag,1<<bSec ; clear flag
  ; Increase time by one second
  inc rSec ; Increase seconds
  cpi rSec,60 ; End of minute?
  brcs DisplayTimeInp ; No, output time
  clr rSec ; Restart second
  inc rMin ; Next minute
  cpi rMin,60 ; End of hour?
  brcs DisplayTimeInp ; No, output time
  clr rMin ; Restart minute
  inc rHr ; Next hour
  cpi rHr,24 ; End of day?
  brcs DisplayTimeInp ; No, output time
  clr rHr ; Restart hour
DisplayTimeInp:
  ; Display current time or the input time?
  ldi ZH,High(sTime) ; Point to time hours
  ldi ZL,Low(sTime)
  ; Output input time instead?
  sbrs rFlag,bSetA ; Key input active?
  rjmp Convert ; No, display current time
DisplayInput:
  ldi ZH,High(sInput) ; Output input time, MSB
  ldi ZL,Low(sInput) ; dto., LSB
;
; Convert time/input to display
Convert:
  ldi XH,High(sRgb) ; Point X to SRAM buffer, MSB
  ldi XL,Low(sRgb) ; dto., LSB
  ldi rmp,3 ; Three bytes to convert
  mov rConvCnt,rmp ; Set convert counter
  ldi rmp,24/4 ; Comparer for hours
  mov rCmp,rmp ; To compare register
ReadDateTime:
```

```
  ld rmp,Z ; Read the next data byte time/input
  ldi rCol,cColor1 ; Set color for quarter 1
  cp rmp,rCmp ; Compare with first quarter value
  brcs ToBcd ; Smaller, convert to BCD
  ldi rCol,cColor2 ; Set color for quarter 2
  lsl rCmp ; Compare value * 2
  cp rmp,rCmp ; Compare with second quarter value
  brcs ToBcd ; Smaller, convert to BCD
  ldi rCol,cColor3 ; Set color for quarter 3
  mov R0,rCmp ; Copy compare value
  lsr R0 ; Divide by two
  add rCmp,R0 ; Add to compare value, 3/4
  cp rmp,rCmp ; Compare with third quarter value
  brcs ToBcd ; Smaller, convert to BCD
  ldi rCol,cColor4 ; Set color for quarter 4
ToBcd:
  ldi rmp,60/4 ; Set compare value for minutes/seconds to 15
  mov rCmp,rmp ; in compare register
  ; Read and convert value to BCD in rBcd
  ld rmp,Z+ ; Read time/input byte again, point to next
  ldi rBcd,-0x10 ; Tens to minus 1
ToBcd1:
  subi rBcd,-0x10 ; Add tens
  subi rmp,10 ; Subtract 10 from value
  brcc ToBcd1 ; No carry, continue adding tens and subtracting
  subi rmp,-10 ; Add ten to restore last subtraction
  or rBcd,rmp ; Add the ones to the tens
ToRgb:
  ldi rmp,7 ; Seven times three colors
  clr R0 ; Clear buffer, byte 1
  clr R1 ; dto., byte 2
  clr R2 ; dto., byte 3
ToRgb1:
  ldi rColCop,0x07 ; Color all ones = leds of
  lsr rBcd ; Shift next BCD bit to carry
  brcc ToRgb2 ; Bit is zero, led off
  mov rColCop,rCol ; Bit is one, led to color
ToRgb2:
  ; Shift the three color bits into the buffer
  lsr rColCop ; Shift first color bit to carry
  ror R2 ; and into buffer from high to low
  ror R1
  ror R0
  lsr rColCop ; Shift second color bit to carry
  ror R2 ; and into buffer from high to low
  ror R1
  ror R0
  lsr rColCop ; Shift third color bit to carry
  ror R2 ; and into buffer from high to low
  ror R1
  ror R0
  dec rmp ; Count led down
  brne ToRgb1 ; Next led
  st X+,R0 ; Store buffer in SRAM for mux display, byte 1
  st X+,R1 ; dto., byte 2
  st X+,R2 ; dto., byte 3
  adiw XL,1 ; jump over anode byte
  dec rConvCnt ; Decrease counter
  brne ReadDateTime ; Read and convert next byte
  ret ; Conversion done
;
```

