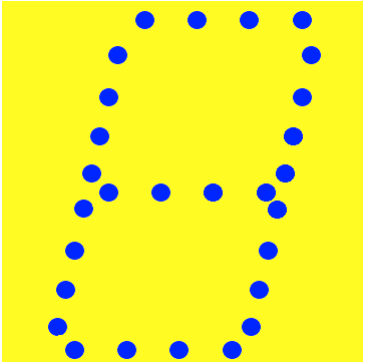


AVR-Anwendungen

Groß-Uhr mit ATmega48
Hardware, Aufbau, Anwendung und Software für die Groß-Uhr

Groß-Uhr mit ATmega48

1 Eigenschaften der Groß-Uhr mit ATmega48

Die Hardware-Eigenschaften der beschriebenen Groß-Uhr:

- Vier Ziffern in Siebensegment-Manier zeigen Stunden- und Minuten- Zehner- und -Einer an,
- Siebensegment-Ziffern aufgebaut aus je vier 10mm-LEDs macht 28 LEDs pro Ziffer, sehr hell,
- zwei LEDs in der Mitte blinken im Sekundentakt,
- Riesig: 680*240 mm groß, auch aus sehr großer Entfernung noch gut ablesbar,
- stromsparende Ansteuerung: durch serielle Ansteuerung der vier LEDs eines Segments nur 25 mA pro Segment oder maximal 200 mA pro Ziffer (wenn alle Segmente eingeschaltet sind), Vierer-Multiplex der Ziffern,
- vielfältige Stellmöglichkeiten der Uhr:
    1. manuell mit zwei Tasten und einem Poti,
    2. automatisch durch Anschluss eines DCF77-Empfangsmoduls,
- Helligkeitssteuerung wahlweise mit Umgebungslichtsensor oder mit Poti,
- quarzgenaue Uhr mit 4,096 MHz-Quarz,
- geringer Stromverbrauch: ca. 7 kWh für ein ganzes Jahr,
- vielfältige Diagnosemöglichkeiten im Quellcode zum Auffinden von Verdrahtungsfehlern der Hardware,
- gut dokumentierter freier Assembler-Quellcode, leicht modifizierbar für andere Konfigurationen,
- freie Dokumentation mit Quellcode und allen Rechen- und Grafikdokumenten
- einfach in der fertigen Schaltung umprogrammierbar mit der eingebauten ISP6-Schnittstelle, daher schnelle Änderungen der Konfiguration möglich.

Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

2 Hardware

2.0 Vorbemerkungen

Bei der Entwicklung der Hardware dieser Uhr habe ich folgende Erfahrungen gesammelt, die vielleicht für andere hilfreich sein könnten:

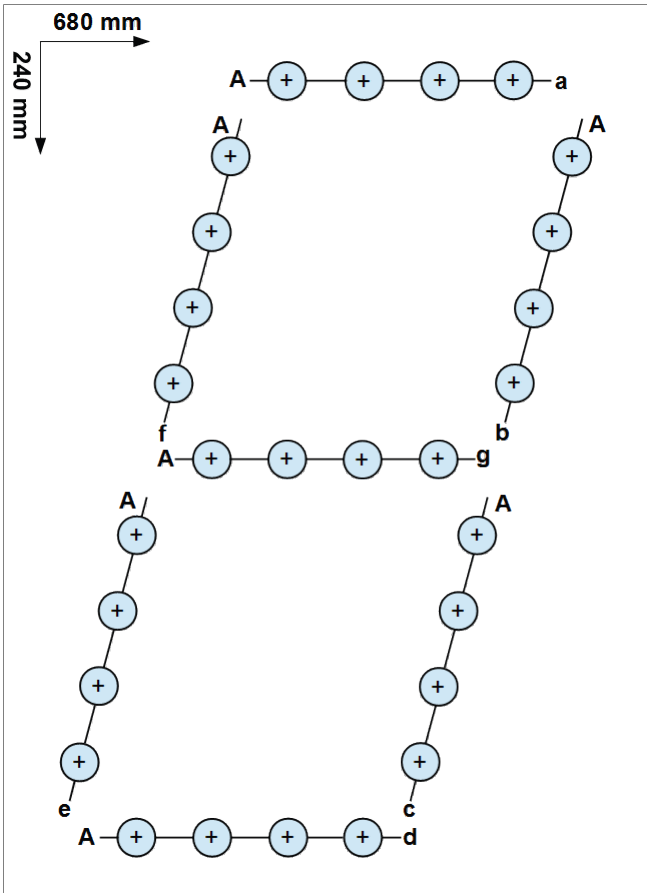
1. Mein erster Versuch, den Prozessor direkt mit einem Uhrenquarz mit 32,768 kHz zu takten, erwies sich als nicht realisierbar. Nicht dass das nicht gehen täte. Man muss dazu natürlich die Fuse auf einen externen Quarz und nicht auf einen externen Quarzoszillator einstellen (Fehler #1). Das Studio 4.19 hat diese fehlerhafte Umstellung übrigens gar nicht bemerkt und alles als ok verifiziert. Erst als ich über ISP nochmals auf die Fuses zugreifen wollte, kam die Fehlermeldung. Als der Fehler dann mit Hilfe der Hochvolt-Parallelprogrammierung im STK500 behoben war, ging es aber trotzdem nicht. Der Chip war über HV sehr wohl, aber über ISP in der Uhrenschaltung gar nicht ansprechbar. Nur wenn ich die Betriebsspannung in der Zielplatine der Uhr auf 5 V erhöhte (natürlich ohne angeschlossenes Display), lief die Blinkroutine ordnungsgemäß ab. Ursache war aber nicht, dass der Prozessor nicht mit 32,768-kHz-Quarzen klar kommt, sondern dass der Brown-Out-Detektor in der Einstellung 2,7 V nicht wirklich bei 2,7 V auslöste sondern noch weit darüber bei 4 V anspricht und den Chip dauerhaft rücksetzte. Erst nach dem Abschalten der Brown-Out-Detektion lief alles stabil (Fehler #2). Aber noch nicht genug damit: das Studio 4.19 weigert sich beharrlich, mit 4 kHz ISP-Frequenz zu programmieren. Es behauptet, das sei unter 5 kHz nicht möglich, obwohl das Handbuch des ATmega48 keinerlei Hinweis auf eine solche Beschränkung enthält und das Original-STK500-Protokoll noch viel niedrigere Programmiergeschwindigkeiten ohne Probleme beherrscht (Fehler #3). Es muss sich daher um einen Fehler beim Implementieren der Flash-Programmierung im Studio handeln. Da bei Prototypen die Entwicklung etwas komplizierterer Programme ohne ISP-Programmierung eine Quälerei ist, habe ich das Vorhaben mit 32,768 kHz Taktfrequenz schließlich schweren Herzens aufgegeben. Obwohl ich schon gerne wüsste, ob ich das alles zeitlich korrekt hingekriegt hätte (ich vermute: ja - 32.768 Instruktionen in der Sekunde sind in Assembler halt doch noch viel Holz).
2. Da ich geplant hatte, auf die 16 Dioden zu verzichten, die acht Konstantstromregler mit Transistoren normalerweise brauchen, habe ich die Betriebsspannung des Prozessors so weit herabgesetzt, dass die Dioden nicht gebraucht werden und die Regeltransistoren mit der Basis direkt auf Portspannungsniveau hängen. Nach meiner Rechnung war dafür 3,3 V Betriebsspannung ideal, um mit einem Emitterwiderstand von 100Ω auf 27 mA Konstantstrom zu kommen. Leider ergab schon der erste Versuch, dass der Spannungsregler BA033FB stolze 4,04 V abgab. Mit den sich daraus ergebenden (4,04 - 0,6) / 100 * 1000 = 34,6 mA wären die Leds einen schnellen Tod gestorben (Fehler #4) und der Trafo wäre mit 277 mA in den Kurzschlussmodus übergegangen. Es blieb nichts anderes über als ein ordentliches 3,3 V-Netzteil zu Fuß und ohne fertigen (falschen Regler) selbst zu bauen.
3. Der vom Versandhändler gelieferte 2,8VA-Trafo erwies sich als viel zu schwach und hat nach Gleichrichtung bei Belastung mit 82Ω (198 mA) nicht mehr seine Nominalspannung abgeliefert (Fehler #5). Er hatte auch nicht 12 V nominal sondern 15 V (Fehler #6). Und das erklärt, warum er bei 82Ω Last schon zusammenbrach (2,8VA / 15V = 187mA). Die höhere Spannung brachte den Ladeelko fast zum Aufblähen seiner Backen (zu hohe Überspannung bei Nulllast, war nur durch den Einbau einer Betriebsspannungs-Led zu beheben) sondern auch das 27mA-Konzept zum Scheitern, denn bei allen acht Segmenten angeschaltet wären mit 216 mA der Trafo und das Netzteil überfordert gewesen. Es blieb daher nur, auf die minimalen 2,7V Betriebsspannung des ATmega48 und auf ca. 19 mA Konstantstrom zu gehen.

Der Helligkeitsunterschied erwies sich allerdings als kaum merklich, deshalb war der Kompromiss noch erträglich.

4. Normalerweise reichen beim Multiplexen von vier Ziffern 50 bis 60 Hz vollkommen aus und man erkennt kein Flimmern mehr. Bei Versuchen mit dem großen Display stellte sich aber heraus, dass 60 Hz doch noch unangenehm flimmern (Fehler #7). Das Konzept musste daher auf 100 Hz und mehr umgestellt werden.

Ich finde sieben Fehler einfach zu viel, von denen sechs von mir nicht zu vertreten waren. So macht Basteln keinen Spaß mehr.

2.1 Anzeigen-Teil



Dies ist eine der vier Ziffern der Anzeige der Uhr. Sie besteht aus 28 10mm-Leds, jeweils pro Segment vier hintereinandergeschaltet. Alle Anoden einer Ziffer (A) werden verbunden und als A1..A4 über Flachkabel an eine 16-polige Buchse geführt.

Die Kathoden a bis g werden über alle vier Ziffern zusammen verbunden und ebenfalls an die 16-polige Buchse geführt.

Die zwei blinkenden LEDs in der Mitte bilden das achte Segment. Dessen Anode kann an eine der vier Ziffern angeschlossen werden. Welche das ist muss per Software eingestellt werden.

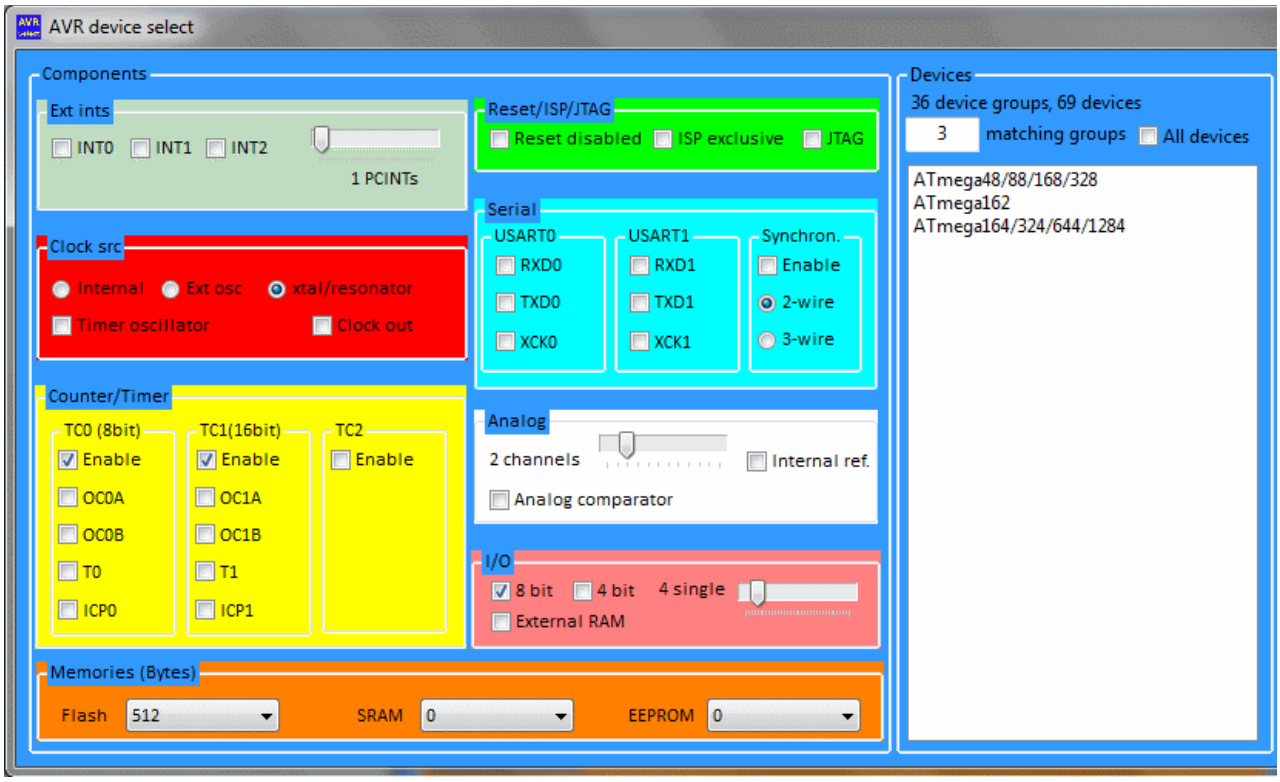
Die Belegung der 16-poligen Buchse ist im Schaltbild der Prozessor- und Treibereinheit dokumentiert.

