AVR-Anwendungen

DCF77-Uhr mit ATtiny24
Hardware, Aufbau, Anwendung und Software für die DCF77-Uhr

DCF77-Uhr mit ATtiny24, Version 3

Die DCF77-Uhr mit dem ATtiny24 in der ersten Version hier erfreut sich seit Jahren großer Beliebtheit. Bislang (April 2019) habe ich insgesamt über 2.500 Zugriffe auf die Projektwebseite registriert, 48% davon haben den Quellcode von der Seite abgerufen.

Trotz der Popularität weist die Version aber einige Nachteile auf:

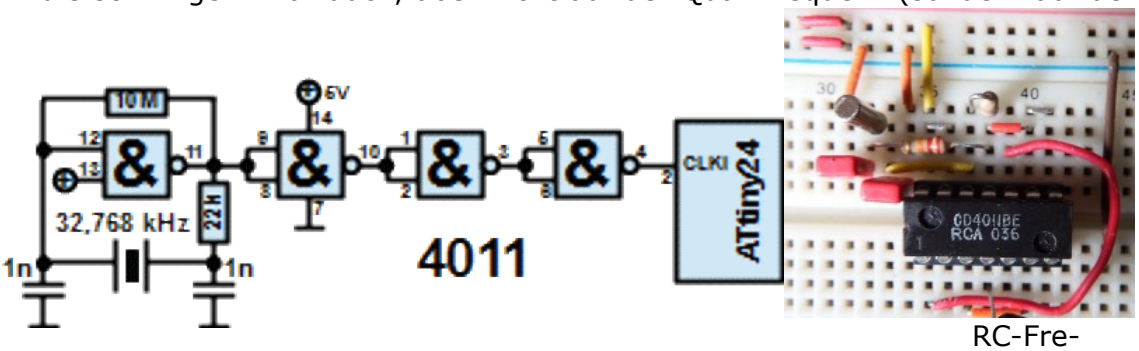
1. Sie ist auf das LCD-Format 2*24 ausgelegt. 24-spaltige LCDs gibt es aber heute gar nicht mehr zu kaufen, nur noch einige antike Exemplare sind im Ramschhandel zu finden. Ein Umbau hätte erhebliche Änderungen im Programm erforderlich gemacht, so dass eine völlig neue Konstruktion von Hard- und Software weniger aufwändig war.
2. Die Auswertung der DCF77-Signale erfolgte nicht sehr ausgefeilt. Das geht besser. Auch das ein Grund, das Programm vollständig neu zu schreiben.
3. Die Uhr hatte keinen Quarztakt, so dass sie bei längerem Betrieb ohne DCF77-Signal um 10% verkehrt ging. Das ist für eine Uhr nicht akzeptabel.
4. Die Ansteuerung der LCD war im "quick-and-dirty-Modus" erstellt und nicht mit standardisierten Methoden realisiert. Auch das geht mittlerweile besser, mit der Software hier.

Dieses Projekt hier hat diese Nachteile nicht mehr. Allerdings hat es ein paar interessante Besonderheiten:

1. Der Prozessortakt kommt aus einem 32,768 kHz-Uhrenquarz. Das kommt daher, weil der ATtiny24 NUR mit Niedrigfrequenzquarzen betrieben werden kann. Bei höheren Frequenzen muss zwingend ein externer Quarzoszillator angeschlossen werden. Und das finde ich sowas von überkandidelt, dass ich die 32 kHz Takt ausprobieren wollte (und: ja, es geht auch noch mit so niedrigem Takt eine Uhr mit DCF77-Erkennung und LCD-Ansteuerung zu betreiben).
2. Bedingt durch die niedrige Taktfrequenz gibt es zwei weitere Besonderheiten zu beachten:
    - Es gehen nur solche Programmiergeräte, die mit weniger als 8 kHz und mehr als 5 kHz ISP-Taktfrequenz programmieren können. STK500-Clone-Programmiergeräte wie das einfache Diamex können das nicht! Der nächstniedrige Takt unter 8 kHz liegt so niedrig, dass das Studio4-Programmiertool sich weigert, mit weniger als 5 kHz das Flash zu programmieren. Es gehen nur AVRISP-mkII-Clones, bei denen als ISP-Frequenz 6,48 kHz eingestellt wer-

den kann. Wer also dieses Projekt nachbauen will, braucht zwingend ein solches Programmiergerät (wie z. B. den Diamex-All-AVR) und sollte sich vorher davon vergewissern, dass er mit diesem oder einem anderen Takt zwischen 5 und 8 kHz programmieren kann.

- Mit der sehr niedrigen Taktfrequenz ist eine weitere Gefahr verbunden: hat man die defaultmäßig gesetzte CLKDIV8-Fuse nicht vorher gelöscht, dann arbeitet der Prozessor nach der Umstellung auf den externen Quarz effektiv mit 4,096 kHz Takt und kann nur noch mit einer ISP-Frequenz unter 1.024 Hz angesprochen werden. Diesen niedrigen Takt können nur ganz wenige Programmiergeräte (wie z. B. das AVRISP-mkII und seine Clones). Wer also diese Fuse versehentlich nicht gelöscht hat, kommt nur noch mit solchem niedrigen Takt an die Fuses heran. Es ist daher eine gute Idee, die CLKDIV8-Fuse in einem ersten Schritt alleine zu löschen und erst danach die Umstellung auf den externen Crystal vorzunehmen.
- Warnung! In der Praxis kam es vor, dass sich die Umstellung der Taktfrequenz-Fuse vom externen Quarz zurück auf den internen 8 MHz-RC-Oszillator nicht mehr vornehmen ließ. Bei einem ATtiny24 hat es funktioniert, bei einem anderen nicht. Es kann also sein, dass Sie die 90¢ in die Tonne treten können. Oder Sie kriegen das mit der folgenden Oszillatorschaltung wieder irgendwie hin:
  Noch ein paar Hinweise: Das geht auch mit einfachen Invertern oder NOR-Gattern (4001). Nehmen Sie aber keinen mit Schmitt-Triggern am Eingang: die schwingen zwar auch, aber nicht auf der Quarzfrequenz (sondern auf der



RC-Frequenz). Bemerkenswert an der Schaltung sind die relativ großen Kondensatoren am Quarz. Mit kleineren schwingt die Schaltung nicht an, auch nicht mit 1MΩ.

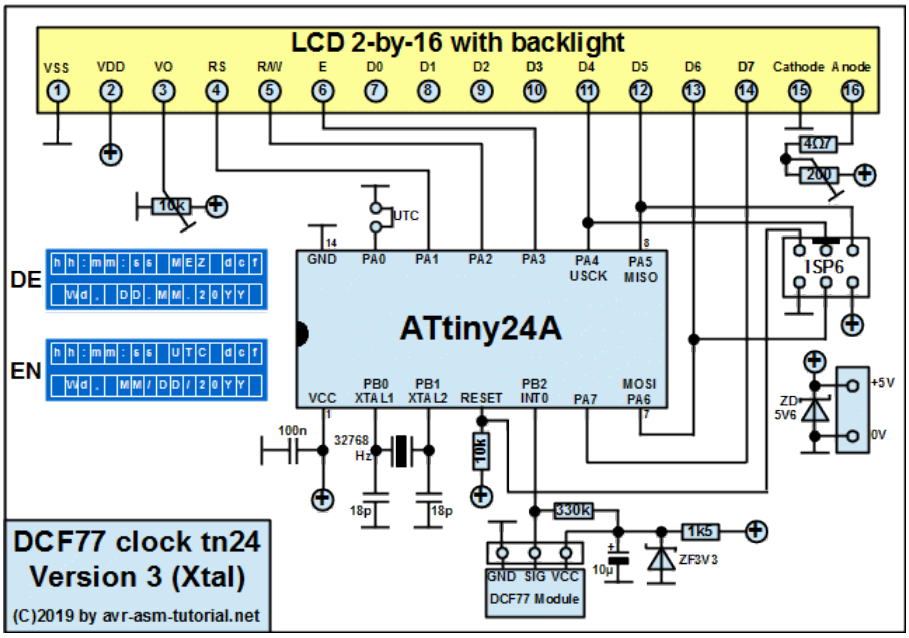
3. Da mit diesem Takt der gesamte Ablauf sehr langsam wird, war sicherzustellen, dass die gesamte Taktauswertung des DCF77-Signals, einschließlich aller Fehlerprüfungen (22 Arten von Fehlern können auftreten) und der Umwandlung der DCF77-Daten in binäre Datums- und Zeit-Formate nicht länger dauert als der kürzeste Takt von DCF77 (100 ms minus Toleranz = 80 ms). Daher waren alle Routinen zu simulieren und ihre Ausführungszeiten zu messen. Die Zeiten liegen allesamt unter 30 ms und halten daher die Anforderungen ein.
4. Mit einem Jumper lässt sich die Uhrzeit von MEZ/MESZ, wie sie DCF77 liefert, auf die Angabe der UTC-Zeit umrechnen. Dabei wird nicht einfach eine (MEZ) oder zwei (MESZ) Stunden abgezogen, da dies kurz nach Mitternacht ein falsches Datum ergäbe. Die Umrechnung von MEZ/MESZ auf UTC erfolgt bei diesem Projekt absolut korrekt, da auch das Datum und der Wochentag korrekt umgerechnet wird.
5. Ein weiteres besonderes Feature ist, dass die DCF77-Signalerkennung sowohl bei Aktiv-High- als auch bei Aktiv-Low-Signalen korrekt erfolgt, ohne das Programm umstellen zu müssen.
6. Die Verfolgung des DCF77-Erkennungsprozesses kann mit folgenden Mitteln erfolgen:
   - Fehlercodes von **E0** bis **E9** sowie von **EA** bis **EM** erscheinen jeweils auf dem

Display. E0 bis E5 betreffen Fehler, die bei der Signalauswertung erkannt wurden, E6 gibt an, dass in einer Minute nicht exakt 59 Bits empfangen wurden. Die anderen Fehler betreffen Probleme bei der Umwandlung der DCF77-Bits in das Binärformat (Paritätsfehler, Einer-Ziffern größer als Neun, Zahlen größer als im zulässigen Bereich) sowie Formatfehler (Nullbit zu Beginn oder Einsbit am Datum-Zeit-Beginn falsch). Die erscheinenden Fehlercodes werden nach einer Sekunde wieder ausgeblendet.

- Sind alle Fehlerprüfungen korrekt absolviert und wurden Datum und Uhrzeit mit DCF77 erfolgreich synchronisiert, erscheint **DCF** im Display. Nachfolgende Fehler überschreiben dies wieder.
- Falls gewünscht, können die erkannten letzten drei Bits des Datenstroms auf der Anzeige ausgegeben werden (vor dem Assemblieren **cDcfMoniBits** im Abschnitt *Adjustable constants* auf 1 setzen). Fehlermeldungen sind dabei ausgeschaltet.
- Die Anzahl empfangener Bits kann durch Setzen der Konstante **cDcfMoniBitCount** im Abschnitt *Adjustable constants* auf Eins jeweils dargestellt werden. Fehlermeldungen erscheinen nicht.

1 Hardware

1.1 Prozessor und LCD



Das ist das Schaltbild der Uhr mit der angeschlossenen LCD. Die LCD arbeitet im 4-Bit-Modus, die Anschlüsse D0 bis D3 bleiben daher offen. Der Datenbus ist an das obere Nibble des Port A des ATtiny24 angeschlossen. Die drei Steuerungsanschlüsse RS, R/W und E der LCD liegen an den Portanschlüssen PA1 bis PA3. An PA0 liegt der Jumper, der die Zeit- und Datumsausgabe von MEZ/MESZ auf UTC umstellt, wenn er geschlossen wird. Das kann jederzeit und im laufenden Betrieb der Uhr erfolgen, da Pegelwechsel laufend erkannt werden. Man kann daher statt des Jumpers auch einen Schalter verwenden.