```
; ************************************
;  A D C   C Y C L E   C O M P L E T E D
; ************************************
;
; Perform reaction on ADC sum complete
AdcRdy:
  cbr rFlag,1<<bAdc ; Clear the ADC flag
  .if dbg_adc == Yes
    ; Display ADC result as 0..23 on hour position
    ldi rmp,24 ; Convert to hour
          rcall Multiply
          mov rHr,ZH ; Write to hour
          ldi ZH,High(sTime) ; Point Z to actual time, MSB
          ldi ZL,Low(sTime) ; dto., LSB
          rcall Convert ; Display the current time
          .endif
  sbrc rFlag,bSetA ; No time setting active?
  rjmp AdcDebounce ; No, check debounce
  ; Input inactive, set dim value and check key 1 pressed
  ldi rmp,250 ; Multiply ADC MSB by 250
  rcall Multiply
  lsr ZH ; Divide by 2, MSB
  ror ZL ; dto., LSB
  lsr ZH ; Divide by 4, MSB
  ror ZL ; dto., LSB
  out OCR1BH,ZH ; Set compare B port in TC1, MSB
  out OCR1BL,ZL ; dto., LSB
  sbic pKeyI,bKey1 ; Key 1 pressed?
  rjmp AdcRdyRet ; No
  sbr rFlag,1<<bSetA ; Set Set flag
  cbr rFlag,(1<<bSetS)|(1<<bSetM)|(1<<bSetO) ; Clear the other flags
  mov rSecInp,rSec ; Copy seconds
  mov rMinInp,rMin ; Copy minutes
  rjmp AdcSetHour ; Set hour to ADC MSB result and debounce
AdcDebounce:
  tst rBounce ; Bouncing period ended?
  breq AdcOFlag ; Yes, check O flag set
  sbis pKeyI,bKey1 ; Key 1 active?
  rjmp AdcRestartBounce ; Yes, restart bounce counter
  sbis pKeyI,bKey2 ; Key 2 active?
  rjmp AdcRestartBounce ; Yes, restart bounce counter
  dec rBounce ; Decrease debounce counter
  rjmp AdcRdyRet ; Done
AdcOFlag:
  sbrs rFlag,bSetO ; O flag set?
  rjmp AdcKey1 ; No, check key 1 pressed
  cbr rFlag,1<<bSetA ; Clear time set flag
  rjmp DisplayTimeInp
AdcKey1:
  sbic pKeyI,bKey1 ; Key 1 pressed?
  rjmp AdcKey2 ; No, check key 2
  sbrs rFlag,bSetS ; S flag set?
  rjmp AdcKey1M ; No, check M flag
  cbr rFlag,1<<bSetS ; Clear S flag
  ldi rmp,60 ; Multiply by 60
  rcall Multiply
  mov rMinInp,ZH ; Set minutes
  mov rSecInp,ZH ; and set seconds
  rjmp AdcRestartBounce
AdcKey1M:
  sbrs rFlag,bSetM ; M flag set?
```