2.2 Prozessorteil

2.2.1 Auswahl des AVR-Typs

Der AVR-Typ ergibt sich aus folgenden Hardware-Anforderungen:

1. Da die Uhr auch ohne DCF77-Synchronisation über längere Zeit ziemlich genau laufen soll, muss entweder der Prozessor mit einem externen Quarz getaktet werden oder der Timer/Counter TC2 muss mit einem externen Uhrenquarz laufen.
2. Damit auf eine dauernde Abfrage des DCF77-Eingangs und der beiden Taster-Eingänge verzichtet werden kann, sollten Flanken an diesen drei Eingängen mit einem INTn oder PCINTn zu Interrupts führen.
3. Die acht Konstantstrom-Schalter sollten in einem Port, vier weitere Anodenschalter in einem weiteren Port liegen.
4. Die Messung des Potentiometer- und des Lichtsensor-Eingangs muss in zwei Analog-Eingängen erfolgen.
5. Für das Multiplexen der Anzeige ist ein 8-Bit-Timer, für die Messung der DCF77-Signaldauer ein 16-Bit-Timer vonnöten.



Durch die Software hier engt sich die Auswahl auf einige wenige AVR-Gruppen ein. Der ATmega48 erfüllt alle genannten Anforderungen. Wer den gerade nicht vorrätig hat, kann auch einen ATmega88 nehmen, der nur ein wenig mehr unnützes Flash-Memory mit sich herumträgt.

2.2.2 Auswahl der Taktfrequenz

Den Takt erhält der ATmega48 aus einem angeschlossenen Quarz mit 4,096 MHz. Die beiden Keramikkondensatoren von 22 pF helfen beim Anschwingen des Quarzoszillators. Die Quarzoszillator-Fuse des ATmega48 ist vom internen RC-Oszillator auf den externen Quarz umzustellen (siehe Fuse-Umstellung), weil sonst die Uhr um den Faktor 1,024 zu langsam geht (35 Sekunden pro Tag).

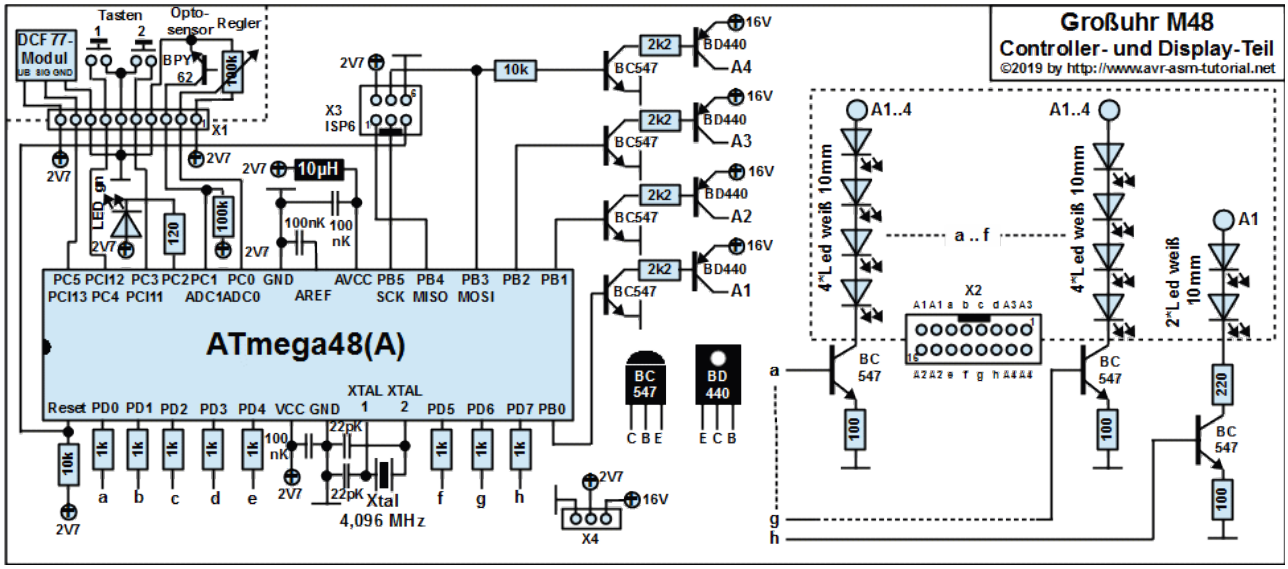
Die Gründe für die Auswahl dieses Quarzes waren folgende (siehe auch das Tabellenblatt Timer im Open-Office Tabellenwerk hier):

1. Der Quarz sollte auch im Handel erhältlich sein.
2. Die Quarzfrequenz, geteilt durch 8 oder 64 sowie durch 256 muss eine Ganzzahl ohne Rest ergeben.
3. Das Teilen durch 256 ist erforderlich, damit der Multiplextakt durch einen 8-Bit-Timer erfolgen kann, dessen Compare-Register auch das vorzeitige Abschalten der Anodentreiber vornehmen kann (Dimm-Funktion, Wertebereich zum Abschalten 0..255). Wer stattdessen einen Quarz verwendet, der zum Erreichen ganzzahliger Teilerfrequenzen den CTC-Modus erzwingt (z. B. 2,0 oder 4,0 MHz), muss die Abschaltfunktion auf Compare B verlegen und durch Ändern der Multiplikation die Potentiometerwerte zum Dimmen auf diesen Maximalwert im Compare-A-Port begrenzen.
4. Wählt man als Prescaler 8, dann werden bei der niedrigsten Quarzfrequenz von 2,048 MHz viel zu hohe Multplexfrequenzen erhalten (1000 Hz), was zu unnötigem Stromverbrauch und zu HF-Feldern fürt. Wählt man aber 64, ist die MUX-Frequenz mit 31,25 Hz aber etwas zu niedrig. Es empfehlen sich also Quarze ab 3,2768 MHz aufwärts (MUX-Frequenz 50 Hz und höher).

4,096 MHz erfüllt die Anforderungen, die MUX-Frequenz liegt mit 125 Hz in einem optimalen Bereich.

2.2.4 Schaltbild der Uhr

Dies ist der Prozessorteil mit dem ATmega48 und allen seinen angeschlossenen Komponenten.



2.2.5 Kathodentreiber als Konstantstromquellen

Der Port D des Prozessors steuert die Kathoden der Anzeige. Die Ausgänge des Ports werden mit 1k-Widerständen an die Basen von NPN-Transistoren BC547 geführt, deren Emitterwiderstände von 100Ω die Segmentströme auf ca. 21 mA begrenzen:

$$I_E = (2{,}7V - 0{,}6V) / 100 * 1000 = 21\ mA$$

Die Portausgänge sind aktiv high.

Das ist auch der Grund, warum die Betriebsspannung des AVR bei 2,7 V liegt: bei 5 V wären am Emitterwiderstand schon 4,4 V verbraten worden, was zusammen mit den 4 * 3,18 V = 12,72 V für die vier LEDs und den ca. 0,2 V U_{CE}(sat) des BD440 17,3 V ergäben hätte. Die Konsequenz wäre gewesen, dass ein 12V-Trafo nicht ausreichend gewesen wäre und stattdessen ein 15V-Typ erforderlich gewesen wäre, den es aber in der zweiwickligen Ausführung nicht in der passenden Größe zu kaufen gibt.

Eine Alternative wäre gewesen, die 5 V mit zwei Dioden auf 1,4 V herabzusetzen. Da dazu aber 16 dieser Dioden zu verbauen gewesen wären, wurden 2,7 V Betriebsspannung gewählt.

Ursprünglich hatte ich den Plan, die Uhr mit einem 32kHz-Uhrenquarz zu betreiben. Diesen Plan habe ich wegen unüberwindlichen Schwierigkeiten aufgegeben.

2.2.6 Anodentreiber

Die vier Anoden werden vom unteren Teil von Port B angesteuert. Die unteren drei Bits sind direkt an die Basen von BC547-NPN-Transistoren gefürt, die Durchsteuerung erfolgt mittels den per Software eingeschalteten Pull-Up-Widerständen. Der Strom aus den Pull-Up-Widerständen von ca. 47k reicht aus, um die Transistoren mit einem hFE_{min} von 150 zu treiben:

$$I_C = (2{,}7V - 0{,}7V) / 47k\Omega * hFE_{min} * 1000 = 6{,}4\ mA$$

Die eingeschalteten NPN-Transistoren schalten über 2k2-Widerstände die Basis der Ano-

dentreibertransistoren BD440 an. Die bei 16V Betriebsspannung fließenden 6,4 mA Basisstrom reichen bei weitem bin, um die BD440-PNPs mit einem hFE schon von ca. 30 mit einem Kollektorstrom von maximal 200 mA voll durchzusteuern.

Da das vierte Anodenbit beim Betrieb der ISP6-Schnittstelle mit dem MOSI-Signal getaktet wird, ist der NPN-Transistor in diesem Fall mit einem 10k-Widerstand angeschlossen und der Portausgang ist als Ausgang konfiguriert.

2.2.7 ISP6-Schnittstelle

Die Portpins PB3 bis PB5 bedienen die ISP6-Schnittstelle, über die der Flashspeicher in der Schaltung mit dem Programmcode gefüttert werden kann. Ist beim Programmieren kein Netzteil angeschlossen, muss das Programmiergerät 3,3 V liefern und entsprechend konfiguriert werden. Sind die LEDs beim Programmieren nicht angeschlossen, können auch 5 V zum Programmieren verwendet werden. Würden die LEDs bei 5 V angeschlossen, läge ihr Strom bei 44 mA, was sie zwar kurzzeitig vertragen, aber nicht für länger. Allerdings würde sich der Strom bei allen acht Kathoden auf 350 mA belaufen, was das Netzteil nicht liefern kann.

2.2.8 Sonstige Peripherie

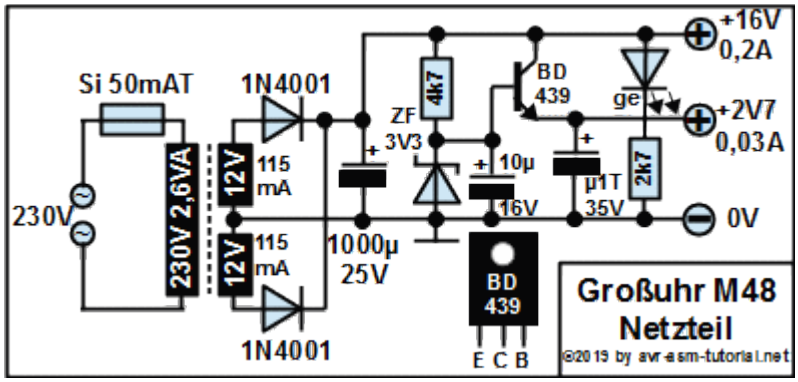
An Port C ist die restliche Peripherie angeschlossen:

1. An PC0 (ADC0) ist das Potentiometer angeschlossen und wird im Wechsel mit dem Lichtsensor laufend in einen Binärwert umgewandelt.
2. An PC1 (ADC1) ist ein Fototransistor zur Messung der Umgebungshelligkeit angeschlossen.
3. An PC2 ist eine kleine grüne LED angeschlossen. Sie kann als Betriebsspannungs-LED oder im Hardware-Debug-Modus zum Anzeigen von Ereignissen dienen (siehe Hardware-Debugging)
4. An PC3 und PC4 bzw. den folgenden drei Pins der 10-poligen Steckverbindung sind die beiden Tasten angeschlossen, die durch die Pull-Ups von PC3 und PC4 auf Eins liegen und beim Schließen auf Null gezogen werden (aktiv low).
5. An PC5 wird das DCF77-Modul angeschlossen. Es ist egal, ob dessen Ausgang aktiv low oder high ist. PC5 hat den Pull-Up ebenfalls gesetzt, so dass bei nicht angeschlossenem Modul keine Störsignale den Programmablauf stören. Das Modul sollte bei 2,7 V Betriebsspannung schon arbeiten.

2.3 Netzteil

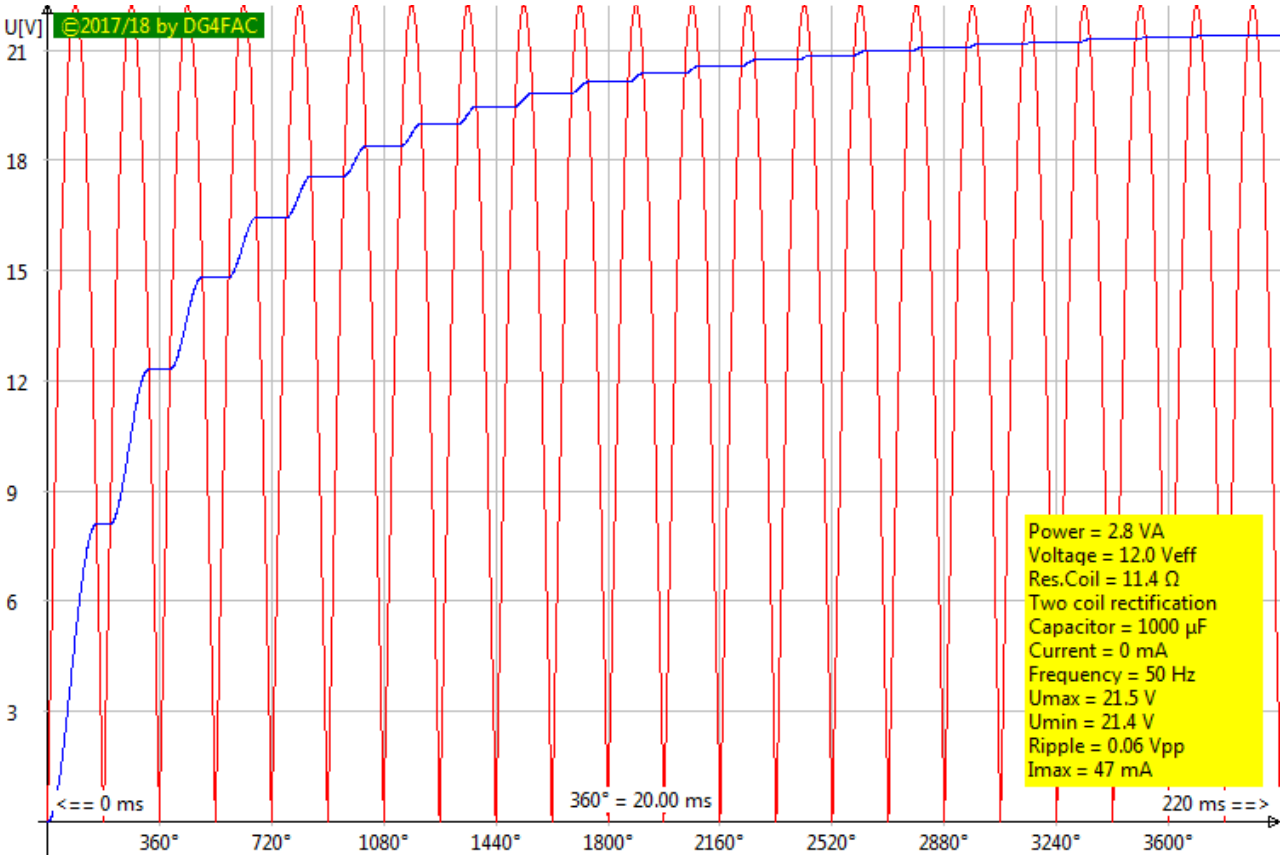
Das Netzteil muss folgende Eigenschaften besitzen:

1. Es muss die mindestens 16 V 0,2A liefern, die für den Betrieb der LEDs erforderlich sind.
2. Es muss ferner die 2,7 V für den Betrieb des Prozessors liefern. Der Regler muss bis zu 25 V vertragen, die das Netzteil ohne angeschlossene Last im Leerlauf produzieren kann.