An der LCD ist am VO-Eingang noch ein 10k-Trimmer zur Kontrasteinstellung angeschlossen. Die Intensität der Hintergrundbeleuchtung kann mit einem Trimmer von 200 Ω zwischen ca. 7 und 25 mA verstellt werden, die Helligkeitsunterschiede sind nur im Dunkeln wahrnehmbar.

An den beiden XTAL-Anschlüssen des ATtiny24 ist der Uhrenquarz mit zwei Kondensatoren von 18 pF angeschlossen. Der Reset-Eingang liegt mit 10 kΩ an Plus und ist mit dem ISP6-Anschluss verbunden, über den der Prozessor mittels USCK, MISO und MOSI in der Schaltung programmiert werden kann.

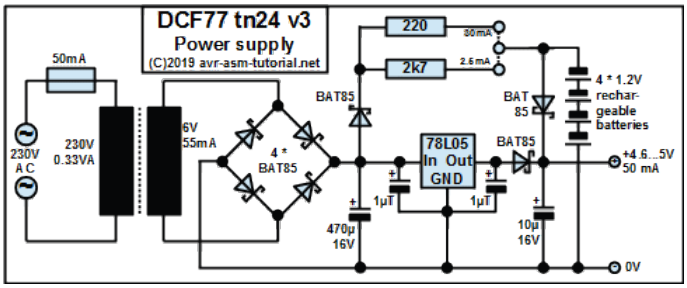
Da die verwendete LCD nur mit 4,5 bis 5,5 V betrieben werden kann, das verwendete DCF77-Modul aber nur 3,3 V verträgt, wird die Betriebsspannung des DCF77-Moduls mit einem Widerstand und einer Zenerdiode erzeugt und mit einem Elko vom Zener-Rauschen befreit. Der Signalausgang wird mit einem Widerstand von 330 kΩ auf Plus 3,3 V gezogen, damit auch Open-Collector-Module funktionieren. Das Signal wird am INT0-Eingang des

ATtiny24 (PB2) eingespeist, dessen Pull-Up nicht eingeschaltet wird.

In der Stromzufuhrleitung liegt noch eine kräftige 5,6V-Zenerdiode, damit auch beim Dauerladen des Akkus mit niedrigen Strömen keine Überspannung auftreten kann und um die Schaltung vor Verpolung der Stromzufuhr zu schützen.

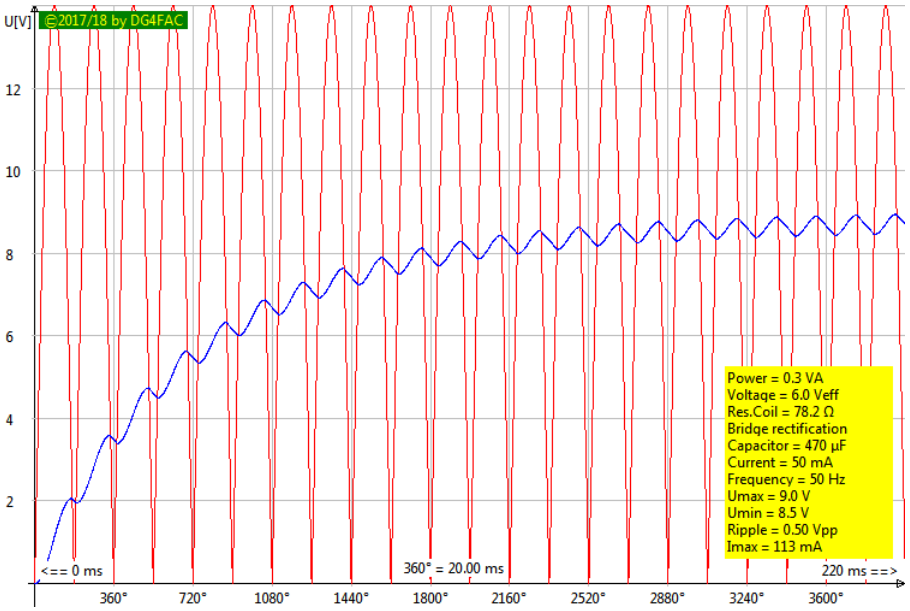
Die Ausgabe von Uhrzeit und Datum ist von der Einstellung der Konstanten **cDateFormat** in der Sektion *Adjustable Constants* der Software abhängig und kann deutsches (0) und englisches (1) Datumsformat. Natürlich betrifft das dann auch die Abkürzungen der Wochentage.

1.2 Netzteil und Akkuversorgung



Die Versorgung mit Strom erfolgt wahlweise aus einem trafobestückten Netzteil mit einem 6V/0,33VA-Trafo und geregelt mit einem 78L05, oder einem Viererpack Akkus. Alle Sorten von Akkus mit einer Nennspannung von 1,2 V können verwendet werden.

Die Akkus können bei Netzbetrieb mit einem kleinen Strom von 2,5 mA bei Laune gehalten werden. Durch Umlegen des Jumpers erfolgt die Ladung etwas schneller (bei einer Nennkapazität der Akkus von 2000 mAh in knapp drei Tagen). Es wird nicht empfohlen, beim schnelleren Laden auch die Uhr anzuschließen. Schaden tut es nicht, dafür sorgt die Zenerdiode am Spannungseingang der Uhr, die alles über 5,6 V auffrisst.



Die Umschaltung von Netz- auf Akkubetrieb und zurück erfolgt unterbrechungsfrei.

Dies hier zeigt das Netzteil unter Volllast. Die Eingangsspannung für den 78L05 reicht vollkommen aus.

Seitenanfang | Hardware | Aufbau | Software

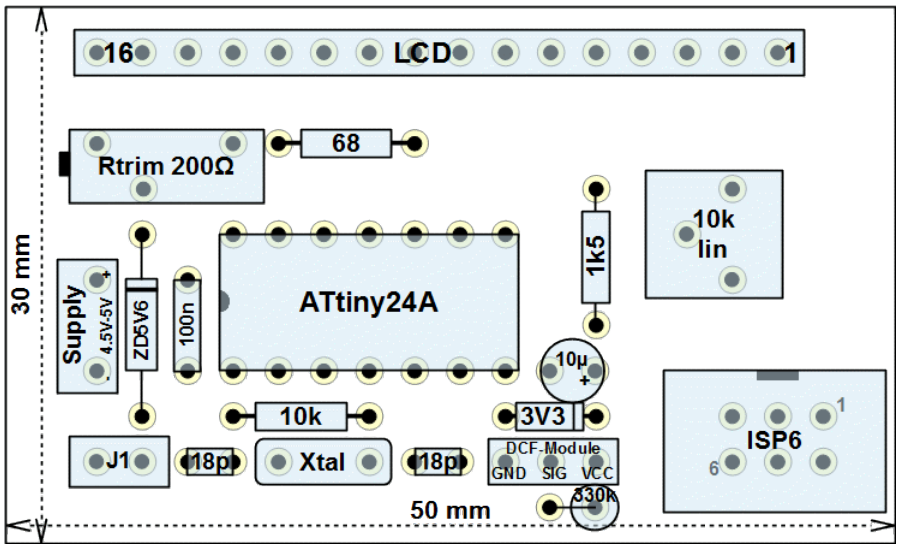
2 Aufbau

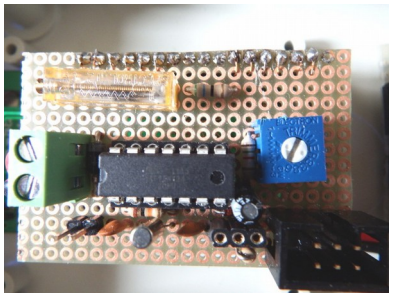
2.1 Prozessor und LCD

Der Aufbau der Prozessorplatine erfolgt auf einer doppelseitig kaschierten Lochrasterplatine mit 30x50 mm Größe. Die Verwendung des 10-Gang-Trimmers für die Hintergrundbeleuchtungsstärke ist etwas überkandidelt, es war aber sonst kein einfacherer 200Ω-Trimmer zu kriegen. Trimmer und Widerstand kann man auch durch einen Festwiderstand zwischen 68 und 270Ω ersetzen, der Helligkeitsunterschied ist sowieso nur akademisch und

nachts.

Da die Platine huckepack auf der LCD sitzt (Rücken an Rücken) wird die LCD mit einer 16-poligen Stiftleiste nach unten bestückt. Das Gegenstück dazu sitzt auf der Lötseite der Prozessorplatine und wird von der Oberseite her eingelötet.

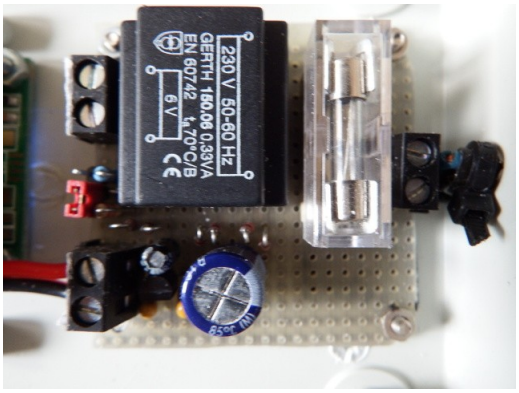




Wer ein anderes als das hier verwendete DCF77-Modul verwenden will, braucht vielleicht eine vierpolige Buchse dafür.

2.2 Netzteil und Akkuversorgung

Das Netzteil und die Akkuversorgung werden ebenfalls auf einer kleinen Lochrasterplatine aufgebaut.



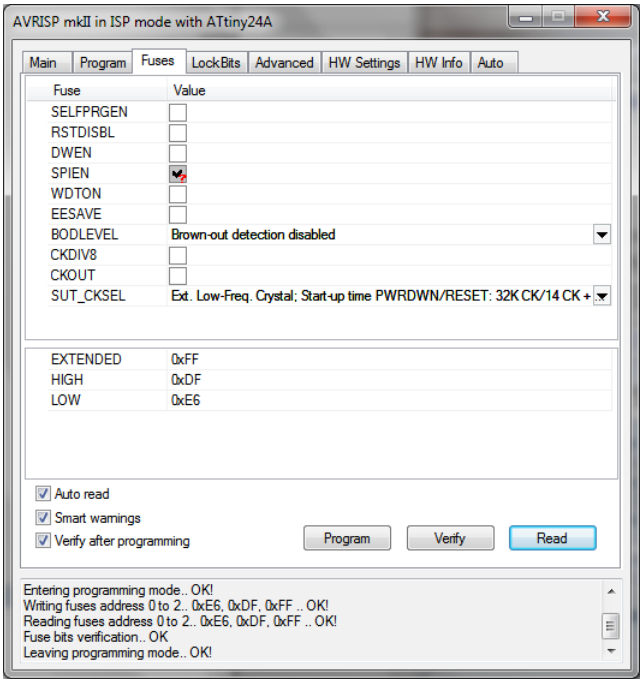
Seitenanfang	Hardware	Aufbau	Software

3 Software

3.1 Downloads

Den Assembler-Quellcode gibt es in englisch hier zum Download sowie hier im Anhang. Zum Assemblieren ist noch die Include-Routine hier notwendig.