```
  rjmp AdcSetO ; No, set O flag, leave input mode
  cbr rFlag,1<<bSetM ; Clear M flag
  ldi rmp,60 ; Multiply by 60
  rcall Multiply
  mov rMinInp,ZH ; Set minutes input
  rjmp AdcSetHour ; and set hour
AdcKey2:
  sbic pKeyI,bKey2 ; Key 2 pressed?
  rjmp AdcTimeOut ; No, check input change
  sbrs rFlag,bSetS ; S flag set?
  rjmp AdcKey2M ; No, check M flag
  ldi rmp,60 ; Multiply by 60
  rcall Multiply
  ldi rmp,cSecDiv ; Restart second divider for a full second
  mov rSecDiv,rmp ; Seconds divider to restart = restart second
  cbr rFlag,1<<bSec ; Clear second flag if currently set
  mov rSec,ZH ; Set seconds
  mov rMin,rMinInp ; Copy input minutes to time minutes
  mov rHr,rHrInp ; Copy input hours to time hours
  rjmp AdcSetO ; Set O flag to leave input mode
AdcKey2M:
  sbrs rFlag,bSetM ; M flag set?
  rjmp AdcKey2H ; No, check hour flag
  sbr rFlag,1<<bSetS ; Set S flag
  ldi rmp,60 ; Multiply ADC MSB by 60
  rcall Multiply
  mov rMinInp,ZH ; Set minute
  mov rSecInp,ZH ; Set second
  rjmp AdcRestartBounce ; Restart bounce counter
AdcKey2H:
  sbr rFlag,1<<bSetM ; Set minute flag
  cbr rFlag,1<<bSetS ; Clear second flag
  ldi rmp,60 ; Multiply ADC MSB by 60
  rcall Multiply
  mov rMinInp,ZH ; Set minute
  rjmp AdcSetHour ; and hour
AdcTimeOut:
  inc rInpToL ; Check time-out, LSB
  brne AdcNoKey ; Not zero
  inc rInpToH ; dto., inc MSB
  brne AdcNoKey ; Not zero
  ; Time-out input period
  cbr rFlag,1<<bSetA ; Clear TSet flag
  rjmp DisplayTimeInp ; Display time
AdcNoKey:
  sbrc rFlag,bNoUpd ; No uddate flag set?
  rjmp AdcRdyRet ; Yes, do not update
  ldi XH,High(sInput) ; Point X to hour input, MSB
  ldi XL,Low(sInput) ; dto., LSB
  ldi rmp,24 ; Convert MSB ADC to hours
  rcall Multiply
  sbrs rFlag,bSetM ; M flag set?
  rjmp AdcNoKey1 ; No, check change
  adiw XL,1 ; Next position
  ldi rmp,60 ; Convert MSB ADC to minutes/seconds
  rcall Multiply
  sbrc rFlag,bSetS ; S flag clear?
  adiw XL,1 ; No, increase address
AdcNoKey1:
  ld rmp,X ; Read current value at position
  cp ZH,rmp ; Compare with new value
```

```
  breq AdcRdyRet ; Not changed, do nothing
  st X,ZH ; Store changed value
  rjmp DisplayInput
AdcSetHour:
  ldi rmp,24 ; Multiply MSB ADC by 24
  rcall Multiply
  mov rHrInp,ZH
  rjmp AdcRestartBounce
AdcSetO:
  sbr rFlag,1<<bSetO ; Set O flag
AdcRestartBounce:
  ldi rmp,High(cInpTO)
  mov rInpToH,rmp
  ldi rmp,Low(cInpTO)
  mov rInpToL,rmp
  ldi rBounce,cBounce
AdcRdyRet:
.if dbg_moni_flags == Yes
  mov rSecInp,rFlag ; Copy rFlag to seconds input
  rcall DisplayTimeInp ; and display
  .endif
.if dbg_moni_bounce == Yes
  mov rSecInp,rBounce ; Copy bounce to seconds input
  rcall DisplayTimeInp
  .endif
  ret
;
; Multiply MSB of last ADC result by rmp
;   MSB Result in ZH
;   rmp is multiplicator
Multiply:
  mov R0,rAdcRes ; Copy MSB ADC result to R0
  clr R1 ; Clear MSB multiplier
  clr ZL ; Clear multiplication result in Z, LSB
  clr ZH ; dto., MSB
Multiply1:
  tst rmp ; Ready multiplying?
  breq Multiply3 ; Yes
  lsr rmp ; Next bit
  brcc Multiply2 ; Not one, skip adding
  add ZL,R0 ; Add multiplicator, LSB
  adc ZH,R1 ; dto., MSB
 Multiply2:
  lsl R0 ; Multiplicator * 2, LSB
  rol R1 ; dto., MSB
  rjmp Multiply1 ; Continue multiplication
Multiply3:
  ret
;
; Blink active input leds off
Blink:
  cbr rFlag,1<<bBlink ; Clear flag
  sbrs rFlag,bSetS ; Seconds active?
  rjmp Blink1
  clr rSecInp ; Seconds to zero
  rjmp DisplayTimeInp ; and display
Blink1:
  sbrs rFlag,bSetM ; Minutes active?
  rjmp Blink2
  clr rMinInp ; Minutes to zero
  rjmp DisplayTimeInp
```

```
Blink2:
  clr rHrInp ; Hours to zero
  rjmp DisplayTimeInp
;
; End of source code
;
```

Lob, Tadel, Fehlermeldungen, Genöle und Geschimpfe oder Spam bitte über das Kommentarformular an mich.