Im Schaltbild ist so ein Standard-Netzteil mit einem 2*12 V-Trafo mit seiner Dimensionierung zu sehen. Die beiden Dioden sind 1N4001.

Das hier simuliert das dargestellte Netzteil ohne angeschlossene Last (siehe die Power-Supply-Software hier). Die Spannung am Elko bleibt ohne Last in der Theorie bei deutlich unter 25 V, deshalb kommen die Elkos mit einer Spannungsbelastbarkeit von 25 V aus. In der Praxis stellte sich allerdings heraus, dass der Trafo den angegebenen Spannungsfaktor bei weitem nicht einhält und fast doppelt so hohe Spulenwiderstände hatte. Im Leerlauf

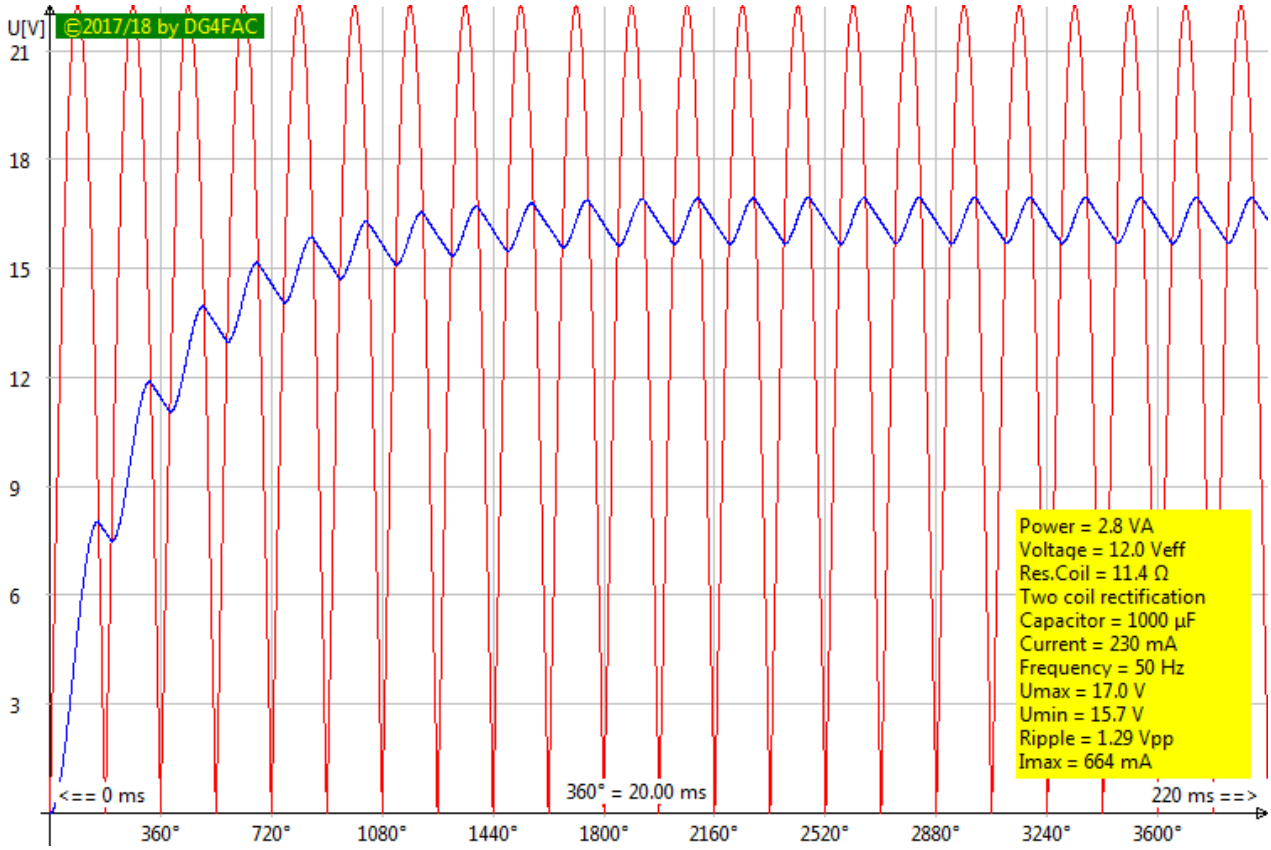


lieferte er daher eine viel höhere Spannung als berechnet. Um den Elko nicht zu gefährden, war es notwendig, die Leerlaufspannung mit der Kombination aus gelber 3mm-LED und 2k7 herabzusetzen, um unter die 25V zu kommen. Verlasse Dich also nie auf Herstellerangaben, in der Regel ist die Realität schlechter.

Der viel höhere Leerlaufspannungsfaktor hatte außerdem zur Folge, dass die Spannung bei einer Last von 200 mA gerade mal nur noch 12,4 V betrug. Da die 200 mA nur auftreten, wenn alle 8 Segmente angeschaltet sind (und das wegen des Multiplexens der Anzeige auch nur für 16 ms lang, also maximal für einen Netzspannungsdurchgang lang), ergeben sich keine nachteiligen Konsequenzen für die Spannung der Anzeige. Was alles so unter dem Siegel "2*12V 2,8 VA" verkauft wird ...

Normalerweise nimmt man für die Regelung der niedrigen Spannung gerne einen integrierten Spannungsregler. Das ergab aber katastrophale Ergebnisse: der integrierte 3,3V-Regler lieferte stolze 4,04V Ausgangsspannung, und das bei immerhin schon 30mA Last. Da das die Anzeigenlampen zerstört hätte, musste eine Lösung mit niedrigerer Spannung her, also 2,7 V. Das ist die Mindestspannung, die der ATmega48 zum Funktionieren seiner Hardware erwartet. Und für 2,7 V gibt es keine integrierten Spannungsregler, die diese Mindestspannung auch garantieren (der 78L02 geht gerne auch mal mit weniger als 2,7V zu Werke, z. B. mit 2,45V).

Die Lösung war, den 2,7 V-Spannungsregler über eine Zenerdiode mit einem NPN-Transistor selbst zu bauen. Damit entgeht man allem Ärger mit den leidigen Bauteiltoleranzen.



Das hier ist die berechnete Netzteilspannung bei 230 mA Last, der Maximallast während der höchsten Multiplexphase und mit dem höchsten Prozessorverbrauch. Die Spannung sinkt leicht unter 16 V, aber nur geringfügig, so dass der Betrieb der LEDs nicht gestört wird. Es besteht ausreichend Lastreserve und der Elko ist mit einem Ripple von 1,3 V, das von der Konstantstromregelung der LEDs abgefangen wird, großzügig dimensioniert.

In der Praxis wies der Trafo aber einen viel höheren Leerlauf-Spannungsfaktor auf und lieferte wegen der höheren Spulenverluste auch weniger Strom als die aufgedruckten 117 mA pro Spule. Die Konsequenz war die Herabsetzung des Lampenstroms durch Senkung der Betriebsspannung des ATmega48 und der Einbau der gelben LED, denn im Leerlauf erreichte der Trafo 27 V am Elko.

Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

3 Aufbau

3.1 LED-Frontplatte

Die 28 10mm-LEDs pro angezeigter Ziffer sind mit einer Breite von 170 mm auf einer insgesamt 680*240 mm großen Plexiglasscheibe angebracht. Jeweils vier LEDs sind zu einem Segment verdrahtet (siehe Bild oben). Jede Ziffer hat sieben Segmente.

Für das Bohren der 112 Löcher werden die vier DINA4-Ausdrucke der Ziffern (siehe das Open-Office-Draw-Dokument mit den Zeichnungen hier) auf die Plexglasplatte aufgeklebt und mit einem 1,5mm-Bohrer vorgebohrt. Das Aufbohren mit immer größeren Bohrern erfolgt behutsam, damit die Platte nicht ausreißt, und mit niedriger Geschwindigkeit, damit das Plexiglas beim Bohren nicht schmilzt.

Ab 6mm wird mit einem schnelldrehenden Entgrater von vorne und von hinten auf etwa

9mm Durchmesser weitergebohrt. Bei der Verwendung einer Ständerbohrmaschine kommt danach der 10mm-Bohrer mit hoher Drehgeschwindigkeit und sehr langsamem Vorschub zum Einsatz. Das führt zwar zum Schmelzen des Plexiglases an der Bohrstelle, verhindert aber ein Ausreißen der Bohrung.

Für diejenigen Löcher, die mit der Ständerbohrmaschine nicht zugänglich sind, weil der Ständer im Weg steht, wird mit 7, 8 und 9mm langsam gebohrt und dann mit 10mm das Endloch gebohrt. Man kann auch mit einem scharfen Messer die Bohrung auf 10mm ausweiten, das ist am schonendsten und führt nicht zum Ausreißen.

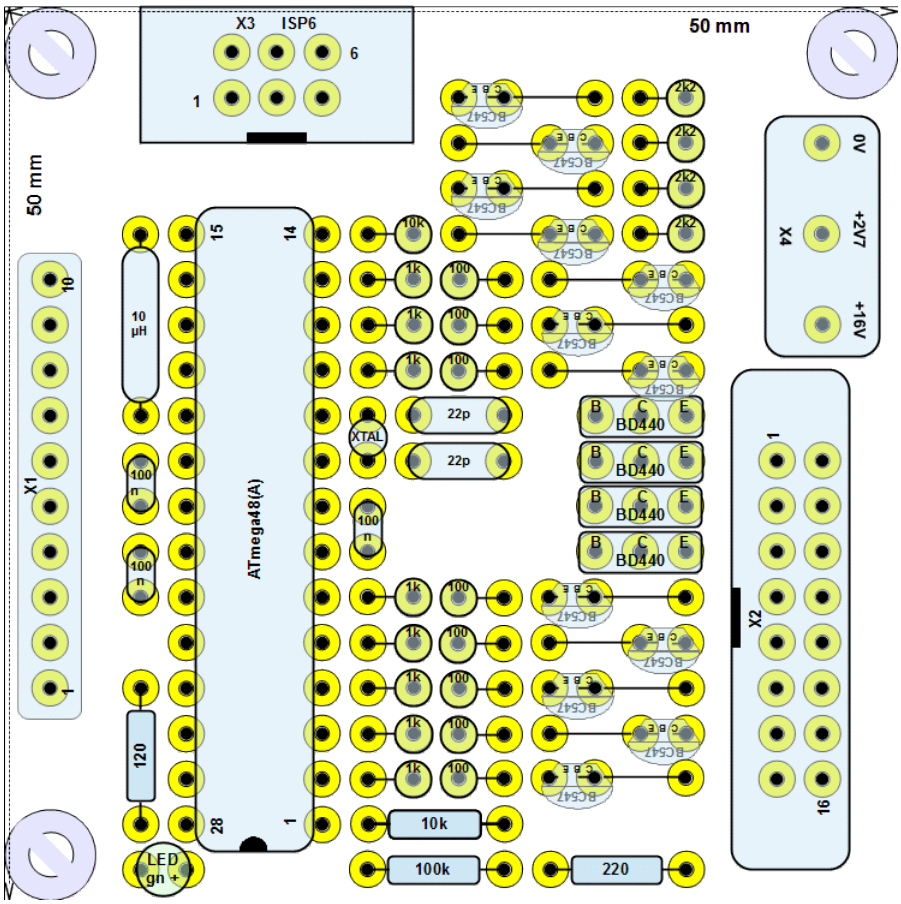
Auf den Seiten 2 und 3 des Ausdrucks sind auch noch die beiden Mittel-LEDs eingezeichnet.

Zuletzt werden in der Mitte noch die beiden 10mm-Löcher für die Mitten-LEDs gebohrt. Deren Anode kann mit irgendeiner der vier anderen Anoden verdrahtet werden, ihre Platzierung ist im Assembler-Quellcode anzugeben.

Zum Schutz und zum Aufstellen der Uhr wird in vier bis fünf cm Abstand eine zweite Plexiglasscheibe der gleichen Größe mit sechs Abstandshaltern montiert.