Alle nötigen Berechnungen zu diesem Projekt finden sich in den Rechenblättern im OpenOffice-Dokument hier.



3.2 Fuses

So werden die Fuses des ATtiny24 gesetzt. Bitte unbedingt die Warnungen zu Beginn dieser Seite beachten und die CLKDIV8-Fuse zuallererst löschen. Ich übernehme jedenfalls keinerlei Garantie dafür, dass der Rückweg zum internen RC-Oszillator danach auch noch funktioniert.

3.2 Einstellungen von Eigenschaften

In der Sektion *Adjustable constants* im Quellcode lassen sich folgende Eigenschaften der Software verstellen:

- **crystal**: Die Taktfrequenz des externen Quarzes. Bitte beachten, dass der ATtiny24 nur niedrige Quarzfrequenzen zulässt. Es macht daher keinen Sinn, hier mehr als 100 kHz anzugeben.
- Die Dauer der DCF77-Signale lässt sich mit folgenden Parametern einstellen (alle Größen in Millisekunden).
    1. **Spur**: Mindestdauer von Signalen am INT0-Eingang,
    2. **Zero**: Dauer eines Null-Bits,
    3. **One**: Dauer eines Eins-Bits,
    4. **Pause**: Dauer der Pause zwischen zwei Null-/Eins-Bits,
    5. **Minute**: Dauer zwischen dem letzten übertragenen Bits einer Minute und Beginn des ersten Bits der neuen Minute.

  Mit der Konstanten **MySet** lassen sich zwei Konstantensets umschalten: der Default-Set und eine indivuell angepasste Version, wie sie z. B. aus eigenen Messungen mit **DebugDcfDur** resultieren (siehe weiter unten).
- **TimeOut**: Die Dauer innerhalb derer eine INT0-Flanke am INT0-Eingang erwartet wird, trifft keine Flanke ein, erscheint eine Fehlermeldung,
- **cDcfTol**: Die Toleranzbreite der DCF77-Signale in Prozent (mit dem Standardset zwischen 6 und 32%),
- die Darstellung von DCF-Bits auf der LCD anstelle von Fehlercodes:
    1. **cDcfMoniBits**: Stelle die letzten drei empfangenen Bits binär dar,
    2. **cDcfMoniBitCount**: Stelle den Bitzähler der empfangenen Bits dar.

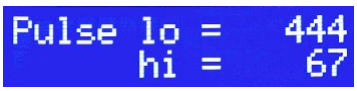
  Es macht keinen Sinn, beide Schalter gleichzeitig einzuschalten.
- **cDateFormat**: deutsches (0) oder englisches (1) Format,
- **cTxtDelay**: Dauer der Pause zwischen Eingangsanzeige und Start der Uhr, in Vielfachen von 50 ms.

Änderungen werden erst wirksam, wenn der Quellcode assembliert und in das Flash übertragen ist.

3.3 Verfügbare Fehlerdiagnosen

Im Kopf des Quellcodes lassen sich folgende Debug-Optionen einschalten:

- **SkipLcd**: Alle Schreibvorgänge und Verzögerungsroutinen der LCD werden ausgeblendet. Das macht Sinn, wenn Durchläufe im Simulator getestet werden sollen.
- **DebugDcfPulse**: Simuliert einen DCF77-Impuls am INT0-Eingang. Die Dauer des Impulses kann in der Konstanten **DebugDcfPulseTestTime** in Millisekunden vorgegeben werden. Handelt es sich um einen Minutenwechsel (850 - Toleranz bis 850 + Toleranz), erfolgt anschließend die Umwandlung der in der Tabelle **DcfTestTable** vorgegebenen DCF-Bits in rDcf0 bis rDcf7 in Zeit-/Datums-Formate.
- **DebugDcfCode**: Testet die Routine zur Umwandlung der in Tabelle **DcfCodeTable** vorgegebenen DCF77-Bits in rDcf0 bis rDcf7 in Zeit- und Datumsformat.
- **DebugPulseDur**: Zeigt Aktiv-Low- und Aktiv-High-Impulsdauern auf der LCD an. Angezeigt werden die aktiven TC1-Timer-Ticks.



  Die angezeigten Pulsdauern müssen mit 1,953 multipliziert werden, um Millisekunden zu erhalten. Zappelt die Anzeige ist das ein untrügliches Anzeichen für eine falsche Ausrichtung der DCF77-Antenne oder eine Störung des Empfangs durch einen massiven HF-Störer (Schaltnetzteile, Energiesparlampen, etc.).

Bei den beiden Fehlerdiagnosen mit **DebugDcfPulse** und **DebugDcfCode** können übrigens beliebige Testtabellen im Rechenblatt *DCF77_pattern* des OpenOffice-Dokuments hier aus Zeit- und Datumseinstellungen erzeugt und in den Quellcode kopiert werden, deren Dekodierung in der nachfolgenden Simulation dann erfolgen soll.

3.4 Erweiterungen des Programms

Falls Du vorhast, über geänderte Einstellungen hinaus das Programm zu ändern, bitte Folgendes beachten:

1. Das Programm belegt in seiner derzeitigen Version einen sehr großen Teil des verfügbaren Flashspeichers im ATtiny24. Für längere Einfügungen ist daher kaum noch Platz. Es muss dann auf den ATtiny44 oder ATtiny84 umgestiegen werden. Bitte beachten, dass beim ATtiny84 zusätzlich das High-Byte des Stackpointers gesetzt wird (erfolgt automatisch wenn der Port **SPH** vorhanden ist).
2. Bei allen Erweiterungen ist sicherzustellen, dass diese die 80/100-ms-Ausführungsgrenze einhalten. Bei länger andauernden Behinderungen von Interrupt-Flaggenbehandlungen kann es zu unerwarteten Nebeneffekten kommen.

3.5 Ablaufsteuerung

Nachfolgend werden einige Details zur Ablaufsteuerung des Programmes beschrieben, die bei Änderungen zu beachten sind.

3.5.1 Sekundenimpulse

Die Sekundenimpulse, die die Uhrzeit und ggfs. das Datum um eine Sekunde erhöhen, werden mittels des Timers TC0 erzeugt. Er arbeitet mit einem Vorteiler von 64 und wird daher aus der Taktfrequenz von 32,768 kHz mit 512 Hz getaktet. Der 8-Bit-Timer läuft bei 256 alle 0,5 Sekunden über und erzeugt einen Overflow-Interrupt. In der Interrupt-Service-Routine wird ein Zählregister abwärts gezählt. Beim Erreichen von Null wird die Flagge **bSec** gesetzt und der Zähler mit Zwei neu gestartet.

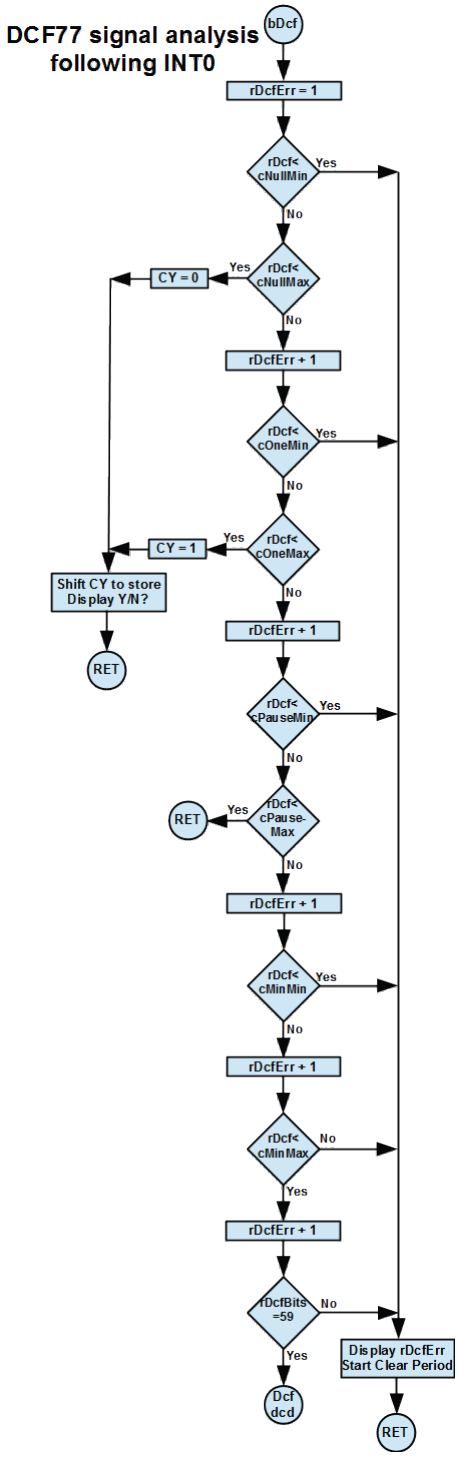
Außerhalb der Interrupt-Service-Routine werden die Sekunden (im SRAM unter Adresse 0x0060) um Eins erhöht und auf der LCD in Zeile 1 in den Spalten 7 und 8 mit führenden Nullen ausgegeben. Erreicht der Sekundenzähler 60, beginnt es wieder bei Null und die Minuten werden erhöht. Nun werden sowohl die Minuten (Zeile 1, Spalten 4 und 5) als auch die Sekunden auf die LCD geschrieben.

Analog dazu

- werden die Stunden erhöht, wenn die Minuten 60 erreichen,
- wird der Tag und der Wochentag erhöht, wenn die Stunden 24 erreichen,
- wird der Monat erhöht, wenn der Tag die Monatstage überschreitet, und
- wird das Jahr erhöht, wenn der Monat 12 überschreitet.

Es werden immer nur diejenigen Stellen der Uhr aktualisiert, deren Aktualisierung angezeigt war.

Die Uhr läuft daher quarzgenau vollkommen eigenständig, also auch ohne DCF77-Synchronisation, und muss nur beim ersten Programmieren auf die korrekte Uhrzeit und das korrekte Datum eingestellt werden (Tabelle **InitDTTable:** im Quellcode). Bei jedem Reset (z. B. beim Unterbrechen der Stromversorgung) stellt sich die Uhr wieder auf diese Voreinstellung zurück, so dass dies nur dann Sinn macht, wenn eine Batterie oder ein Akku unterbrechungsfrei angeschlossen ist. Beim Einstellen der Sekunden ist zu beachten, dass die Flashprogrammierung bei ca. 6,5 kHz ISP-Frequenz etwa 39 Sekunden dauert, und es bei eingeschaltetem Verifizieren doppelt so lange dauert. Hinzu zu addieren ist diejenige Verzögerungszeit, die zwischen der Eröffnung und dem Loslaufen der Uhr eingestellt ist (default: 5 Sekunden), so dass es bis zu 83 Sekunden dauert, bis die Uhr nach dem Start des Flash-Schreibens dann effektiv losläuft.