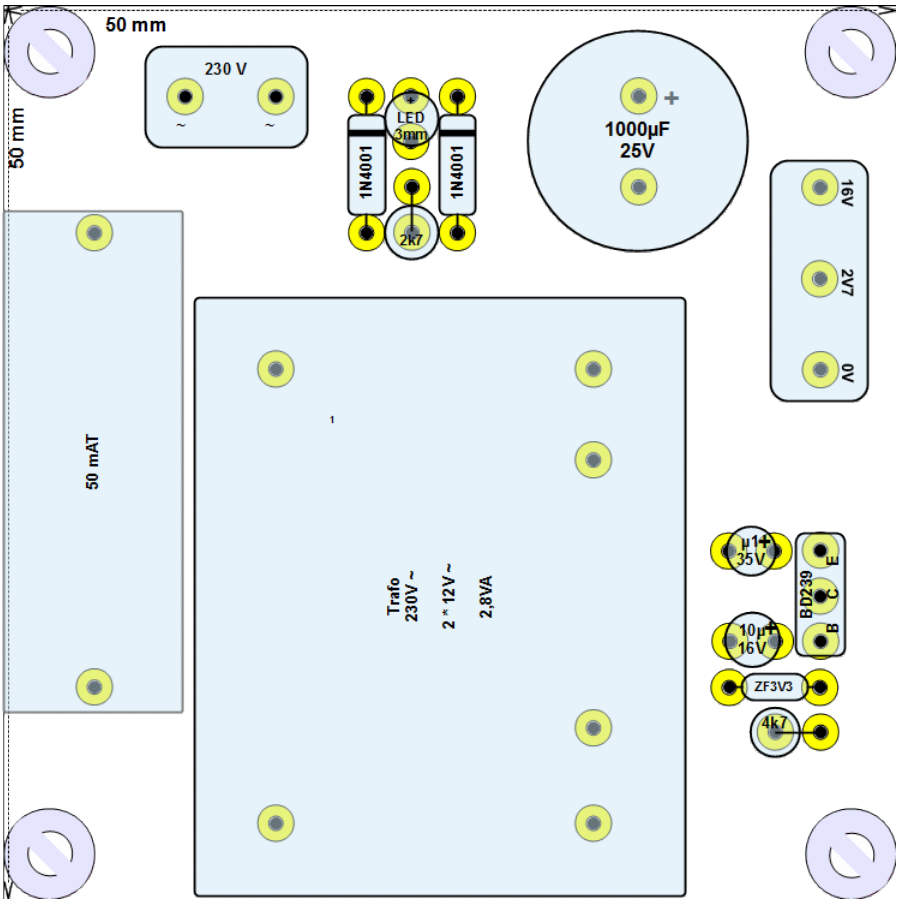
3.2 Prozessor-Platine

Das sind die Komponenten mit dem Prozessor auf einer 50x50mm großen Lochrasterplatine. Alle äußeren Anschlüsse sind mit Steckverbindern versehen, damit man die Platine auch nach dem Einbau und der Fertigstellung der Uhr noch ganz herausnehmen kann.



3.3 Netzteil-Platine

Das Netzteil wird auf einer ebenfalls 50x50 mm großen Lochrasterplatine aufgebaut.



Seitenanfang	Eigenschaften	Hardware	Aufbau	Software

4 Software

4.1 Downloads

Der Quellcode in Assembler ist hier verfügbar und kann im Anhang hier als Text angesehen werden.

Die folgenden Dokumente sind zusätzlich im Open-Office-Format verfügbar:

1. ein Tabellendokument mit allen Berechnungen (Timer, Ströme, Messwerte der LEDs, u.v.a.m. gibt es hier,
2. die Schaltbilder des Prozessors und des des Netzteils gibt es hier,
3. die Layouts der Prozessor- und der Netzteilplatine gibt es hier (nein, ich habe keine gedruckte Platine entwickelt),
4. die vier Seiten des Siebensegment-Layouts gibt es hier.

4.2 Assemblieren des Quellcodes

Den Quellcode gibt es in Assemblerformat hier zum Download und ist in diesem Anhang dokumentiert. 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 herunterzu-

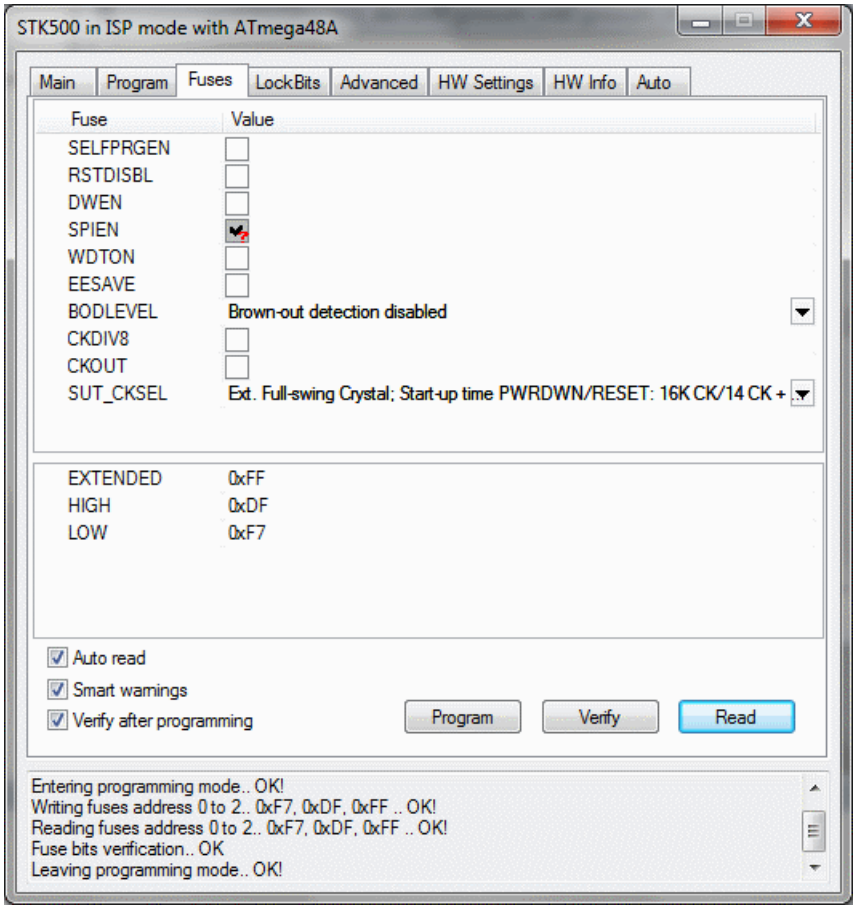
laden und zu installieren oder wer kein Windows-Betriebssystem hat, ist mit meinem Assembler gavrasm besser und einfacher bedient. Wie damit mit einem 64-Bit-Betriebssystem assembliert wird, findet sich für Windows hier und für Linux hier. Wer ein ganz anderes Betriebssystem (32-Bit, Mac-OS, etc.) hat, 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 herumliegen.

4.3 Flashen, Fuses

Der Hexcode muss dann in den Flashspeicher des ATmega48 geschrieben werden. Das geht mit einem AVR-Brenner und der entsprechenden Software.

Vor dem ersten Programmieren sollten die Fuses des ATmega48 auf den externen Quarz umgestellt werden. Bitte dabei beachten, auch die CLKDIV8-Fuse zu löschen, die CKOUT-Option abzuschalten und die Brown-out detection entweder auf 1,7V einzustellen oder abzuschalten. In der Praxis erwies sich die Brown-Out-Detection bei 1,7V als wesentlich sensibler als eingestellt. So blockierte der Brownout schon, wenn die Betriebsspannung mal unter 2,5 V lag. Man schaltet die daher besser ganz aus und vermeidet so diesen Ärger.



Ob der Prozessor korrekt läuft, kann mit der Debug-Funktion überprüft werden, die die grüne LED blinken lässt (siehe hier).

4.4 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. "Debug_ledgreen": Beim Einschalten dieser Option sollte die grüne LED, die an PC2 angeschlossen ist, hektisch blinken. Wenn die LED gar nichts tut, läuft der Prozessor nicht und man muss im Clock-Bereich auf Fehlersuche gehen. Blinkt die LED sehr langsam, ist die CLKDIV8-Fuse noch gesetzt und der Prozessor arbeitet mit 512 kHz.
2. "Debug_current": Alle acht Kathodentreiber werden aktiviert, mit einem Amperemeter kann der Konstanstrom an den Kathodenpins des Steckverbinders X2 nach

+16V gemessen werden. Er sollte um die 27 mA betragen. Ist die LED-Anzeige angeschlossen, gehen alle sieben Segmente der Zehner-Stunden an.

3. "Debug_segments": Das schaltet nacheinander die sieben Segmente a bis h jedes Segments ein und geht dann zur nächsten Ziffer. Nach der vierten beginnt die Ausgabe wieder von vorne. Die Geschwindigkeit kann mit der Variablen "cDebug_seg-Delay" herabgesetzt werden. Mit 1 geht es sehr flott, mit 10 schon etwas gemütlicher zu.
4. "Debug_mux8" Diese Variante stellt in einer simulierten Multiplex-Ausgabe alle Ziffern-Leds (und den Doppelpunkt) an und gibt sie nacheinander auf den vier Ziffern aus. Die Multiplexfrequenz kann mit der Konstanten "cDebug_muxfreq" vorgewählt werden. Bei der Minimaleinstellung mit 4 Hz sieht man die einzelnen Ziffern noch einzeln leuchten. Bei Frequenzen von 20 bis 50 Hz ist ein deutliches Flimmern erkennbar. Bei 62,5 Hz ist das Flimmern fast nicht mehr zu merken. Oberhalb von 100 Hz ist garantiert nichts mehr zu bemerken.
5. "Debug_adc": Die Messergebnisse des ADC, des MSB der ADC-Summe, werden in dezimalem Format auf der Anzeige ausgegeben.
6. "Debug_keys": Die Eingangsignale an den beiden Tasteneingängen werden in den Zehnerstunden (Taste 1) und Einerstunden (Taste 2) angezeigt. Der Eingang wird als kleines i (Segment c der Anzeige) für High oder als kleines o für Low angezeigt.
7. "Debug_dcferr": Alle Fehlerereignisse am DCF77-Eingang werden mit einem E-Fehlercode ausgegeben. Dabei bedeuten:
    0. oder leer: Kein Fehler
    1. Signal kürzer als für eine 0 erforderlich
    2. Signal kürzer als für eine 1 erforderlich
    3. Signal kürzer als für eine Pause erforderlich
    4. Signal kürzer als für ein fehlendes Sekundensignal erforderlich
    5. Signal länger als für ein fehlendes Sekundensignal erforderlich
    6. Beim Minutenwechsel waren keine 59 Sekundensignale empfangen
    7. Minuten-Parität ungerade
    8. Stunden-Parität ungerade
    9. Kein Signal am DCF77-Eingang.

    Korrekte Signaldauern überschreiben den Fehlercode mit einem Leerzeichen, so dass nur die Fehlernummern zu sehen sind.
8. "Debug_dcfany": Alle ausgewerteten Signaldauern werden von hinten nach vorne auf dem Display angezeigt: empfangene Nullen werden als kleines o angezeigt, empfangene Einsen mit einem kleinen i (c-Segment ist an), Pausen mit einem großen P.
9. "Debug_dcfdur": Alle Signaldauern werden in hexadezimalem Format auf dem Display angegeben (die Hexadezimalziffern A, b, C, d, E, F in diesem Format).

4.5 Einstellmöglichkeiten der Software

Im Abschnitt "Adjustable constants" des Quellcodes können diverse Einstellungen im Quellcode vorgenommen werden, die die Eigenschaften der Uhr verändern. Alle in diesem Abschnitt vorhandenen Einstellungen können ohne Weiteres geändert werden.

1. **.equ xtal = 4096000 ; in Hz**
   Wenn man einen anderen Quarz anschließen möchte als den 4,096 MHz: hier kann

man dessen Frequenz angeben. Die Frequenz des Quarzes muss durch 4096 oder 8192 ohne Rest teilbar sein, sonst würde die Uhr ungenau. Im Open-Office-Tabellenkalkulationsblatt **Timer** im Dokument Berechnungen können entsprechende Berechnungen durchgeführt werden.

2. **.equ cClkpr = 4 ; Either two or four**
   Hiermit kann der Vorteiler verstellt werden, aus dem sich die Taktfrequenz des Prozessors ergibt. Es können nur 2, 4 oder 8 sinnvoll verwendet werden. Der Effekt kann im Rechenblatt angeschaut werden.
3. **.equ cStartHours = 0x09 ; Start at 20:00 h**
   .equ cStartMinutes = 0x59
   Das stellt die Uhrzeit ein, mit der die Uhr startet, wenn die Betriebsspannung angelegt wird.
4. **.equ cDcfOnly = Yes ; Display/Clear unsynced time**
   Schaltet die Leds aus, solange noch keine DCF77-Synchronisation vorliegt.
5. **.equ tDcf0 = 100 ; 100 ms for a 0**
   .equ tDcf1 = 200 ; 200 ms for a 1
   .equ tDcfP = 850 ; Pause for 0 and 1 to next second
   .equ tDcfM = 1850 ; Pause for 59th second pulse
   .equ tDcfT = 3000 ; Time-out of DCF signal
   Diese Parameter stellen die Dauer von DCF77-Signalen in Millisekunden ein. Falls das DCF77-Modul weit davon abweichende Dauern (über die Toleranz hinaus) liefert, kann man diese hier verstellen.
6. **.equ cDcfTol = 10 ; Tolerance in %**
   Die unvermeidlichen geringfügigen Abweichungen bei der DCF77-Signaldauer lassen sich hier mit einem Toleranzwert in % einstellen. Abweichungen nach unten und oben werden so akzeptiert.
7. **.equ cAnDp = 3 ; Should be between 1 and 4**
   Dies stellt ein, an welche Anodenleitung der Doppelpunkt in der Mitte angeschlossen ist.
8. **.equ tBounce = 50 ; Bouncing time, in ms**
   Dies stellt ein, nach welchem Zeitraum das Torkeln der Tasteneingänge beendet sein muss. Taster mit noch längerer Torkelzeit können hier abgefangen werden.
9. **.equ cDimOpto = No ; Select the source**

   Dieser Parameter wählt aus, ob der Optosensor (Yes) oder das Poti zum Dimmen verwendet werden soll (No).

Änderungen werden erst nach erneutem Assemblieren und nach Übertragung in den Flashspeicher wirksam.

4.6 Funktionsweise der Software

Die folgenden Einzelkapitel stellen die grundlegenden Funktionen der Software dar.

4.6.1 Zeitsteuerung

Die gesamte Zeitsteuerung ist mit dem Timer/Counter TC0 realisiert. Es gelten folgende Zeitbeziehungen:

1. Quarzoszillator mit externem Quarz = 4,096 MHz,
2. Taktvorteiler 4 mit CLKPR, Prozessortakt = 1,024 MHz,
3. TC0-Vorteiler 8 mit Prescaler, TC0-Takt = 128 kHz,
4. Fast-PWM-Modus mit TOP=255, TC0-Overflow-Int = 500 Hz.

Innerhalb des Overflow-Interrupts werden folgende Aufgaben erledigt:

1. Ausgabe der nächstniedrigen Ziffer der Uhr und Einstellen der nächsten Anode nach

links,
2. Abwärtszählen des Halbe-Sekunden-Vorteilers um Eins, falls Null: Setzen der Halbe-Sekunden-Flagge und Abwärtszählen des Minutenvorteilers (von 120 auf Null), falls Null: Setzen der Minuten-Flagge,
3. Falls der Torkelzähler der Tasten nicht Null ist: wenn eine oder beide Tasten gedrückt sind Neustart des Torkelzählers, falls nicht: Abwärtszählen des Torkelzählers.

Ferner wird TC0 dazu benutzt, um beim Erreichen des Compare-Wertes A die Anoden-Leitungen abzuschalten (Dimm-Funktion). Dies erfolgt in der Interrupt-Service-Routine für Compare A.

Da sich der Compare-Wert nach Auswerten der Potentiometer-Stellung bzw. des Opto-Sensors relativ häfig ändert, war Vorsorge dagegen zu treffen, dass der neue Wert kleiner als der aktuelle Zählerstand werden kann, weil in diesem Fall der Compare-Match verfehlt wird und die Anoden bis zum Ende des Zyklus an bleiben. Das wird durch den Betrieb des TC0 im Fast-PWM-Modus verhindert, weil dann der geänderte Wert zwischengespeichert und erst beim nächsten PWM-Zyklus angewendet wird.

4.6.2 AD-Wandler-Steuerung als weitere Taktquelle

Der AD-Wandler arbeitet mit einem Clock-Prescaler von 128 und wandelt entweder die Analogspannung am Potentiometer an ADC0 (Dimmen mit Poti, Zeiteingabe mit Tasten) oder des Fototransistors an ADC1 aus (Dimmen mit Optosensor, keine Zeiteingabe aktiv).

In der Interrupt-Service-Routine des ADC werden jeweils 64 dieser Messwerte aufsummiert. Sind diese komplett, wird das MSB des Ergebnisses kopiert und eine Flagge gesetzt. Je nachdem in welchem Modus die nächsten 64 Wandlungen erfolgen, wird der AD-Multiplexer mit dem entsprechenden Kanal neu eingestellt.

Daraus ergibt sich folgendes Frequenzschema der AD-Wandlung:

1. Prozessortakt: 1,024 MHz
2. Clock prescaler: 128
3. Anzahl Takte pro Wandlung: 13
4. Anzahl Wandlungen pro Zyklus: 64
5. Wandlungsfrequenz: 9,62 Hz
6. Zeitbedarf: 104 ms

4.6.3 Multiplexen

Die vier Bytes mit den bei den vier Ziffern einzuschaltenden Leds sind im SRAM abgelegt. Sie werden mit 500 Hz im TC0-Overflow-Int nacheinander ausgegeben. Die Ausgabe erfolgt in umgekehrter Reihenfolge von Ziffer 4 nach Ziffer 1, weil sich dann das Ende des Zyklus leichter feststellen lässt.

Bevor die nächste Byte-Kombination ausgegeben wird, werden die Anodentreiber abgeschaltet, so dass kein zeitweises Überblenden erfolgen kann.

4.6.4 Einstellung der Zeit mit den Tasten

Die Einstellung der Zeit wird gestartet, wenn Taste 1 das erste Mal auf niedriges Potenzial geht. Dies wird im PCINT festgestellt, der für die beiden Tasteneingänge ermöglicht ist. Der Tasteninterrupt wird nur akzeptiert, wenn der Torkelzähler auf Null steht und der Tasteneingang auf Low geht.

Das Drücken von Taste 1 bewirkt, dass eine Flagge gesetzt wird. Diese Flagge schaltet die AD-Wandler-Messungen auf den Poti-Eingang um. Die fertigen Messwertsummen werden im Hardware-Multiplikator des ATmega48 mit 24 multipliziert, das Ergebnis in BCD umgewandelt, die beiden Nibble in Siebensegmentformat umkodiert und in die ersten beiden

Zellen des Multiplexspeichers geschrieben (Stundenanzeige). Befindet sich der Sekundenteiler der Uhr in weniger als 25% Restzeit, dann erfolgt das nicht und die beiden Stellen werden durch Schreiben von Nullen geleert. Dadurch wird das Blinken erreicht.

Der nächste Tastendruck auf Taste 1 (nach Ablauf der Torkelzeit) bewirkt, dass zusätzlich das Minuteneingabe-Flag gesetzt wird. Die Messwerte werden dann mit 60 multipliziert und die Siebensegment Kodierung in den Stellen 3 und 4 des SRAM-Speichers abgelegt.

Ein dritter Tastendruck auf Taste 1 setzt die nunmehr eingestellte Zeit in die aktuellen Zeitregister, stellt den Sekunden- und den Minutenteiler auf ihren Anfangswert und löscht eventuell bereits gesetzte Flaggen des Zeitgebers. Ab jetzt wird wieder die Uhrzeit ausgegeben.

Drücken auf Taste 2 bewirkt, dass um eine Stelle zurück gewechselt wird (von Minuten auf Stunden, von Stunden auf Eingabe-Aus).

Jeder Tastendruck setzt den Torkelzähler auf einen Anfangswert. Er wird vom TC0-Overflow-Interrupt alle zwei Millisekunden um Eins vermindert, solange keine Taste gedrückt ist. Dieser Mechanismus unterbindet Tastenprellen.

4.6.5 Einstellung der Zeit mit DCF77

Jede Pegeländerung am DCF77-Eingang löst einen PCINT aus und eine Flagge wird gesetzt.

Die gesetzte Flagge bewirkt außerhalb der Interrupt-Service-Routine, dass der Zählerstand des TC1 abgelesen wird. Dieser wird mit einem Vorteiler von 1.024, also mit 4.000 Hz getaktet, was einer Zeit von 0,25 ms pro Zählererhöhung entspricht. Der Zähler wird mit jeder Eingangsflanke wieder rückgesetzt, so dass die Dauer jeder Pegeländerung exakt erfasst wird.

Ein Null-Bit von DCF77 sollte um die 100 ms lang dauern, setzt man 10% Toleranz an ist 90 bis 110 ms eine zulässige Dauer. Mit dem Takt des TC1 entspricht dies einem Zählerstand zwischen 360 und 440. Liegt die Dauer eines Signals in diesem Bereich, handelt es sich um eine übertragene Null.

Entsprechend hat jede Signaldauer ihre Zuordnung. So liegt ein Minutenwechsel mit 1800 bis 1900 ms und bei 10% Toleranz irgendwo zwischen 1665 (Zählerstand 6660) und 2035 (Zählerstand 8140). Da bei allen Pegelwechseln ein Interrupt auftritt, ist auch die Pause zwischen einer Null oder Eins bis zum nächsten Sekundenbeginn eine zulässige Signaldauer (Sekundenpause).

Daraus ergibt sich folgender Algorithmus:

1. Ist die Signaldauer kürzer als der erste Wert, liegt ein Fehler vor.
2. Ist die Signaldauer gleich oder höher als der erste Wert und kleiner als der zweite Wert, liegt eine Null, eine Eins, eine Sekundenpause oder ein Minutenwechsel vor.
3. Liegt die Signaldauer bei oder über dem zweiten Wert, muss die nächste Kategorie geprüft werden. Ist der Minutenwechsel überschritten, liegt ebenfalls ein Fehler vor.

	> =	<
Null-Bit	Untergrenze	Obergrenze + 1
Eins-Bit	Untergrenze	Obergrenze + 1
Signalpause	Untergrenze	Obergrenze + 1
Minutenpause	Untergrenze	Obergrenze + 1

Nullen und Einsen werden mit ROR in einen Pufferspeicher eingeschoben, der mindestens 40 Bits fasst. Dabei werden sie gezählt.

Sekundenpausen werden ignoriert und übergangen.

Tritt ein Minutenwechsel auf, dann

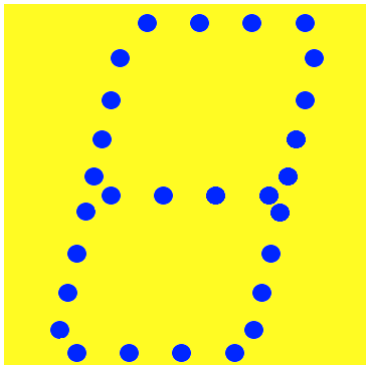
1. wird geprüft, ob exakt 59 Datenbits eingegangen waren,
2. werden die Minuten aus dem Bitstrom extrahiert, auf ein korrektes Paritäts-Bit (even parity) hin überprüft und in den Minutenspeicher abgelegt,
3. werden die Stunden extrahiert, ebenfalls auf ein korrektes Paritätsbit hin geprüt und im Stundenspeicher abgelegt,
4. sind beide korrekt empfangen, werden Stunden und Minuten in die Zeitregister kopiert, die Sekunden- und Minutenteiler neu gestartet und zwischenzeitlich gesetzte Flaggen gelöscht.

Dies sind die Zahlenwerte der Tabelle mit den Dauern der DCF77-Signale in TC1-Timer-Ticks im Normalfall. Man sieht, es gibt keine Überlappung. Und immerhin entspricht jeder Timer-Tick 1.024 Einzelschritten des Prozessors, so dass auch bei einer zeitlichen Verspätung des Ablesens des Wertes keine wesentliche Verzögerung eintritt.

	> =	<
Null-Bit	90	111
Eins-Bit	180	221
Signalpause	765	936
Minutenpause	1.665	2.036