3.5.2 DCF77-Signalauswertung

Die DCF77-Signalauswertung erfolgt folgendermaßen:

1. Das DCF77-Empfangsmodul ist an den Eingang INT0 (PB2) angeschlossen. Bei jeder Flanke an diesem Eingang erfolgt ein INT0-Interrupt.
2. In der Interrupt-Service-Routine wird der Zählerstand von TC1 ausgelesen, die Flagge **bDcf** gesetzt, der Zähler mit Null neu gestartet und der abgelesene Zählerstand in das Registerpaar **rDcfH:rDcfL** kopiert. War der Zählerstand allerdings kleiner als die Konstante **cSpur** (im Quellcode auf 5 ms eingestellt), dann unterbleibt dies und INT0 wartet auf die nächste eintreffende Flanke.
3. Der Zähler TC1 arbeitet mit einem Vorteiler von 64, so dass er alle 64/32.768 = 1,95 ms um Eins erhöht wird. Eine von DCF77 gesendete Null (100 ms) entspricht dabei einem Zählerstand von etwa 51, eine Eins ca. 102 und ein Minutenwechsel 923 oder 974, je nachdem ob das letzte gesendete Zeichen eine Null oder eine Eins war.
4. Treffen für 5 Sekunden lang gar keine Pegelwechsel ein, wird TC1 auf Null gesetzt (CTC-Modus) und der Fehlercode **E0** auf der LCD ausgegeben (Zeit einstellbar mit der Konstanten **TimeOut** im Quelltext).
5. Bedingt durch die Reaktion auf ALLE Pegelwechsel ist es egal, ob Aktiv-High- oder -Low-Signale vorliegen, sie werden alle korrekt gezählt. Da auf den Flankenwechsel eines korrekten Null- oder Eins-Bits immer auch ein zweiter Flankenwechsel erfolgt, wenn das nächste Bit beginnt, werden diese Flankenwechsel aufgrund ihrer Dauer (800 bis 900 ms) erkannt und ignoriert.
6. Für die Dauer aller Signale lässt sich eine Toleranz einstellen (Konstante **cDcfTol** in Prozent). Diese sollte nicht zu niedrig (weniger als 5%) und nicht zu hoch (mehr als 30%) eingestellt werden. Überlappen sich Erkennungsbereiche, wird dies beim Assemblieren als Fehler eingestuft und mit einer Fehlermeldung abgefangen.

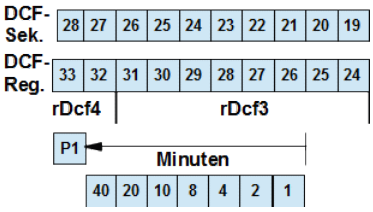
Allen Signaldauern, die kürzer als für eine Null erforderlich sind oder die zwischen den Bereichen für eine Null, eine Eins, eine Pause oder einem Minutenwechsel liegen oder die länger als ein Minutenwechsel dauern, sind fünf Fehlermeldungen zugeordnet, die als **E1** bis **E5** auf der LCD angezeigt werden. Liegen beim Minutenwechsel nicht genau 59 korrekt empfangene Bits vor, wird **E6** angezeigt.

Alle empfangenen Bits werden in acht Registern (**rDcfn** mit n=0 bis 7) aufbewahrt. Sie werden jeweils von rDcf7, beginnend mit dessen Bit 7, wie in einem Schieberegister von links nach rechts eingeschoben, so dass das vor dem Minutenwechsel zuletzt gesendete Bit in Bit 7 von rDcf7 liegt. Die Lage aller Bits bei dieser Auswertemethode ist hier im Detail dargestellt.

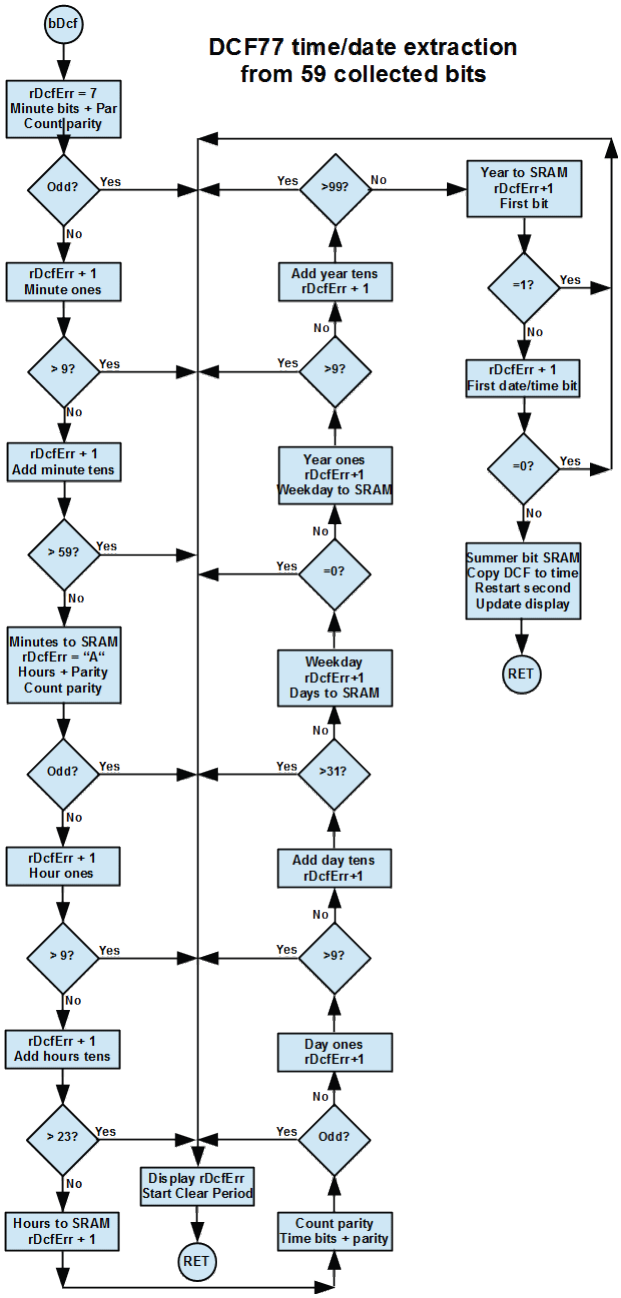
3.6 DCF77-Zeit- und Datumsextraktion

Die in rDcf0 bis rDcf7 gesammelten DCF77-Bits werden, wenn genau 59 fehlerfrei gesammelt sind, ausgewertet und in Datums- und Zeitwerte umgewandelt.

Um zu demonstrieren, wie das gemacht wird, hier das Vorgehen bei der Extraktion der Minuten aus dem Datenstrom. Die Bits der DCF77-Sekunden (Beginn der Zählung mit 0) sind in den Registern rDcf3 (Minuten-Einer: Bit 2, bis Minuten-Zwanziger: Bit 7) und rDcf4 (Minuten-Vierziger: Bit 0, Minutenparität: Bit 1) versammelt. Um die Minutenbits vollständig in ein Register zu bekommen, werden



1. die Register rDcf3 und rDcf4 in ein anderes Registerpaar kopiert (z. B. nach ZH:ZL), was mit MOV geht,
2. das höhere Byte einmal mit LSR rechts geschoben, wobei Bit 0 von ehemals rDcf4 in das Carry-Bit des Statusregisters gelangt, und von dort
3. mit ROR in das untere Byte in ZL von links her eingeschoben wird,
4. nach erneuter Wiederholung von LSR und ROR alle acht Bits der Minuten einschließlich des Paritätsbits in ZL versammelt.



Dann muss die Paritätsprüfung erfolgen. Dazu muss die Anzahl an Einsen in dem Byte geradzahlig sein (even parity). Da die CPU der AVR keine Paritätsflagge kennt, müssen die Anzahl Einsen in einem Zählregister gezählt und mit ANDI Register,1 und anschließendem bedingten Sprung BREQ oder BRNE, oder auch mit SBRC Register,0, entschieden werden, ob die Parität korrekt eine Null ergibt.

Ist das der Fall, können die Einer des Minutenbytes, nach Löschen des Paritätsbits und der

Zehnerbits, z. B. mit ANDI ZL,0x0F, daraufhin überprüft werden, ob sie mehr als Neun ergeben. Wenn ja, liegt ein Fehler vor. Hat man vor dem Löschen der Zehner diese nach ZH kopiert, können nun die Zehner, Zwanziger und Vierziger hinzuaddiert werden, z. B. mit SBRC ZH,4 und subi ZL,-10 und analog mit den beiden höherwertigen Bits die Zwanziger und Vierziger.

Danach liegen die Minuten im Binärformat vor und können darauf hin überprüft werden, ob sie größer als 59 sind. Ist das nicht der Fall, sind die Minuten korrekt und können zwischengespeichert werden. Das Zwischenspeichern empfiehlt sich, weil bei der weiteren Umwandlung von Stunden und des Datums weitere Fehler auftreten könnten, so dass die ausgewertete DCF77-Zeit nicht korrekt ist und nicht in die Uhr übernommen werden sollte.

Beim Wochentag ist noch zu beachten, dass DCF77 den Montag mit 1 beginnt, in dieser Uhr aber mit Null gespeichert wird.

In gleicher Weise werden alle weiteren benötigten Datms- und Zeitinformationen zusammengestellt. Am Ende wird noch überprüft, ob das allererste DCF77-Bit eine Null war und ob das erste Zeit- und Datumsbit eine Eins war. Falls bei allen Prüfungen keine Fehler auftraten, können die im Zwischenspeicher gesammelten Auswertungen in die Uhr übertragen werden. Die Sekunden werden auf Null gesetzt. Da mit der Auswertung bis hierhin eine Zeitverzögerung von etwa 30 ms auftrat, wird auch die TC0-Uhr auf einen entsprechenden Wert gesetzt. Wichtig ist noch das Löschen der bSec-Flagge, die in der TC0-OVF-Interrupt-Service-Routine bereits gesetzt sein könnte und bewirken würde, dass die Uhr um eine Sekunde vorgeht.

3.7 LCD-Steuerung

Die LCD-Ansteuerung wurde mit den Include-Routinen in lcd.inc realisiert. Diese enthält alle Routinen

- zur Initialisierung der LCD (**LcdInit**),
- zum Setzen der Ausgabeposition in Zeile (ZH) und Spalte (ZL), **LcdPos**,
- zur Ausgabe von im Flash in Tabellen gespeicherten Texten (**LcdText**),
- zur Ausgabe einzelner Zeichen (**LcdChar**),
- zur Ausgabe von zweistelligen Dezimalzahlen (**LcdDec2**), und
- für Zeitverzögerungen von 50 ms.

Alle Einstellungen, die dazu nötig sind, wie Ansteuerungsports (Datenport, Kontrollpins), Ansteuerungsarten (mit Busy-Abfrage über den oberen Datenport), Anzahl Zeilen und Spalten der LCD (LcdLines, LcdCols), u.v.a.m., sind im Abschnitt "LCD Configuration" im Quelltext vorgenommen. Darüber können bequem Änderungen des LCD-Typs vorgenommen werden.

Ein großer Vorteil dieser lcd.inc ist, dass mit umfangreichen .IF-Einstellungen kein unnützer Code im Flash landet sondern wirklich nur das, was tatsächlich auch benötigt wird. Nur so war es möglich, mit dem im ATtiny24 verfügbaren Flashspeicher auszukommen.