Seitenanfang | Eigenschaften | Hardware | Aufbau | Software

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



AVR-Anwendungen

Groß-Uhr mit ATmega48
Assembler Software für die Groß-Uhr

Assembler-Quellcode für die Groß-Uhr mit ATmega48

Der Original-Assembler-Quellcode ist hier.

```
;
; ********************************
; * Grossuhr mit ATmega48        *
; * (C)2019 avr-asm-tutorial.net *
; ********************************
;
.nolist
.include "m48adef.inc" ; Define device ATmega48A
.list
;
; ********************************
;    D E B U G   S W I T C H E S
; ********************************
;
; Switches debug options on/off
;   Make sure that all switches are off in the final version
;
.equ Yes = 1 ; For debug switches
.equ No = 0
;
; Debug the green led
.equ Debug_ledgreen = No ; Debug the green led only
;
; Debug the leds on start-up
.equ Debug_leds = No ; Debug the leds on start-up
;
; Debug the currents and blink the green led
.equ Debug_current = No ; Switch the current drivers on
;
; Debug the segments of the display
.equ Debug_segments = No ; Switch the segments on
  ; 1 = 9 seconds for all four digits
  ; 10 = slow, 80 seconds for all four digits
  ; 100 = extremely slow, 25 seconds per digit
  .equ cDebug_segDelay = 100 ; Delay the active segment
;
; Debug the muxing
.equ Debug_mux8 = No ; Multiplex the four displays
```

```
  ; The mux frequency for all four digits once
  ; Must be between 4 and 10,000 Hz
  .equ cDebug_muxfreq = 150 ; MUX frequency in Hz
;
; Debug the ADC results
.equ Debug_adc = No ; Displays the ADC results
;
; Debug the two keys
.equ Debug_keys = No ; Displays the key status
;
; Debug errors in DCF signal reception
;   Displays E and error number instead of hours
.equ Debug_dcferr = No ; Yes/No
.equ Debug_dcfany = No ; Displays any signals
.equ Debug_dcfdur = No ; Displays signal durations
;
; *********************************
;        H A R D W A R E
; *********************************
;
; Device: ATmega48A, Package: 28-pin-PDIP-S
;
;           __________
;        1 /          |28
;  Res o--|RESET   PC5|--o DCF77-In
;    a o--|PD0     PC4|--o Key1-In
;    b o--|PD1     PC3|--o Key2-In
;    c o--|PD2     PC2|--o Led green cathode
;    d o--|PD3     PC1|--o Optosensor
;    e o--|PD4     PC0|--o Pot
;  +5V o--|VCC     GND|--o 0V
;   0V o--|GND    AREF|--o 100nF
;   X1 o--|PB6    AVCC|--o +5V
;   X2 o--|PB7     PB5|--o SCK
;    f o--|PD5     PB4|--o MISO
;    g o--|PD6     PB3|--o MOSI+A4
;    h o--|PD7     PB2|--o A3
;   A1 o--|PB0     PB1|--o A2
;       14|___________|15
;

; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
.equ p7SegO = PORTD ; Seven-segment output port
.equ p7SegD = DDRD ; Seven-segment direction port
.equ pAnodeO = PORTB ; Anode driver output port
.equ pAnodeD = DDRB ; Anode driver direction port
.equ pLedGO = PORTC ; Green led output port
.equ pLedGD = DDRC ; Green led direction port
.equ pLedGI = PINC ; Green led input port
.equ bLedGO = PORTC2 ; Green led portbit output
.equ pDcfKeyO = PORTC ; DCF and key port output port
.equ pDcfKeyD = DDRC ; DCF and key direction port
.equ pDcfKeyI = PINC ; DCF&Key input port
.equ bDcfI = PINC5 ; DCF77 input pin
.equ bKey1I = PINC4 ; Key1 input portpin
.equ bKey2I = PINC3 ; Key2 input portpin
;
; *********************************
```

```
;   A D J U S T A B L E   C O N S T
; *********************************
;
; Frequency of the external xtal
;   has to be dividable by 2/4, 8 and 256 (=4096/8192)
;   without any fractional remainder
.equ xtal = 4096000 ; in Hz
;
; Clock prescaler applied
.equ cClkDiv = 4 ; Either one, two, four or eight
;
; Start time setting of the clock
;   Packed BCD format
.equ cStartHours = 0x20 ; Start at 20:00 h
.equ cStartMinutes = 0x00
;
; DCF time only (clear display as long as not synced with DCF77)
.equ cDcfOnly = No ; Display/Clear unsynced time
;
; DCF signal durations in ms
.equ dcfdur_personally = No ; No = set default durations
;
.if dcfdur_personally != Yes
  .equ tDcf0 = 100 ; 100 ms for a 0
  .equ tDcf1 = 200 ; 200 ms for a 1
  .equ tDcfP = 850 ; Pause for 0 and 1 to next second
  .equ tDcfM = 1850 ; Pause for 59th second pulse
  .equ tDcfT = 3000 ; Time-out of DCF signal
   ; DCF77 signal duration tolerance
  .equ cDcfTol = 15; Tolerance in %
  .else
  .equ tDcf0 = 80 ; for too short signals
  .equ tDcf1 = 190 ; 200 ms for a 1
  .equ tDcfP = 850 ; Pause for 0 and 1 to next second
  .equ tDcfM = 1850 ; Shorter period for 59th second pulse
  .equ tDcfT = 3000 ; Time-out of DCF signal
  ; DCF77 signal duration tolerance
  .equ cDcfTol = 25; Tolerance in %
  .endif
;
;
; DCF77 signal input pull-up resistor
.equ cDcfPullUp = No ; Yes or no
;
; The anode line to which the double point in the
;   middle is attached to
.equ cAnDp = 3 ; Should be between 1 and 4
;
; Key bouncing period
.equ tBounce = 50 ; Bouncing time, in ms
;
; Skip input mode after inactive time
.equ cSkipInpMinutes = 10 ; Minutes until input has to be finished
;
; Select the dimming source
;   0: Selects the potentiometer
;   1: Selects the opto sensor
.equ cDimOpto = No ; Select the source
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
```

```
; *********************************
;
; Clock divider conversion
.if cClkDiv == 1
  .equ cClkPr = 0
  .else
  .if cClkDiv == 2
    .equ cClkPr = 1
    .else
    .if cClkDiv == 4
      .equ cClkPr = 2
      .else
      .if cClkDiv == 8
        .equ cClkPr = 3
        .else
        .error "cClkDiv has illegal value!"
        .endif
      .endif
    .endif
  .endif
;
; Clock frequency
.equ clock=xtal/cClkDiv ; Define clock frequency
;
; Half seconds and minute dividers
.equ cTc0Prsc = 8 ; TC0 prescaler
.equ cTc0Frq = clock / cTc0Prsc / 256 ; TC0 int frequency
.equ cSec2 = (cTc0Frq+1) / 2 ; Half second counter
.equ c75pcon = cSec2 / 4 ; Period over which the
   ; selected digits are displayed when input is
   ; active
   .if c75pcon>255
     .error "Off period too long, reduce c75pcon!"
      .endif
;
; DCF signal counts, with rounding
.equ cTc1Tick = (clock+512) / 1024 ; Timer TC1 tick in Hz, @4.096 MHz = 4000 Hz
.equ cDcf0Min = ((tDcf0-tDcf0*cDcfTol/100)*cTc1Tick+500)/1000 ; Count 0 minimum
.equ cDcf0Max = ((tDcf0+tDcf0*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count 0
maximum
.equ cDcf1Min = ((tDcf1-tDcf1*cDcfTol/100)*cTc1Tick+500)/1000 ; Count 1 minimum
.equ cDcf1Max = ((tDcf1+tDcf1*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count 1
minimum
.equ cDcfPMin = ((tDcfP-tDcfP*cDcfTol/100)*cTc1Tick+500)/1000 ; Count pause
minimum
.equ cDcfPMax = ((tDcfP+tDcfP*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count pause
maximum
.equ cDcfMMin = ((tDcfM-tDcfM*cDcfTol/100)*cTc1Tick+500)/1000 ; Count minute
minimum
.equ cDcfMMax = ((tDcfM+tDcfM*cDcfTol/100)*cTc1Tick+500)/1000+1 ; Count minute
maximum
.equ cDcfT = (tDcfT*cTc1Tick+500)/1000+1 ; Counter time out
.if cDcfT>65535
  .error "Clock frequency too high for DCF duration counting"
  .endif
.if (cDcf0Max>=cDcf1Min)||(cDcf1Max>=cDcfPMin)||(cDcfPMax>=cDcfMMin)
  .error "Overlapping DCF duration(s), reduce cDcfTol!"
  .endif
;
.if cDcfPullUp == Yes
 .equ mDcfKeyO = (1<<PORTC5)|(1<<PORTC4)|(1<<PORTC3)
```

```
 .else
 .equ mDcfKeyO = (1<<PORTC4)|(1<<PORTC3)
 .endif
;
; Key bouncing constant
.equ cTc0Presc = 64 ; TC0 prescaler value
.equ cTc0Mux = clock / cTc0Presc / 256 ; MUX interrupt frequency
.equ cBounce = (tBounce*cTc0Mux+500)/1000 ; Bounce constant
;
; *********************************
;             C L O C K S
; *********************************
;
; Default xtal frequency 4.096 MHz, CLKPR=4, effective clock=1.024 MHz
;
; TC0:
;   Clocked with a prescaler of 8, clock tick = 128 kHz @ 1.024 MHz
;   Fast PWM counting, TOP = 0xFF, overflow int = 500 Hz (2 ms) @ 1.024 MHz
;     MUX-frequency = 500 / 4 = 125 Hz
;   16-Bit-Register downcount from 1,000, yields 2 Hz signal for blinking the
double dot
;     If zero: Set bSec flag, register downcount from 120 to yield minute for
clock,
;       If zero: Set bMin flag
;   If rBounce not at zero:
;     Both key inputs high: downcount rBounce, otherwise rBounce = cBounce
;   Timer interrupt on overflow
;   Compare match A: OCR0A interrupt (clears anode driver for dimming the LEDs)
; TC1:
;   Counts the duration of DCF77 signals
;     Clocked with a prescaler of 1,024 = 1 kHz (1.0 ms) @ 1.024 MHz
;   Normal counting (cleared by PCINT on DCF signal input)
;   Timer interrupt on compare match A: sets the bDcfTO flag
; ADC:
;   Converts measurements on ADC0/ADC1 inputs
;   Clock prescaler = 128
;   N clock cycles per conversion = 13
;   Conversion frequency = 615.38 Hz (1.625 ms) @ 1.024 MHz
;   Sums up 64 ADC results = 104 ms @ 1.024 MHz
;   If 64 adders complete: Set bAdc flag
;     ca. 1 MUX cycle per update
;     As the TC0 compare match update is done at the beginning
;     of the next mux event (each 2 ms), no missing compare matches
;     occur (no flickering)
;
; *********************************
;       R E G I S T E R S
; *********************************
;
; used: R1:R0 for DCF signal duration and
      ; for hardw multiplication
.def rAdcCtr = R2 ; ADC sum counter
.def rAdcSumL = R3 ; ADC result sum, LSB
.def rAdcSumH = R4 ; dtp., MSB
.def rAdc = R5 ; ADC result MSB
.def rInput = R6 ; Input pins
.def rDcfBits = R7 ; Counter for DCF bits
.def rDcf3 = R8 ; DCF bits, byte 4
.def rDcf4 = R9 ; DCF bits, byte 5
.def rDcf5 = R10 ; DCF bits, byte 6
.def rDcf6 = R11 ; DCF bits, byte 7
```

```
.def rDcf7 = R12 ; DCF bits, byte 8
.def rDcfErr = R13 ; DCF77 signal error
.def rMux = R14 ; Mux channel
.def rSreg = R15 ; Save status register
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
.def rFlag = R18 ; Flag register
  .equ bMin = 0 ; A minute is over
  .equ bSec2 = 1 ; A half second is over
  .equ bDcf = 2 ; Level change on DCF input
  .equ bDcfTO = 3 ; Time out Dcf input signal
  .equ bKey1 = 4 ; Key1 pressed
  .equ bKey2 = 5 ; Key2 pressed
  .equ bKeyA = 6 ; Key input active
  .equ bKeyM = 7 ; Minute key input active
.def rFlag2 = R19 ; Second flag register
  .equ bAdc = 0 ; An ADC result is available
.def rHours = R20 ; Hours time, packed BCD
.def rMinutes = R21 ; Minutes time, packed BCD
.def rBounce = R22 ; Debouncing key counter
.def rMin = R23 ; Minute counter
.def rSec2L = R24 ; 1/2 seconds counter, LSB
.def rSec2H = R25 ; dto., MSB
; used: R27:R26 = X as pointer
; used: R29:R28 = Y for MUX
; used: R31:R30 = Z for multiple purposes outside int
;
; *********************************
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
sMux:
.byte 4 ; 4 bytes for muxing the display
sMuxEnd:
;
sInpTime:
.byte 2 ; 2 bytes input time buffer
;
sSkipInp:
.byte 1 ; Skip input after inactive time
;
; If debug any DCF signals
sDcfPos:
.byte 1 ; Position of the next display (+1)
;
; *********************************
;         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
      reti ; INT0
      reti ; INT1
      reti ; PCI0
```

```
      rjmp Pci1Isr ; PCI1 for DCF and key input changes
      reti ; PCI2
      reti ; WDT
      reti ; OC2A
      reti ; OC2B
      reti ; OVF2
      reti ; ICP1
      rjmp Tc1CmpAIsr ; OC1A, Dcf77 time-out
      reti ; OC1B
      reti ; OVF1
      rjmp Tc0CmpAIsr ; OC0A: clear anode driver
      reti ; OC0B
      rjmp Tc0OvfIsr ; MUX and time
      reti ; SPI
      reti ; URXC
      reti ; UDRE
      reti ; UTXC
      rjmp AdcIsr ; ADCC, Conversion complete
      reti ; ERDY
      reti ; ACI
      reti ; TWI
      reti ; SPMR
;
; *********************************
;  I N T - S E R V I C E   R O U T .
; *********************************
;
; PCI1 Interrupt service routine
Pci1Isr:
  in rSreg,SREG ; Save SREG
  in rimp,pDcfKeyI ; Read DCF signal and keys
  eor rimp,rInput ; Compare with last input
  sbrc rimp,bDcfI ; DCF77 bit set?
  sbr rFlag,1<<bDcf ; Set DCF flag
Pci1Isr1:
  tst rBounce ; Check bouncing counter
  brne Pci1Isr3 ; Still bouncing, ignore
  ; rInput key bit was 1/0, now is 0/1: EOR bit is one
  sbrs rimp,bKey1I ; Key 1 input changed?
  rjmp Pci1Isr2 ; No, skip
  sbis pDcfKeyI,bKey1I ; Input is one?
  sbr rFlag,1<<bKey1 ; Set key 1 flag
Pci1Isr2:
  sbrs rimp,bKey2I ; Key 2 input changed?
  rjmp Pci1Isr3 ; No, skip
  sbis pDcfKeyI,bKey2I ; Input is one?
  sbr rFlag,1<<bKey2 ; Set key 2 flag
Pci1Isr3:
  in rInput,pDcfKeyI ; Read DCF signal and keys again
  out SREG,rSreg ; Restore SREG
  reti
;
; TC0 overflow interrupt service routine
Tc0OvfIsr:
  in rSreg,SREG ; Save SREG
  clr rimp ; Anodes off
  out pAnode0,rimp
  ld rimp,-Y ; Read next mux byte
  out p7Seg0,rimp ; Cathodes out
  out pAnode0,rMux ; Set anodes
  lsr rMux ; Next lower anode
```

```
  brcc Tc0OvfIsr1 ; Not at the end
  ldi rimp,0b00001000 ; Start with anode 4
  mov rMux,rimp ; in rMux
  ldi YH,High(sMuxEnd) ; Restart from the end
  ldi YL,Low(sMuxEnd)
Tc0OvfIsr1:
  tst rBounce ; Check if bouncing active
  breq Tc0OvfIsr4 ; No, skip
  sbis pDcfKeyI,bKey1I ; Key 1 not pressed?
  rjmp Tc0OvfIsr2 ; No, restart bouncing
  sbic pDcfKeyI,bKey2I ; Key 2 pressed?
  rjmp Tc0OvfIsr3 ; No, decrease bounce count
Tc0OvfIsr2:
  ldi rBounce,cBounce ; Restart bouncing
  rjmp Tc0OvfIsr4 ; Continue ISR
Tc0OvfIsr3:
  dec rBounce ; Down count rBounce
Tc0OvfIsr4:
  sbiw rSec2L,1 ; Decrease seconds divider
  brne Tc0OvfIsr5 ; Not zero, skip
  ldi rSec2H,High(cSec2) ; Restart seconds divider
  ldi rSec2L,Low(cSec2)
  sbr rFlag,1<<bSec2 ; Set half second flag
  dec rMin ; Decrease minute divider
  brne Tc0OvfIsr5 ; Not zero, skip
  ldi rMin,120 ; Restart minute counter divider
  sbr rFlag,1<<bMin ; Set minute flag
Tc0OvfIsr5:
  out SREG,rSreg ; Restore SREG
  reti
;
Tc1CmpAIsr:
  in rSreg,SREG ; Save SREG
  sbr rFlag,1<<bDcfTO ; Set DCF time-out flag
  out SREG,rSreg ; Restore SREG
  reti
;
; TC0 Compare A Interrupt service routine
Tc0CmpAIsr:
  ldi rimp,0 ; Clear anode driver
  out pAnodeO,rimp ; in anode port
  reti
;
; ADC conversion complete interrupt service routine
AdcIsr:
  in rSreg,SREG ; Save SREG
  lds rimp,ADCL ; Read LSB result
  add rAdcSumL,rimp ; Add to LSB sum
  lds rimp,ADCH ; Read MSB
  adc rAdcSumH,rimp ; Add this and carry to MSB sum
  dec rAdcCtr ; Decrease counter
  brne AdcIsr1 ; Not zero
  sbr rFlag2,1<<bAdc ; Set flag
  out SREG,rSreg ; Restore SREG
  reti
AdcIsr1:
  ldi rimp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rimp ; Restart ADC
  out SREG,rSreg ; Restore SREG
  reti
;
```

```
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
Main:
  ; Init stack
      ldi rmp,High(RAMEND)
      out SPH,rmp ; Init MSB stack pointer
      ldi rmp,Low(RAMEND)
      out SPL,rmp ; Init LSB stack pointer
  ; Init clock prescaler
  ldi rmp,1<<CLKPCE ; Activate clock prescaler
  sts CLKPR,rmp ; in clock prescaler port
  ldi rmp,cClkPr ; Set new prescaler
  sts CLKPR,rmp ; in clock prescaler port
;
; *****************************
;  H A R D W A R E   D E B U G
; *****************************
;
; Debug hardware options
;
; Debug the current drivers
.if Debug_current == Yes
  ldi rmp,0xFF ; All driver pins as output
  out p7SegD,rmp ; in direction port
  out p7SegO,rmp ; and activated
  .endif
;
; Blink the green LED
.if (Debug_ledgreen == Yes)||(Debug_current == Yes)
    sbi pLedGD,bLedGO ; enable output
  Debug_ledgreen1:
    rcall ToggleGreen
  Debug_ledgreen2:
    sbiw ZL,1
    brne Debug_ledgreen2
    rjmp Debug_ledgreen1
  .endif
;
; Debug the segments
.if Debug_segments == Yes
  ; Hint: uses rHours as anode driver and
  ; rMinutes as cathode driver
    ldi rmp,1<<bLedGO ; Green led as output
      out pLedGD,rmp ; Set portpin direction
    rcall ToggleGreen ; Toggle the green led
    ldi rmp,0xFF ; All cathode pins as outputs
    out p7SegD,rmp ; in direction port
    clr rmp ; All cathodes off
    ldi rmp,0b00001000 ; Anode driver 4 as output
    out pAnodeD,rmp ; in direction port
    clr rmp ; Anode drivers off
    out pAnodeO,rmp ; in anode port
  Debug_seg1:
    ; Start with digit 1
    ldi rHours,0x01 ; Start with digit 1 anode
  Debug_seg2:
    ldi rmp,0
      out pAnodeO,rmp
    ldi rMinutes,0x01 ; and with the first segment
```

```
  Debug_seg3:
    out p7SegO,rMinutes ; Activate the cathodes
      out pAnodeO,rHours ; and the anode
    ldi rmp,cDebug_segDelay ; Load segment delay counter
  Debug_seg4:
    sbiw ZL,1 ; down-count delay
    brne Debug_seg4 ; until zero
    dec rmp ; Repeat counter
    brne Debug_seg4 ; Additional delay
    lsl rMinutes ; next segment
    brcc Debug_seg3
    lsl rHours
    sbrs rHours,4 ; Bit 4 zero?
    rjmp Debug_seg2 ; No, next digit
    rjmp Debug_seg1 ; Restart with digit 1
  .endif
;
; Debug muxing
.if Debug_mux8 == Yes ; Multiplex the four displays
  ; Wait time for MUX frequency
  ;   Delay of loop is N = 2 + 4 * (c-1) + 3
  ;   c = (N-5) / 4 + 1
  ;   c= (clock/fMux/4-5)/4+1
  .equ cDebug_muxdelay =(clock/cDebug_muxfreq/4-5)/4+1
  ; Hint: uses rHours as anode driver
    ldi rmp,1<<bLedGO ; Green led as output
      out pLedGD,rmp ; Set portpin direction
    rcall ToggleGreen ; Toggle the green led
    ldi rmp,0xFF ; All cathode pins as outputs
    out p7SegD,rmp ; in direction port
    ldi rmp,0xFF ; All cathodes on
      out p7SegO,rmp
    ldi rmp,0b00001000 ; Anode driver 4 as output
    out pAnodeD,rmp ; in direction port
  Debug_mux8a:
      ldi rHours,0x01 ; Anode driver 1 on
  Debug_mux8b:
      out pAnodeO,rHours
    ldi ZH,High(cDebug_muxdelay)
      ldi ZL,Low(cDebug_muxdelay)
  Debug_mux8c:
    sbiw ZL,1
      brne Debug_mux8c
      lsl rHours
      sbrs rHours,4
      rjmp Debug_mux8b
      rjmp Debug_mux8a
  .endif
;
; ******************************
;     N O R M A L   I N I T
; ******************************
;
  ; Init ports
  clr rmp ; Anodes off
  out pAnodeO,rmp ; in Anode port
  ldi rmp,0b00001000 ; Anode 4 as output pin
  out pAnodeD,rmp ; in anode port
  clr rmp ; Outputs cathodes low
  out p7SegO,rmp ; to portpins
  ldi rmp,0xFF ; Port as output
```

```
  out p7SegD,rmp ; to direction port
  clr rmp ; DCF and key inputs off
  out pDcfKeyD,rmp ; in DCF and key inputs
  ldi rmp,mDcfKeyO ; Set pull-ups
  out pDcfKeyO,rmp ; in DCF and key pins
  ; Init TC0 for MUX
  ldi rmp,0x01 ; Start with very short dim period
  out OCR0A,rmp
  ldi rmp,(1<<WGM00)|(1<<WGM01) ; Fast PWM mode
  out TCCR0A,rmp
  ldi rmp,(1<<CS01) ; Prescaler=8
  out TCCR0B,rmp
  ldi rmp,(1<<TOIE0)|(1<<OCIE0A) ; Interrupt on overflow and compare A
  sts TIMSK0,rmp
  ; Init TC1
  ldi rmp,High(cDcfT) ;  DCF time-out, MSB
  sts OCR1AH,rmp
  ldi rmp,Low(cDcfT) ; dto, LSB
  sts OCR1AL,rmp
  clr rmp ; CTC mode on compare A
  sts TCCR1A,rmp
  ldi rmp,(1<<CS10)|(1<<CS12)|(1<<WGM12) ; CTC on compare A, presc=1024
  sts TCCR1B,rmp
  ldi rmp,1<<OCIE1A ; Timer int mask for compare A
  sts TIMSK1,rmp ; to int mask port TC1
  ; Init ADC
  ldi rmp,64 ; Start ADC counter
  mov rAdcCtr,rmp ; ... with 64
  ldi rmp,(1<<REFS0) ; MUX to channel ADC0
  sts ADMUX,rmp ; Set channel selection
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rmp ; Restart ADC
  ; Init flags and other parameters
  clr rFlag
  ldi rSec2H,High(cSec2) ; Init second counter, MSB
  ldi rSec2L,Low(cSec2) ; dto., LSB
  ldi rMin,120 ; Init minute counter
  ldi rHours,cStartHours ; Set initial time, hours
  ldi rMinutes,cStartMinutes ; dto., minutes
  rcall SetTime ; Convert time to display mux
  ldi YH,High(sMuxEnd)
  ldi YL,Low(sMuxEnd)
  ldi rmp,0b00010000
  mov rMux,rmp
  ldi rmp,Low(sMuxEnd) ; DCF output position
  sts sDcfPos,rmp
  ; Init PCINT
  in rInput,pDcfKeyI ; Read inputs
  ldi rmp,(1<<PCINT13)|(1<<PCINT12)|(1<<PCINT11) ; The interrupt generators
  sts PCMSK1,rmp
  ldi rmp,1<<PCIE1 ; PCINT1 enable
  sts PCICR,rmp
  ; Sleep mode
  ldi rmp,1<<SE ; Sleep mode idle
  out SMCR,rmp ; in SMCR
  ; Enable interrupts
      sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
```

```
;
Loop:
  sleep ; Go to sleep
  nop ; after wake-up
  sbrc rFlag,bDcfTo ; DCF77 time-out clear?
  rcall DcfTimeOut ; Yes, time out
  sbrc rFlag,bDcf ; DCF input level change clear?
  rcall Dcf ; Yes, analyze
  sbrc rFlag,bKey1 ; Key 1 pressed?
  rcall Key1 ; Yes, react
  sbrc rFlag,bKey2 ; Key 2 pressed?
  rcall Key2 ; Yes, react
  sbrc rFlag,bMin ; Minute flag clear?
  rcall Minute ; Set, go to minutes
  sbrc rFlag,bSec2 ; Second flag clear?
  rcall Second ; Set, go to seconds
  sbrc rFlag2,bAdc ; ADC flag clear?
  rcall AdcFlag ; Set, go to ADC conversion
  .if Debug_keys == Yes
    rcall KeyDisplay
      .endif
  rjmp loop ; Restart loop from the beginning
;
; *********************************
;    F L A G   R E A C T I O N S
; *********************************
;
; *********************************
;  H A L F   S E C O N D   O V E R
; *********************************
;
; Half second over, blink double point
Second:
  cbr rFlag,1<<bSec2 ; Clear flag
  lds rmp,sMux+cAnDp-1 ; Read mux byte where double point is attached to
  sbrc rmp,7 ; Seventh bit clear?
  rjmp Second1
  ori rmp,0x80 ; Set seventh bit
  sts sMux+cAnDp-1,rmp ; Bit 7 high to mux
  ret
Second1:
  andi rmp,0x7F ; Clear seventh bit
  sts sMux+cAnDp-1,rmp ; Bit 7 low to mux
  ret
;
; *********************************
;      M I N U T E   O V E R
; *********************************
;
; A minute is over, increase time
Minute:
  cbr rFlag,1<<bMin ; Clear flag
  lds rmp,sSkipInp ; Read skip input time
  tst rmp ; At zero?
  breq Minute0
  dec rmp
  sts sSkipInp,rmp
  brne Minute0
  cbr rFlag,(1<<bKeyA)|(1<<bKeyM) ; Clear input flags
Minute0:
  ldi rmp,0x07 ; Add 7 to BCD
```

```
  add rMinutes,rmp ; to minutes
  brhs Minute1 ; Half overflow
  ldi rmp,0x06 ; Subtract 6
  sub rMinutes,rmp ; from minutes
Minute1:
  cpi rMinutes,0x60 ; 60 minutes over?
  brcs SetTime ; Set the time
  clr rMinutes ; Restart at zero
  ldi rmp,0x07 ; Add 7 to BCD
  add rHours,rmp ; to hours
  brhs Minute2 ; Half overflow
  ldi rmp,0x06 ; Subtract 6
  sub rHours,rmp ; from hours
Minute2:
  cpi rHours,0x24 ; Next day?
  brcs SetTime ; No
  clr rHours ; Restart day
;
; *********************************