Seitenanfang	Hardware	Aufbau	Software

Pfad: Home => AVR-DE => Anwendungen => DCF77_tn24_v3 => Assembler-Quellcode

DCF77-Uhr mit ATtiny24
Assembler-Quellcode für die DCF77-Uhr

Assembler-Quellcode für die DCF77-Uhr

Der Assembler-Code ist hier, es wird noch die LCD Include hier benötigt.

```
;
; ***********************************
; * DCF77 receiver ATtiny24 V3      *
; * with external crystal 32768 Hz  *
; * (C)2019 by avr-asm-tutorial.net *
; ***********************************
;
.nolist
.include "tn24adef.inc" ; Define device ATtiny24A
.list
;
; ***********************************
;    D E B U G  S W I T C H E S
; ***********************************
;
.equ Yes = 1
.equ No = 0
;
.equ SkipLcd = No ; Skip all LCD routines
;
; Test the DCF77 pulse calculation routine
.equ DebugDcfPulse = No ; Debug the DCF pulse code
;
; Test the DCF77 conversion routine
.equ DebugDcfCode = No ; Debug the DCF code
;
; Test the INT0 pulse durations
;   measures pulse durations and displays TC1 ticks
.equ DebugPulseDur = No ; Debug pulse durations
;
; ***********************************
;        H A R D W A R E
; ***********************************
;     ;
; Device: ATtiny24A, Package: 14-pin-PDIP_SOIC
;
;                 _________
;           1 /           |14
;    + 5V o--|VCC     GND|--o 0 V
;   XTAL1 o--|PB0     PA0|--o J1-UTC
;   XTAL2 o--|PB1     PA1|--o LCD-RS
;   RESET o--|RESET   PA2|--o LCD-R/W
;   DCF77 o--|PB2     PA3|--o LCD-E
; LCD-D7 o--|PA7     PA4|--o LCD-D4
; LCD-D6 o--|PA6     PA5|--o LCD-D5
;          7 |___________|8
```

```
;

; *********************************
;  P O R T S   A N D   P I N S
; *********************************
;
; Jumper 1 ports and pins
.equ pJ1O = PORTA ; Jumper 1 output port
.equ bJ1O = PORTA0 ; Jumper 1 output portpin
.equ pJ1D = DDRA ; Jumper 1 direction port
.equ bJ1D = DDA0 ; Jumper 1 direction portpin
.equ pJ1I = PINA ; Jumper 1 input port
.equ bJ1I = PINA0 ; Jumper 1 input portpin
;
; DCF77 ports and pins
.equ pDcfO = PORTB ; DCF77 output port
.equ bDcfO = PORTB2 ; DCF77 output portpin
.equ pDcfD = DDRB ; DCF77 direction port
.equ bDcfD = DDB2 ; DCF77 direction portpin
.equ pDcfI = PINB ; DCF77 input port
.equ bDcfI = PINB2 ; DCF77 input portpin
;
; (Ports of the LCD see LCD configuration section)
;
; *********************************
;   A D J U S T A B L E   C O N S T
; *********************************
;
.equ crystal = 32768 ; Crystal frequency
;
; DCF77 signal durations, all in ms
.equ MySet = Yes ; Use individually measured set
.if MySet == No
  ; The standard set
  .equ Spur = 5 ; Spurious signal
  .equ Zero = 100 ; Binary zero
  .equ One = 200 ; Binary one
  .equ Pause = 1000-(Zero+One)/2 ; Pause between bit and next second pulse
  .else
  ; The individual set
  .equ Spur = 10 ; Longer spurious signal
  .equ Zero = 130 ; Longer zero
  .equ One = 225 ; Longer one
  .equ Pause = 825 ; Shorter pause
  .endif
.equ Minute = 2000-(Zero+One)/2 ; Minute pulse
.equ TimeOut = 5000 ; Signal time out
;
; DCF77 signal tolerance
.equ cDcfTol = 25 ; Tolerance in percent
;
; Monitor incoming DCF77 signals
;   Select only one of those as they write on the
;   same space on the LCD
.equ cDcfMoniBits = No ; Monitor the incoming bits
.equ cDcfMoniBitCount = No ; Monitor the bit count
;
; Date format selection
.equ cDateFormat = 0 ; 0=DE, 1=EN
;
; Delay between init display and mask display
```

```
.equ cTxtDelay = 100 ; Delay start-up, multiples of 50 ms
;
; *********************************
;  F I X  &  D E R I V.  C O N S T
; *********************************
;
; TC0 as seconds pulse generator
.equ cTc0Presc = 64 ; Prescaler value
.equ cTc0Div = 256 ; Overflow
.equ cDivSec = crystal / cTc0Presc / cTc0Div
;
; TC1 as DCF77 signal duration counter
.equ cTc1Presc = 64 ; Prescaler value
.equ cTc1Count = crystal / cTc1Presc ; Counter frequency
.equ cTc1T = 1000000 / cTc1Count ; Time per Tick in us
;  Default is: 1,953 us/tick
;
; DCF77 signal durations as TC1 counts, with tolerances
.equ cSpur=(1000*Spur)/cTc1T + 1 ; Spurious signal duration count
.equ cZeroMin=((Zero-Zero*cDcfTol/100)*1000)/cTc1T ; Minimum zero
.equ cZeroMax=((Zero+Zero*cDcfTol/100)*1000)/cTc1T + 1 ; Maximum zero
.equ cOneMin=((One-One*cDcfTol/100)*1000)/cTc1T ; Minimum one
.equ cOneMax=((One+One*cDcfTol/100)*1000)/cTc1T+1 ; Maximum one
.equ cPauseMin=((Pause-Pause*cDcfTol/100)*1000)/cTc1T ; Minimum pause
.equ cPauseMax=((Pause+Pause*cDcfTol/100)*1000)/cTc1T+1 ; Maximum pause
.equ cMinuteMin=((Minute-Minute*cDcfTol/100)*1000)/cTc1T ; Minimum minute
.equ cMinuteMax=((Minute+Minute*cDcfTol/100)*1000)/cTc1T+1 ; Maximum minute
.equ cTimeOut=(1000*TimeOut)/cTc1T ; Timeout value
;
; Error checking
.if cZeroMin <= cSpur
  .error "Zero min less or equal spurious!"
  .endif
.if cOneMin <= cZeroMax
  .error "One min less or equal zero max!"
  .endif
.if cPauseMin <= cOneMax
  .error "Pause min less or equal one max!"
  .endif
.if cMinuteMin <= cPauseMax
  .error "Minute min less or equal pause max!"
  .endif
.if cTimeOut <= cMinuteMax
  .error "Time out less or equal minute max!"
  .endif
;
; *********************************
;  L C D   C O N F I G U R A T I O N
; *********************************
;
; Specific parameter set of properties/definitions
.equ clock = crystal ; Clock frequency of controller in Hz
; LCD size:
  .equ LcdLines = 2 ; Number of lines (1, 2, 4)
  .equ LcdCols = 16 ; Number of characters per line (8..24)
; LCD bus interface
  .equ LcdBits = 4 ; Bus size (4 or 8)
  ; If 4 bit bus:
    .equ Lcd4High = 1 ; Bus nibble (1=Upper, 0=Lower)
  .equ LcdWait = 0 ; Access mode (0 with busy, 1 with delay loops)
; LCD data ports
```

```
  .equ pLcdDO = PORTA ; Data output port
  .equ pLcdDD = DDRA ; Data direction port
; LCD control ports und pins
  .equ pLcdCEO = PORTA ; Control E output port
  .equ bLcdCEO = PORTA3 ; Control E output portpin
  .equ pLcdCED = DDRA ; Control E direction port
  .equ bLcdCED = DDA3 ; Control E direction portpin
  .equ pLcdCRSO = PORTA ; Control RS output port
  .equ bLcdCRSO = PORTA1 ; Control RS output portpin
  .equ pLcdCRSD = DDRA ; Control RS direction port
  .equ bLcdCRSD = DDA1 ; Control RS direction portpin
; If LcdWait = 0:
  .equ pLcdDI = PINA ; Data input port
  .equ pLcdCRWO = PORTA ; Control RW output port
  .equ bLcdCRWO = PORTA2 ; Control RW output portpin
  .equ pLcdCRWD = DDRA ; Control RW direction port
  .equ bLcdCRWD = DDA2 ; Control RW direction portpin
; If you need binary to decimal conversion:
  .equ LcdDecimal = 1 ; If defined: include those routines
; If you need binary to hexadecimal conversion:
  ;.equ LcdHex = 1 ; If defined: include those routines
; If simulation in the SRAM is desired:
  ;.equ avr_sim = 1 ; 1=Simulate, 0 or undefined=Do not simulate
;
; *********************************
;       R E G I S T E R S
; *********************************
;
.def rDcf0 = R0 ; DCF bits shift register 0
.def rDcf1 = R1 ; dto., 1
.def rDcf2 = R2 ; dto., 2
.def rDcf3 = R3 ; dto., 3
.def rDcf4 = R4 ; dto., 4
.def rDcf5 = R5 ; dto., 5
.def rDcf6 = R6 ; dto., 6
.def rDcf7 = R7 ; dto., 7
; free: R8
.def rDcfBits = R9 ; DCF bits counter
.def rDcfErr = R10 ; DCF signal error
.def rTc1L = R11 ; TC1 count, LSB
.def rTc1H = R12 ; dto., MSB
.def rDcfL = R13 ; DCF duration, LSB
.def rDcfH = R14 ; dto., MSB
.def rSreg = R15 ; Save/Restore status port
.def rmp = R16 ; Define multipurpose register
.def rimp = R17 ; Multipurpose inside ints
.def rFlag = R18 ; Flag register
  .equ bSec = 0 ; Seconds flag
  .equ bDcf = 1 ; DCF77 pulse flag
  .equ bDcfTO = 2 ; DCF77 missing signal
  .equ bDcfErr = 3 ; DCF77 error flag
  .equ bDcfOk = 4 ; DCF77 conversion ok
  .equ bUtcChg = 4 ; UTC change
  .equ bUtc = 5 ; UTC flag
.def rDivSec = R19 ; Seconds divider
; free: R20 to R25
; used: R27:R26 = X as pointer outside ints
; used: R29:R28 = Y as output pointer outside ints
; used: R31:R30 = Z for diverse purposes outside ints
;
; *********************************
```

```
;           S R A M
; *********************************
;
.dseg
.org SRAM_START
sMet: ; MET = ss_mm_hh_wd_DD_MM_YY_ST (ST=Summer time)
.byte 8 ; 8 bytes for base time (middle european time)
sUtc: ; UTC = hh_wd_DD_MM_YY
.byte 5 ; 5 bytes for universial time coordinated
sDcf: ; DCF77 time = mm_hh_wd_DD_MM_YY
.byte 7 ; 7 bytes for received DCF77 time
; Displacements relative to sMet:
  .equ dSec = 0
  .equ dMin = 1
  .equ dHour = 2
  .equ dWd = 3
  .equ dDay = 4
  .equ dMonth = 5
  .equ dYear = 6
  .equ dSummer = 7
  .equ dUtcHour = 8
  .equ dUtcWd = 9
  .equ dUtcDay = 10
  .equ dUtcMonth = 11
  .equ dUtcYear = 12
  .equ dDcfMin = 13
  .equ dDcfHour = 14
  .equ dDcfWd = 15
  .equ dDcfDay = 16
  .equ dDcfMonth = 17
  .equ dDcfYear = 18
  .equ dDcfSummer = 19
;
; *********************************
;   D C F   E R R O R   C O D E S
; *********************************
;
; Error codes that occur on the end of line 1 of the LCD
  ; E0: Time-out on the DCF input pin
  ; E1: Signal too short
  ; E2: Signal between zero and one
  ; E3: Signal between one and pause
  ; E4: Signal between a pause and a minute
  ; E5: Longer than minute
  ; (E6): Not 59 bits, displays number of bits decimal
  ;     instead of error code
  ; E7: Parity minutes odd
  ; E8: Minute ones larger than 9
  ; E9: Minutes larger than 60
  ; EA: Hours parity odd
  ; EB: Hour ones larger than 9
  ; EC: Hours larger than 23
  ; ED: Date parity odd
  ; EE: Day ones larger than nine
  ; EF: Day larger than 31
  ; EG: Weekday is zero
  ; EH: Month ones larger than 9
  ; EI: Month ones larger than 12
  ; EJ: Year ones larger than ten
  ; EK: Year larger than 99
  ; EL: First bit is not zero
```

```
  ; EM: Date/Time start bit not a one
;
; *********************************
;         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
        rjmp Int0Isr ; EXT_INT0
        rjmp Pci0Isr ; PCI0
        reti ; PCI1
        reti ; WATCHDOG
        reti ; ICP1
        rjmp Oc1AIsr ; OC1A
        reti ; OC1B
        reti ; OVF1
        reti ; OC0A
        reti ; OC0B
        rjmp Ovf0Isr ; OVF0
        reti ; ACI
        reti ; ADCC
        reti ; ERDY
        reti ; USI_STR
        reti ; USI_OVF
;
; *********************************
;  I N T - S E R V I C E   R O U T .
; *********************************
;
; INT0 interrupt service routine
;   called by: DCF77 input pin on level change
;   read TC1 count and
;     if spurious length: continue
;     if not: copy counter, set bDcf flag, clear counter
Int0Isr: ; 7 clock cycles for int and vector jump
  in rSreg,SREG ; Save SREG, +1=8
  in rTc1L,TCNT1L ; Read LSB counter, +1=9
  in rTc1H,TCNT1H ; dto., MSB, +1=10
  ldi rimp,Low(cSpur) ; LSB of spurious count, +1=11
  cp rTc1L,rimp ; Count smaller spurious? +1=12
  ldi rimp,High(cSpur) ; MSB of spurious count, +1=13
  cpc rTc1H,rimp ; Count smaller spurious, +1=14
  brcs Int0Isr1 ; Spurious, ignore, +1/2=15/16
  clr rimp ; Restart counter, +1=16
  out TCNT1H,rimp ; Clear TC1 MSB, +1=17
  out TCNT1L,rimp ; dto., LSB, +1=18
  mov rDcfL,rTc1L ; Copy count, LSB, +1=19
  mov rDcfH,rTc1H ; dto., MSB, +1=20
  sbr rFlag,1<<bDcf ; Set DCF flag, +1=21
Int0Isr1: ; 16/21 clock cycles
  out SREG,rSreg ; Restore SREG, +1=17/22
  reti ; +4=21/26
  ; 21 or 26 clock cycles = 0.64 ms resp. 0.79 ms
  ; Restart counter delay = 18 clock cycles = 0.55 ms
;
; Jumper input has changed
```

```
Pci0Isr:
  in rSreg,SREG ; Save SREG
  sbr rFlag,1<<bUtcChg ; Set change flag
  out SREG,rSreg ; Restore flag
  reti
;
; OC1A interrupt service routine
;   called by overflow of the DCF77 duration counter
;   set bDcfTO flag
Oc1AIsr:
  in rSreg,SREG ; Save SREG
  sbr rFlag,1<<bDcfTO ; Set DCF signal timeout flag
  out SREG,rSreg ; Restore SREG
  reti
;
; TC0 overflow interrupt service routine
;   called by CTC event every 500 ms
;   downcount the second divider
;   if zero: set seconds flag and restart second divider
Ovf0Isr: ; 7 clock cycles int plus vector jump
  in rSreg,SREG ; Save SREG, +1=8
  dec rDivSec ; Count divider down, +1=9
  brne Ovf0Isr1 ; Not at zero, +1/2=10/11
  sbr rFlag,1<<bSec ; Set seconds flag, +1=11
  ldi rDivSec,cDivSec ; Restart divider, +1=12
Ovf0Isr1: ; 11/12 clock cycles
  out SREG,rSreg ; Restore SREG, +1= 12/13
  reti ; +4=16/17 clock cycles
;
; *********************************
;  M A I N   P R O G R A M   I N I T
; *********************************
;
Main:
  ldi rmp,Low(RAMEND)
  out SPL,rmp ; Init LSB stack pointer
  .ifdef SPH ; For the ATtiny84
    ldi rmp,High(RAMEND) ; Set MSB of stack
    out SPH,rmp
    .endif
  ldi YH,High(sMet)
  ldi YL,Low(sMet)
;
; *********************************
;     D E B U G   R O U T I N E S
; *********************************
;
; Test the DCF77 pulse analysis routine
.if DebugDcfPulse == Yes
  .equ DebugDcfPulseTestTime = 850 ; Time to test in ms
  .equ cDebugDcfPulseTest = DebugDcfPulseTestTime*1000/cTc1T ; Convert to TC1
counts
    sbr rFlag,1<<bDcfErr ; Set or clear the DCF error flag
    ldi ZH,High(cDebugDcfPulseTest) ; Simulate Testtime
    ldi ZL,Low(cDebugDcfPulseTest)
    mov rDcfH,ZH
    mov rDcfL,ZL
    .if (cDebugDcfPulseTest>=cMinuteMin)&&(cDebugDcfPulseTest<cMinuteMax)
      rjmp DebugDcfPulse1
      ; Minute complete, copy DCF77 time pattern
      ; (Use the OpenOffice calculation sheet "dcf77_pattern" to generate)
```

```
      ; DCF77 code for 31.12.2099, 23:59, bytes 0 to 7
      DcfTestTable: .db 0,0,64,102,141,99,202,76
      DebugDcfPulse1:
        ldi rmp,59 ; Simulate 59 bits received
        mov rDcfBits,rmp
        ldi ZH,High(2*DcfTestTable) ; Point to table
        ldi ZL,Low(2*DcfTestTable)
        clr XH ; Point to R0
        clr XL
      DebugDcfPulse2:
        lpm rmp,Z+ ; Read byte
        st X+,rmp ; Write byte to register
        cpi XL,8 ; All 7 written?
        brne DebugDcfPulse2 ; No, continue
      .else
      clr rDcfBits
      .endif
    rcall DcfP ; Check pulse
  DebugDcfPulseLoop:
    rjmp DebugDcfPulseLoop
  .endif
;
; Test the DCF77 conversion routine
.if DebugDcfCode == Yes
     ldi ZH,High(2*DcfCodeTable) ; Point to table
     ldi ZL,Low(2*DcfCodeTable)
     clr XH ; Point to R0
     clr XL
   DebugDcfCodeCopy:
     lpm rmp,Z+ ; Read byte
     st X+,rmp ; Write byte to register
     cpi XL,8 ; All 7 written?
     brne DebugDcfCodeCopy ; No, continue
     rcall DcfMinute ; Convert DCF bits to date/time
   DebugDcfCodeLoop:
     rjmp DebugDcfCodeLoop
   ; (Use the OpenOffice calculation sheet "dcf77_pattern" to generate)
   ; DCF77 code for 31.12.2099, 23:59, bytes 0 to 7
   DcfCodeTable: .db 0,0,64,102,141,99,202,76
  .endif
;
; *********************************
;     N O R M A L   I N I T
; *********************************
  ; Init jumper port
  cbi pJ1D,bJ1D ; Set jumper as input pin
  sbi pJ1O,bJ1O ; Enable pull-up
  ; Init DCF port
  cbi pDcfD,bDcfD ; Set DCF input as input pin
  cbi pDcfO,bDcfO ; Disable pull-up
  ; Init LCD
  .if SkipLcd == No
    rcall LcdInit ; Initiate the LCD
    .endif
  clr ZH ; Back to line 1 column 1
  clr ZL
  .if SkipLcd == No
    rcall LcdPos ; Set position on the LCD
    .endif
  ldi ZH,HIGH(2*InitText) ; Display init text on LCD
  ldi ZL,LOW(2*InitText)
```

```
  .if SkipLcd == No
      rcall LcdText
      ldi rmp,cTxtDelay ; Delay
    Delay1:
      rcall LcdWait50ms ; Wait for 50 ms
      dec rmp ; Count down
      brne Delay1 ; Continue
      clr ZH ; LCD to line 1 column 1
      clr ZL
      rcall LcdPos
      ldi ZH,High(2*ClockMask)
      ldi ZL,Low(2*ClockMask)
      rcall LcdText ; Write mask to LCD
    .endif
  ; Init TC0 as seconds clock
  ldi rDivSec,cDivSec ; Start seconds divider
  ldi rmp,0 ; Normal counting mode
  out TCCR0A,rmp ; in Timer control port A
  ldi rmp,(1<<CS01)|(1<<CS00) ; Prescaler = 64
  out TCCR0B,rmp
  ; Init TC1 as DCF77 signal duration counter
  ldi rmp,High(cTimeOut) ; Time out DCF77 signal input
  out OCR1AH,rmp
  ldi rmp,Low(cTimeOut)
  out OCR1AL,rmp
  ldi rmp,0 ; CTC on OCR0A mode
  out TCCR1A,rmp
  ldi rmp,(1<<WGM12)|(1<<CS11)|(1<<CS10) ; CTC-A, Presc=64
  out TCCR1B,rmp
  ; Init date/time in SRAM
  .if DebugPulseDur == No
    rcall InitDT
    rcall UpDateUtc
    sbrc rFlag,bUtc
    rcall Conv2Utc
    rcall DispAll
        .endif
  ; Interrupt enables
  ldi rmp,1<<TOIE0 ; TC0 overflow interrupt
  out TIMSK0,rmp ; to timer 0 int mask
  ldi rmp,1<<OCIE1A ; Compare match A interrupt
  out TIMSK1,rmp ; to timer 1 int mask
  ; Sleep mode and external interrupts
  ldi rmp,1<<PCINT0 ; Enable PCINT on PA0
  out PCMSK0,rmp
  ldi rmp,(1<<SE)|(1<<ISC00) ; Sleep mode idle, INT0 any changes
  out MCUCR,rmp
  ldi rmp,(1<<INT0)|(1<<PCIE0) ; Enable interrupts
  out GIMSK,rmp ; in general interrupt mask
  sei ; Enable interrupts
;
; *********************************
;    P R O G R A M   L O O P
; *********************************
;
Loop:
  sleep ; Go to sleep
  nop ; Wake up dummy, +1=1
  sbrc rFlag,bDcf ; DCF signal flag clear? +1/2=2/3