;      D I S P L A Y   T I M E
; *********************************
;
; Convert the hhmm time and display
SetTime:
  .if cDcfOnly == Yes
    ; Do not update time
    ret
    .endif
SetTime1:
  sbrc rFlag,bKeyA ; Is a key input active?
  ret ; Yes, skip time output
  ldi XH,High(sMux)
  ldi XL,Low(sMux)
  mov rmp,rHours ; Read hours
  rcall Convert2Seven
  mov rmp,rMinutes
  rjmp Convert2Seven
;
; ******************************
;    D C F 7 7   S I G N A L S
; ******************************
;
; DCF77 Time-Out signal input
DcfTimeOut:
  cbr rFlag,1<<bDcfTo ; Clear flag
  .if cDcfOnly == Yes
    ldi rmp,0 ; Clear the MUX area
    sts sMux,rmp
    sts sMux+1,rmp
    sts sMux+2,rmp
    sts sMux+3,rmp
    .endif
  ret
;
; Active DCF signal
Dcf:
  cbr rFlag,1<<bDcf ; Clear flag
  lds XL,TCNT1L ; Read TC1 count, LSB first
  lds XH,TCNT1H ; dto., MSB next
  tst XH ; Check MSB
  brne Dcf1 ; Larger than zero, fine
```

```
  cpi XL,4 ; Minimum is 1 ms
  brcc Dcf1 ; Ok
  ret ; Ignore pulse, too short!
Dcf1:
  .if Debug_dcfdur == Yes
    ; Display signal duration in hex
      mov R1,XH ; Copy duration to R1:R0
      mov R0,XL
      ldi XH,High(sMux) ; X to sMux
      ldi XL,Low(sMux)
      mov rmp,R1 ; MSB first
      rcall Convert2Seven ; Write first two nibbles
      mov rmp,R0 ; LSB next
      rcall Convert2Seven ; Write second two nibbles
      mov XL,R0 ; Copy duration to X again
      mov XH,R1
      .endif
  ldi rmp,0xFF ; Counter for compares
  clr rDcfErr ; Error counter
  ldi ZH,High(2*DcfDur) ; Point Z to value table
  ldi ZL,Low(2*DcfDur)
Dcf2:
  inc rDcfErr ; Next DCF error
  inc rmp ; Next count
  cpi rmp,5 ; Maximum correct count = 4
  brcc DcfErr9 ; Error 9 (Signal too long)
  lpm R0,Z+ ; Read LSB min from table
  cp XL,R0 ; Compare with LSB min
  lpm R0,Z+ ; Read MSB min from table
  cpc XH,R0 ; Compare with MSB min
  brcs DcfError ; Error, signal too short
  lpm R0,Z+ ; Read LSB max from table
  cp XL,R0 ; Compare with LSB min
  lpm R0,Z+ ; Read MSB max from table
  brcc Dcf2 ; Larger than max table value
  cpi rmp,2 ; Zero or one?
  brcc Dcf4 ; No
  ; Received a correct bit
  .if Debug_dcfAny == Yes
    push rmp
    lsr rmp
      ldi rmp,0b01011100
      brcc Dcf2a
      ldi rmp,0b00000100
  Dcf2a:
    rcall DcfReport
      pop rmp
      .endif
  lsr rmp ; Shift counter bit 0 to carry
  ror rDcf7 ; Roll carry into DCF bit buffer
  ror rDcf6
  ror rDcf5
  ror rDcf4
  ror rDcf3
  inc rDcfBits
  rjmp DcfErrClear
Dcf3:
  .if Debug_dcfAny == Yes
    ldi rmp,0b01110011
    rcall DcfReport
    .endif
```

```
  rjmp DcfErrClear
Dcf4:
  cpi rmp,4
  brcs Dcf3 ; Pause, ignore
  ; Received a correct minute signal
  inc rDcfErr ; Next error
  ldi rmp,59 ; 59 bits received?
  cp rmp,rDcfBits ; Number of bits
  ldi rmp,0 ; Clear number of bits
  mov rDcfBits,rmp ; in counter register
  brne DcfError ; Next error
  inc rDcfErr ; DCF error 7
  lsr rDcf5 ; Shift Parity2 to carry
  ror rDcf4 ; and into Byte 5
  ror rDcf3 ; Minute 40s to byte 4
  lsr rDcf4 ; Shift hours right
  ror rDcf3 ; Shift parity1 to minutes
  mov rmp,rDcf4 ; Minutes to rmp
  rcall Parity ; Check parity in rmp
  brne DcfError ; Parity odd
  inc rDcfErr ; Next error
  mov rmp,rDcf3 ; Check parity minutes
  rcall Parity ; Check parity in rmp
  brne DcfError ; Parity odd
  ; All checks performed and errorfree
  mov rHours,rDcf4 ; Read hours
  andi rHours,0x3F ; Remove upper two bits
  mov rMinutes,rDcf3
  andi rMinutes,0x7F ; Isolate minutes
  rcall DcfErrClear
  rjmp SetTime1 ; Display time
;
; DCF signal errors
;   0: No error
;   1: Signal shorter than 0
;   2: Signal shorter than 1
;   3: Signal shorter than pause
;   4: Signal shorter than missing second
;   5: Signal longer than missing second
;   6: Not 59 seconds received
;   7: Minute parity is odd
;   8: Hour parity is odd
;   9: Time out of signal input
;
DcfErr9:
  ; Error 9: signal too long
  ldi rmp,9
  mov rDcfErr,rmp
DcfError:
.if Debug_dcferr==Yes
    ldi XH,High(sMux)
    ldi XL,Low(sMux)
    mov rmp,rDcfErr
    ori rmp,0xE0 ; Error sign
    rcall Convert2Seven
    clr rmp ; Clear the last two digits
    st X+,rmp
    st X+,rmp
  .endif
  ret
;
```

```
; Clear the DCF error number
DcfErrClear:
  .if Debug_dcferr == Yes
    ldi rmp,0
    sts sMux+1,rmp
      .endif
  ret
;
; Display DCF report in rmp
.if Debug_dcfAny == Yes
  DcfReport:
    ldi XH,High(sMux)
    lds XL,sDcfPos
    st -X,rmp
    cpi XL,Low(sMux)
    brne DcfReport1
    ldi XL,Low(sMuxEnd)
  DcfReport1:
    sts sDcfPos,XL
    ret
  .endif
;
; Check parity of rmp
Parity:
  clr ZL ; ZL is bit counter
Parity1:
  lsr rmp
  brcc Parity2
  inc ZL ; Count
Parity2:
  brne Parity1
  andi ZL,1
  ret
;
; DCF77 signal durations
DcfDur:
.dw cDcf0Min,cDcf0Max
.dw cDcf1Min,cDcf1Max
.dw cDcfPMin,cDcfPMax
.dw cDcfMMin,cDcfMMax
DcfDurEnd:
;
; ******************************
;   K E Y   P R O C E S S I N G
; ******************************
;
; Key1 is pressed
Key1:
  cbr rFlag,1<<bKey1 ; Clear flag
  ldi rmp,0b10101010
  sts sMux,rmp
  ldi rBounce,cBounce ; Start bouncing period
  sbrc rFlag,bKeyA ; Key input inactive?
  rjmp Key1Active ; Yes, key flag is active
  sbr rFlag,1<<bKeyA ; Set key flag active
  ret
Key1Active:
  sbrc rFlag,bKeyM ; Minute flag clear?
  rjmp Key1Minute ; No, go to minute
  sbr rFlag,1<<bKeyM ; Set M flag
  ldi rmp,24 ; Convert ADC to hours
```

```
  mul rmp,rAdc
  ldi XH,High(sInpTime) ; Point to hours input
  ldi XL,Low(sInpTime)
  rcall ToBcd ; Convert binary in rmp to packed BCD
  mov rmp,ZL ; Result to rmp
  ldi XH,High(sMux) ; Hours display
  ldi XL,Low(sMux)
  rjmp Convert2Seven ; Display the selected hours
;
Key1Minute:
  ldi rmp,60 ; Convert ADC to minutes
  mul rmp,rAdc
  ldi XH,High(sInpTime+1) ; Point X to minutes
  ldi XL,Low(sInpTime+1)
  rcall ToBcd ; Convert to BCD
  adiw XL,1 ; Point to behind minutes
  ld rMinutes,-X ; Result to minutes
  ld rHours,-X ; and hours
  ldi rSec2L,Low(cSec2) ; Restart half second divider
  ldi rSec2H,High(cSec2)
  ldi rMin,120 ; Restart minute counter
  cbr rFlag,(1<<bMin)|(1<<bSec2)|(1<<bKeyA)|(1<<bKeyM)
  rjmp SetTime ; Set the current time
;
; Key2 is pressed
Key2:
  cbr rFlag,1<<bKey2 ; Clear flag
  ldi rBounce,cBounce ; Start bouncing period
  sbrs rFlag,bKeyA ; Key input active?
  ret ; No, ignore key
  sbrc rFlag,bKeyM ; Minute active?
  rjmp Key2Minute ; Yes
  cbr rFlag,1<<bKeyA ; Stop input
  rjmp SetTime
;
Key2Minute:
  cbr rFlag,1<<bKeyM ; Return to hour input
  rjmp SetTime
;
; Convert the binary number in R1 to packed BCD in ZL
;   and write result to the X location
ToBcd:
  ldi ZL,-0x10
  ldi rmp,10
ToBcd1:
  subi ZL,-0x10
  sub R1,rmp
  brcc ToBcd1
  add R1,rmp
  add ZL,R1 ; Add the ones
  st X,ZL ; Store packed BCD in SRAM
  ret
;
; Displays the keys
KeyDisplay:
  ldi rmp,0b01011100 ; Small 0
  sbic pDcfKeyI,bKey1I ; Key 1
  ldi rmp,0b00000100 ; Small 1
  sts sMux,rmp
  ldi rmp,0b01011100 ; Small 0
  sbic pDcfKeyI,bKey2I ; Key 1
```

```
  ldi rmp,0b00000100 ; Small 1
  sts sMux+1,rmp
  ret
;
; ************************************
;     A D C   R E S U L T   R E A D Y
; ************************************
;
AdcFlag:
  cbr rFlag2,1<<bAdc ; Clear ADC flag
  mov rAdc,rAdcSumH ; Copy MSB sum
  clr rAdcSumL ; Restart sum
  clr rAdcSumH
  ldi rmp,64 ; Restart counter
  mov rAdcCtr,rmp ; into rAdcCtr
.if cDimOpto == Yes
  ldi rmp,(1<<REFS0)|(1<<MUX0) ; Mux to channel ADC1
  sbrc rFlag,bKeyA ; Clock setting active?
  ldi rmp,(1<<REFS0) ; Yes, MUX to channel ADC0
  .else
  ldi rmp,(1<<REFS0) ; MUX to channel ADC0
  .endif
  sts ADMUX,rmp ; Set new channel selection
  ldi rmp,(1<<ADEN)|(1<<ADSC)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0)
  sts ADCSRA,rmp ; Restart ADC
.if Debug_Adc == Yes
  rjmp AdcOut ; Display the ADC result
  .endif
  sbrc rFlag,bKeyA ; Key input active?
  rjmp AdcFlag2 ; Update the current input digit
  .if cDimOpto == Yes
    com rAdc ; Invert result
      .endif
AdcFlag1:
  out OCR0A,rAdc ; Write result to compare port TC0
    ; for dimming
  ret
AdcFlag2:
  tst rSec2H ; MSB larger than 0?
  brne AdcFlag3 ; Yes, display number
  cpi rSec2L,c75pcon ; LSB smaller than 75%
  brcc AdcFlag3
  ; Switch the two digits off
  ldi XH,High(sMux) ; Point X to hours
  ldi XL,Low(sMux)
  sbrc rFlag,bKeyM ; Minute key clear?
  ldi XL,Low(sMux+2) ; No, point to minutes
  clr rmp ; Write zeroes to digit
  st X+,rmp
  st X,rmp
  ret
AdcFlag3:
  ; Key input is active, calculate digit
  ldi rmp,24 ; Multiply ADC result by 24
  sbrc rFlag,bKeyM ; Minute flag clear?
  ldi rmp,60 ; Multiply ADC result by 60
  mul rmp,rAdc ; Hardware multiplication
  ; Convert MSB result to packed BCD
  ldi ZL,-0x10 ; Start with minus 10
  ldi rmp,10
AdcFlag4:
```

```
  subi ZL,-0x10 ; Add ten to result
  sub R1,rmp ; Subtract 10 from ADC MSB
  brcc AdcFlag4 ; If not carry continue subtracting
  add R1,rmp ; Undo last subtraction
  add ZL,R1 ; Add rest to result
  mov rmp,ZL
  ldi XH,High(sMux) ; Point X to hours
  ldi XL,Low(sMux)
  sbrc rFlag,bKeyM ; Minute key clear?
  ldi XL,Low(sMux+2) ; No, point to minutes
  rjmp Convert2Seven ; Convert to display
;
; Display the ADC results
AdcOut:
  ldi XH,High(sMux) ; Point X to sMux
  ldi XL,Low(sMux)
  ldi rmp,0b01110111 ; A sign
  st X+,rmp ; to hour tens
  mov R1,rAdc ; Move the rAdc to R1
  ldi rmp,100 ; Hundreds
  rcall AdcOutDec ; Count hundreds
  ldi rmp,10
  rcall AdcOutDec
  mov R0,R1 ; Display the rest
  rjmp AdcOutDec2
;
; Convert R1 to a decimal and display
;   rmp is the decimal (100, 10)
;   Uses R0
AdcOutDec:
  clr R0
  dec R0
AdcOutDec1:
  inc R0
  sub R1,rmp
  brcc AdcOutDec1
  add R1,rmp
; Convert R0 to 7segment and display
AdcOutDec2:
  ldi ZH,High(2*SevenSeg)
  ldi ZL,Low(2*SevenSeg)
  add ZL,R0
  ldi rmp,0
  adc ZH,rmp
  lpm rmp,Z
  st X+,rmp
  ret
;
; ************************************
;    B A S I C   S U B R O U T I N E S
; ************************************
;
; Toggles the green led
;   by outputting on the in port
ToggleGreen:
  ldi rmp,1<<bLedGO ; The green led
  out pLedGI,rmp ; Toggle the led
  ret
;
; Convert rmp to 7segment and write result to X
Convert2Seven:
```

```
  push rmp ; Save for LSB
  swap rmp
  rcall Convert2SevenDigit
  pop rmp
Convert2SevenDigit:
  andi rmp,0x0F
  ldi ZH,High(2*SevenSeg) ; Load table, MSB
  ldi ZL,Low(2*SevenSeg) ; dto., LSB
  add ZL,rmp ; Add number, LSB
  ldi rmp,0 ; Zero
  adc ZH,rmp ; Add carry
  lpm rmp,Z ; Read from flash
  st X+,rmp
  ret
;
; Seven-segment table
;     ---- a     hgfedcba
;  f |    | b
;     -g--
;  e |    | c
;     ----  d
;
SevenSeg:
.db 0b00111111,0b00000110 ; 0+1
.db 0b01011011,0b01001111 ; 2+3
.db 0b01100110,0b01101101 ; 4+5
.db 0b01111101,0b00000111 ; 6+7
.db 0b01111111,0b01101111 ; 8+9
.db 0b01110111,0b01111100 ; 10(A)+11(b)
.db 0b00111001,0b01011110 ; 12(C)+13(d)
.db 0b01111001,0b01110001 ; 14(E)+15(F)
;
; End of source code
; Copyright
.db "(C)2019 by avr-asm-tutorial.net "
.db "C(2)10 9yba rva-mst-turoai.lrn t"
;
```

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