  rcall DcfP ; No, evaluate DCF pulse, +3=5 + 14.65 ms max.
  sbrc rFlag,bSec ; Seconds flag clear? +1/2=3/4 wo DcfP, 6/7 + 14.65 ms max
```

```
  rcall Seconds ; No, increase seconds, +3=7
  sbrc rFlag,bUtcChg ; Jumper input unchanged?
  rcall UtcChanged ; UTC jumper has changed
  ; Repeat if lengthy routines in between
  sbrc rFlag,bDcf ; DCF signal flag clear?
  rcall DcfP ; No, evaluate DCF pulse
  sbrc rFlag,bSec ; Seconds flag clear?
  rcall Seconds ; No, increase seconds
  sbrc rFlag,bDcfTO ; DCF signal Timeout clear?
  rcall DcfTimeOut ; Display time out
  rjmp loop
;
; *********************************
;  H A N D L I N G  R O U T I N E S
; *********************************
;
; Seconds flag
Seconds:
  cbr rFlag,1<<bSec ; Clear flag
  .if DebugPulseDur == Yes
    ret
    .endif
  ldd rmp,Y+dSec ; Read seconds
  inc rmp
  std Y+dSec,rmp ; Write seconds
  cpi rmp,60 ; End of minute?
  brcc Minutes ; Yes
  rjmp DisplSc ; Display seconds
Minutes:
  clr rmp ; Seconds restart
  std Y+dSec,rmp
  ldd rmp,Y+dMin ; Read minutes
  inc rmp
  std Y+dMin,rmp ; Write minutes
  cpi rmp,60 ; End of hour?
  brcc Hours
  rjmp DisplMi ; Display minutes
Hours:
  clr rmp ; Minutes restart
  std Y+dMin,rmp
  ldd rmp,Y+dHour ; Read hour
  inc rmp
  std Y+dHour,rmp ; Write hours
  cpi rmp,24 ; End of day?
  brcc Day
  rjmp DisplHr ; Display hour
Day:
  clr rmp ; Restart hours
  std Y+dHour,rmp
  ldd rmp,Y+dWd ; Weekday
  inc rmp
  cpi rmp,7 ; End of week?
  brcs Day1
  ldi rmp,0
Day1:
  std Y+dWd,rmp
  ldd rmp,Y+dDay
  inc rmp ; Next day
  std Y+dDay,rmp
  ldd ZH,Y+dMonth
  cpi ZH,2 ; February
```

```
  brne Day2 ; Not February
  ldd ZH,Y+dYear ; Read year
  andi ZH,0x03 ; Leap year?
  ldi ZL,29 ; February has 28 days
  brne Day4 ; No leap year
  ldi ZL,30 ; February has 29 days
  rjmp Day4 ; Compare day with ZL
Day2:
  cpi ZH,8 ; Month larger than August?
  brcs Day3 ; No, don't reverse number of days
  dec ZH ; Reverse number of days
Day3:
  ldi ZL,31 ; Month has 30 days
  sbrc ZH,0 ; Bit 0 of month
  inc ZL
Day4:
  cp rmp,ZL ; Compare day with number of days
  brcc Month ; Month has ended
  rjmp DisplDy ; Display day
Month:
  ldi rmp,1 ; Restart days
  std Y+dDay,rmp
  ldd rmp,Y+dMonth ; Read month
  inc rmp
  std Y+dMonth,rmp ; Write month
  cpi rmp,13
  brcc Year
  rjmp DisplMo ; Display month
Year:
  ldi rmp,1 ; Restart month
  std Y+dMonth,rmp
  ldd rmp,Y+dYear ; Read year
  inc rmp
  std Y+dYear,rmp ; Write year
  cpi rmp,100 ; End of 100 years?
  brcc Year100 ; Yes
  rjmp DisplYr ; Display year
Year100:
  clr rmp ; Restart Year
  std Y+dYear,rmp
  rjmp DisplYr
;
; *********************************
;  D E F A U L T   D A T E / T I M E
; *********************************
;
; Init date and time
InitDT:
  ldi ZH,High(2*InitDTTable)
  ldi ZL,Low(2*InitDTTable)
InitDTZ:
  ldi XH,High(sMet) ; Pointer to SRAM
  ldi XL,Low(sMet)
InitDT1:
  lpm rmp,Z+ ; Read byte from flash
  st X+,rmp ; Write to SRAM
  cpi ZL,Low(2*InitDTTableEnd)
  brne InitDT1 ; Continue copying
  ret
;
InitDTTable:
```

```
; sMet: ; MET = ss_mm_hh_wd_DD_MM_YY_ST
; sUtc: ; UTC = hh_wd_DD_MM_YY
; sDcf: ; DCF77 time = mm_hh_wd_DD_MM_YY
  .db 0,0,20,6,31,3,19,1,18,6,31,3,19,0,0,0,0,0,0,0
InitDTTableEnd:
;
; The UTC jumper has changed
UtcChanged:
  cbr rFlag,1<<bUtcChg ; Clear change flag
  rcall UpDateUtc ; Get UTC bit
  sbrc rFlag,bUtc ; UTC flag clear?
  rcall Conv2Utc ; Convert date/time to UTC
  rjmp DispAll ; Display all
;
; ************************************
;  D A T E / T I M E   R O U T I N E S
; ************************************
;
; Update the UTC flag
UpdateUtc:
  cbr rFlag,1<<bUtc ; Clear UTC flag
  sbis pJ1I,bJ1I ; UTC jumper input high?
  sbr rFlag,1<<bUtc
  ret
;
; Display all
DispAll:
  ldi ZH,0 ; Set mode position
  ldi ZL,DisplMode
  rcall LcdPos
  ldi ZH,High(2*ModesText) ; Point to modes
  ldi ZL,Low(2*ModesText)
  sbrs rFlag,bUtc ; UTC mode?
  rjmp DispAll1
  adiw ZL,8 ; Point to UTC
  rjmp DispAll2
DispAll1:
  ldd rmp,Y+dSummer ; Summer time?
  cpi rmp,1 ; Summer time = 1?
  brne DispAll2 ; Not summer time
  adiw ZL,4 ; Point to summer time
DispAll2:
  lpm rmp,Z+ ; Read first char
  rcall LcdChar ; Write to LCD
  lpm rmp,Z+ ; Read second char
  rcall LcdChar ; Write to LCD
  lpm rmp,Z+ ; Read third char
  rcall LcdChar ; Write to LCD
  lpm rmp,Z+ ; Read fourth char
  rcall LcdChar ; Write to LCD
  ; Display year
DisplYr:
  ldi ZH,1 ; Set position
  ldi ZL,DisplYear
  rcall LcdPos
  ldd rmp,Y+dYear ; Read year
  sbrc rFlag,bUtc ; MET year?
  ldd rmp,Y+dUtcYear ; Read UTC year
  rcall LcdDec2 ; Display year
DisplMo:
  ldi ZH,1 ; Set position
```

```
  ldi ZL,DisplMonth
  rcall LcdPos
  ldd rmp,Y+dMonth ; Read year
  sbrc rFlag,bUtc ; MET year?
  ldd rmp,Y+dUtcMonth ; Read UTC month
  rcall LcdDec2 ; Display month
DisplDy:
  ldi ZH,1 ; Set position
  ldi ZL,DisplDay
  rcall LcdPos
  ldd rmp,Y+dDay ; Read day
  sbrc rFlag,bUtc ; MET day?
  ldd rmp,Y+dUtcDay ; Read UTC day
  rcall LcdDec2 ; Display day
  ldi ZH,1 ; Set position
  ldi ZL,DisplWd
  rcall LcdPos
  ldi ZH,High(2*Weekday)
  ldi ZL,Low(2*Weekday)
  ldd rmp,Y+dWd ; Read weekday
  sbrc rFlag,bUtc ; Not UTC?
  ldd rmp,Y+dUtcWd ; Read UTC weekday
  lsl rmp ; Multiply by 2
  add ZL,rmp ; Add weekday, LSB
  ldi rmp,0
  adc ZH,rmp ; MSB
  lpm rmp,Z+ ; Read first char
  rcall LcdChar ; char to LCD
  lpm rmp,Z ; Read second char
  rcall LcdChar ; second char to LCD
DisplHr:
  ldi ZH,0 ; Position hours
  ldi ZL,DisplHour
  rcall LcdPos
  ldd rmp,Y+dHour ; Read hours
  sbrc rFlag,bUtc ; UTC flag not set?
  ldd rmp,Y+dUtcHour ; Read UTC hour
  rcall LcdDec2 ; Display hours
DisplMi:
  ldi ZH,0 ; Position minutes
  ldi ZL,DisplMin
  rcall LcdPos
  ldd rmp,Y+dMin ; Read minutes
  rcall LcdDec2 ; Display minutes
DisplSc:
  ldi ZH,0 ; Position seconds, +1=1
  ldi ZL,DisplSec ; +1=2
  rcall LcdPos ; +3+109=114
  ldd rmp,Y+dSec ; Read seconds, +2=116
  rjmp LcdDec2 ; Display seconds, +2+226=344 = 10.5 ms
;
; Weekdays
Weekday:
  .if cDateFormat == 0
    .db "MoDiMiDoFrSaSo"
    .else
    .db "MoTuWdThFrSaSu"
    .endif
;
ModesText:
.if cDateFormat == 0
```

```
  .db "MEZ "
  .db "MESZ"
  .db "UTC "
  .else
  .db "CET "
  .db "CEST"
  .db "UTC "
  .endif
;
; Convert base time to UTC
;   Requires 13..55 clock cycles = 0.40..1.68 ms
Conv2Utc:
  ldd rmp,Y+dSummer ; Read summer time byte, +2=2
  inc rmp ; +1=3
  ldd ZL,Y+dHour ; Read hour, +2=5
  sub ZL,rmp ; Subtract 1/2, +1=6
  std Y+dUtcHour,ZL ; and store in UTC, +2=8
  brcs Conv2Utc1 ; If a carry occurred, +1/2=9/10
Conv2Utc0:
  ret ; No carry, +4=13/27
Conv2Utc1:
  subi ZL,-24 ; Add a day, +1=11
  std Y+dUtcHour,ZL ; +2=13
  ldd rmp,Y+dWd ; Read weekday, +2=15
  subi rmp,1 ; +1=16
  brcc Conv2Utc2 ; Not a monday, +1/2=17/18
  ldi rmp,6 ; Monday, to sunday, +1=18
Conv2Utc2:
  std Y+dUtcWd,rmp ; Store weekday, +2=20
  ldd rmp,Y+dDay ; Read day, +2=22
  subi rmp,1 ; Previous day, +1=23
  std Y+dUtcDay,rmp ; Store day, +2=25
  brne Conv2Utc0 ; Not day zero, +1/2=26/27
  ldd ZL,Y+dMonth ; Read month, +2=28
  subi ZL,1 ; Previous month, +1=29
  std Y+dUtcMonth,ZL ; Write month, +2=31
  brne Conv2Utc3 ; Not zero, +1/2=32/33
  ldi ZL,12 ; Month of last year, +1=33
  std Y+dUtcMonth,ZL ; Write month, +2=35
  ldd ZH,Y+dYear ; Read year, +2=37
  subi ZH,1 ; Previous year, +1=38
  std Y+dUtcYear,ZH ; Write year, +2=40
Conv2Utc3: ; 33/40 clock cycles
  ; Previous month, rmp=UTC-day, ZL=UTC-Month
  cpi ZL,2 ; February? +1=34/41
  brne Conv2Utc4 ; Not February, +1/2=35/36/42/43
  ldd ZH,Y+dUtcYear ; Read UTC-Year, +2=37/38
  andi ZH,0x03 ; Leap year, +1=38/39
  ldi rmp,28 ; Not a leap year, +1=39/40
  brne Conv2Utc6 ; No, +1/2=40/41/42
  ldi rmp,29 ; Leap year, +1=41/42
  rjmp Conv2Utc6 ; +2=43/44
Conv2Utc4: ; 36/43 clock cycles
  ldi rmp,30 ; Month with 30 days, +1=37/44
  cpi ZL,8 ; August ++?, +1=38/45
  brcs Conv2Utc5 ; +1/2=39/40/46/47
  dec ZL ; +1=40/47
Conv2Utc5: ; 40/47
  sbrc ZL,1 ; Month uneven? +1/2=41/42/48/49
  ldi rmp,31 ; Yes, 31 days,+1=42/49
Conv2Utc6: ; 42/43/44/49
```

```
  std Y+dUtcDay,rmp ; Write day, +2=44/45/46/51
  ret ; +4=48/49/50/55
;
; Handle DCF pulse
;   Requires
;      0.89 ms for a Zero
;      1.25 ms for a One
;      1.34 ms for a Pause following an error, or
;     14.55 ms for a usual Pause
;      1.71 ms for minute pulse without date/time conversion
;     14.65 ms for minute pulse date/time conversion
;     27.86 ms for minute pulse plus date/time conv plus message
.equ cSecDly = (2786*256)/50000
DcfP:
.if DebugPulseDur == Yes
  ; Displays counts of INT0 duration
  ldi ZH,0 ; Line on LCD
  sbic pDcfI,bDcfI ; Input pin low?
  ldi ZH,1 ; No, high
  ldi ZL,DisplPulseDur ; Column position
  rcall LcdPos ; Set Position
  mov ZH,rDcfH ; Copy MSB count
  mov ZL,rDcfL ; dto., LSB
  rjmp LcdDec5 ; Display count decimal 5 digits
  .else
  cbr rFlag,1<<bDcf ; Clear flag
  clr rDcfErr ; DCF error to zero
  inc rDcfErr ; dto., to one
  ldi rmp,Low(cZeroMin) ; Signal a zero? LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cZeroMin) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcs DcfE ; Error 1: signal too short
  ldi rmp,Low(cZeroMax) ; Signal a zero, LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cZeroMax) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcc DcfP1 ; Not a zero
  clc ; Clear carry, shift 0 into registers
  rjmp ShiftBit ; Shift into
DcfP1:
  inc rDcfErr ; Next error number
  ldi rmp,Low(cOneMin) ; Signal a one? LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cOneMin) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcs DcfE ; Error 2: Signal between zero and one
  ldi rmp,Low(cOneMax) ; Signal a one, LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cOneMax) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcc DcfP2 ; Not a one
  sec ; Set carry one
  rjmp ShiftBit ; Shift into
DcfP2:
  inc rDcfErr ; Next error number
  ldi rmp,Low(cPauseMin) ; Signal a pause? LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cPauseMin) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcs DcfE ; Error 3: Signal between one and pause
```

```
  ldi rmp,Low(cPauseMax) ; Signal a pause, LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cPauseMax) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcc DcfP3 ; Not a pause
  sbrc rFlag,bDcfOk ; DCF not ok?
  ret ; DCF ok, do not clear DCF space
  sbrs rFlag,bDcfErr ; Error condition on LCD?
  rjmp DcfClr ; Clear the error field
  cbr rFlag,1<<bDcfErr ; Clear error flag
  ret ; A correct pause
DcfP3:
  inc rDcfErr ; Next error number
  ldi rmp,Low(cMinuteMin) ; Signal a minute? LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cMinuteMin) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcs DcfE ; Error 4: Signal between a pause and a minute
  ldi rmp,Low(cMinuteMax) ; Signal a one, LSB
  cp rDcfL,rmp ; Compare LSB
  ldi rmp,High(cMinuteMax) ; dto., MSB
  cpc rDcfH,rmp ; Compare MSB
  brcc DcfP4 ; Not a minute
  inc rDcfErr ; Next error number
  inc rDcfErr ; Next error number
  ldi rmp,59 ; 59 bits received?
  cp rmp,rDcfBits ; Compare
  breq DcfMinute ; Correct, convert DCF bits
  ldi ZH,0 ; Lcd to DCF position
  ldi ZL,DisplDcf
  rcall LcdPos
  mov rmp,rDcfBits ; Display number of bits
  clr rDcfBits
  rjmp LcdDec3 ; Convert DCF bits to date/time
DcfP4:
  inc rDcfErr ; Error 5: Longer than minute
  .endif
;
; An error occurred during DCF signal checking
DcfE:
  ldi ZH,0 ; Position to DCF output
  ldi ZL,DisplDcf
  rcall LcdPos
  ldi rmp,' ' ; A blank
  rcall LcdChar
  ldi rmp,'E' ; E character
  rcall LcdChar
  ldi rmp,'0' ; ASCII Null
  add rmp,rDcfErr ; Add error number
  rcall LcdChar ; Display error character
  sbr rFlag,1<<bDcfErr ; Set error flag
  cbr rFlag,1<<bDcfOk ; Clear ok flag
  ret
;
; Minute complete, convert DCF bits to date/time
;   Requires 26.1  milliseconds
.if DebugPulseDur == No
DcfMinute:
  ; Check  minutes
  clr rDcfBits ; Restart DCF bit collection
  ldi rmp,7 ; Error number 7
```

```
  mov rDcfErr,rmp
  mov ZL,rDcf3 ; Copy DCF bits minutes
  mov ZH,rDcf4
  lsr ZH ; Shift by two bits right
  ror ZL
  lsr ZH
  ror ZL
  rcall Parity ; Check parity minutes
  brne DcfE ; Error 7: Parity minutes odd
  inc rDcfErr ; Next error number
  mov ZL,rDcf3 ; Minute ones
  lsr ZL ; Shift two bits right
  lsr ZL
  andi ZL,0x0F ; Isolate ones
  cpi ZL,10 ; Larger than ten?
  brcc DcfE ; Error 8: Minute ones larger than 9
  inc rDcfErr ; Next error number
  mov ZH,rDcf4 ; Minute tens
  sbrc ZH,0 ; Fourties
  subi ZL,-40 ; Add fourty
  mov ZH,rDcf3 ; 20s and 10s
  sbrc ZH,7 ; Twenties
  subi ZL,-20 ; Add twenty
  sbrc ZH,6 ; Tens
  subi ZL,-10 ; Add ten
  cpi ZL,60 ; Larger than 59?
  brcc DcfE ; Error 9: Minutes larger than 60
  std Y+dDcfMin,ZL ; Save DCF minutes
  ldi rmp,'A'-10-'0' ; Error number to characters
  mov rDcfErr,rmp
  mov ZL,rDcf4 ; Read hours
  mov ZH,rDcf5 ; and their parity bit
  lsr ZH ; Shift parity bit into hours
  ror ZL
  lsr ZL ; Shift hours right
  mov ZH,ZL ; Copy to ZL
  rcall Parity
  breq DcfHours ; Parity hours ok
  rjmp DcfE ; Error A: Hours parity odd
DcfHours:
  inc rDcfErr ; Next error number
  mov ZL,ZH ; Copy hours to ZL
  andi ZL,0x0F ; Isolate ones
  cpi ZL,10 ; Larger than 9?
  brcs DcfHours1 ; No, ok
  rjmp DcfE ; Error B: Hour ones larger than 9
DcfHours1:
  inc rDcfErr ; Next error number
  sbrc ZH,5 ; Bit 7 in rDcf4
  subi ZL,-20 ; Add twenty
  sbrc ZH,4 ; Bit 6 in rDcf4
  subi ZL,-10 ; Add ten
  cpi ZL,24 ; Hours larger than 24?
  brcs DcfHours2 ; Hours ok
  rjmp DcfE ; Error C: Hours larger than 23
DcfHours2:
  std Y+dDcfHour,ZL ; Save DCF hours
  inc rDcfErr ; Next error number
  mov ZL,rDcf5 ; Copy date byte 0
  mov ZH,rDcf6 ; Copy date byte 1
  mov rmp,rDcf7 ; Copy date byte 2
```

```
  lsr rmp ; Shift one bit right
  ror ZH
  ror ZL
  rcall Parity ; Check parity byte 0
  mov ZL,ZH
  rcall Parity1 ; Check parity byte 1
  mov ZL,rDcf7
  lsr ZL
  rcall Parity1
  breq DcfDay
  rjmp DcfE ; Error D: Date parity odd
DcfDay:
  inc rDcfErr ; Next error number
  mov ZL,rDcf5 ; Read day
  lsr ZL ; One bit right
  andi ZL,0x0F ; Isolate ones
  cpi ZL,10 ; Larger than nine?
  brcs DcfDay1 ; Day ones fine
  rjmp DcfE ; Error E: Day ones larger than nine
DcfDay1:
  inc rDcfErr ; Next error number
  mov ZH,rDcf5 ; Read tens
  sbrc ZH,6 ; Twenties
  subi ZL,-20 ; Add twenty
  sbrc ZH,5 ; Tens
  subi ZL,-10 ; Add ten
  cpi ZL,32 ; More than 32 days?
  brcs DcfDay2
  rjmp DcfE ; Error F: Day larger than 31
DcfDay2:
  std Y+dDcfDay,ZL ; Save DCF day
  inc rDcfErr ; Next error number
  mov ZL,rDcf5 ; Read weekday
  mov ZH,rDcf6
  lsl ZL
  rol ZH
  andi ZH,0x07
  brne DcfWeekday
  rjmp DcfE ; Error G: Weekday is zero
DcfWeekday:
  dec ZH ; Weekdays 1 to 7 to 0 to 6
  std Y+dDcfWd,ZH ; Save weekday
  inc rDcfErr ; Next error number
  mov ZH,rDcf6 ; Read month
  mov ZL,rDcf6
  lsr ZL
  lsr ZL
  andi ZL,0x0F ; Isolate ones
  cpi ZL,10 ; Ones larger than 9?
  brcs DcfMonth
  rjmp DcfE ; Error I: Month ones larger than 9
DcfMonth:
  inc rDcfErr ; Next error number
  sbrc ZH,6 ; Tens
  subi ZL,-10 ; Add ten
  cpi ZL,13 ; Month larger than 12?
  brcs DcfMonth1
  rjmp DcfE ; Error H: Month larger than 12
DcfMonth1:
  std Y+dDcfMonth,ZL ; Save DCF month
  inc rDcfErr ; Next error number
```

```
  mov ZL,rDcf6 ; Copy year
  mov ZH,rDcf7
  lsl ZL
  rol ZH
  mov ZL,ZH
  andi ZL,0x0F ; Isolate ones
  cpi ZL,10 ; Ones larger than 10?
  brcs DcfYear
  rjmp DcfE ; Error J: Year ones larger than ten
DcfYear:
  inc rDcfErr ; Next error number
  sbrc ZH,7 ; Eighty
  subi ZL,-80
  sbrc ZH,6 ; Fourty
  subi ZL,-40
  sbrc ZH,5 ; Twenty
  subi ZL,-20
  sbrc ZH,4 ; Ten
  subi ZL,-10
  cpi ZL,100 ; Year larger than 99?
  brcs DcfYear1
  rjmp DcfE ; Error K: Year larger than 99
DcfYear1:
  std Y+dDcfYear,ZL ; Save DCF year
  clr rmp ; Winter time
  mov ZL,rDcf2
  sbrc ZL,6 ; Bit 6 is MESZ bit
  inc rmp
  std Y+dDcfSummer,rmp ; Save DCF summer time
  inc rDcfErr ; Next error number
  mov ZL,rDcf0 ; Read first bit
  sbrc ZL,5 ; Bit 5 is first DCF bit
  rjmp DcfE ; Error L: First bit is not zero
  inc rDcfErr ; Next error number
  mov ZL,rDcf3 ; Read start bit date/time
  sbrs ZL,1 ; Bit 1 is start bit, one?
  rjmp DcfE ; Error M: Date/Time start bit not a one
  ; DCF date/time complete, copy over date/time
  ldi ZH,High(sDcf) ; Point Z to DCF in SRAM
  ldi ZL,Low(sDcf)
  ldi XH,High(sMet+1) ; Point Z to MET-minutes in SRAM
  ldi XL,Low(sMet+1)
DcfCopy:
  ld rmp,Z+ ; Copy byte
  st X+,rmp
  cpi ZL,Low(sDcf+8) ; End of DCF time
  brne DcfCopy
  rcall DcfOkMsg ; Display ok message
  clr rmp ; Clear seconds
  st Y,rmp
  cbr rFlag,1<<bSec ; Clear seconds bit
  ldi rDivSec,cDivSec ; Restart seconds divider
  ldi rmp,cSecDly ; Set seconds counter
  out TCNT0,rmp ; Set execution delay time
  sbr rFlag,(1<<bUtcChg)|(1<<bDcfOk) ; Force update of display
  sbrc rFlag,bDcf ; A new DCF signal to process?
  rjmp DcfP ; Yes, start new
  ret
;
; Check parity of byte in ZL
;   Returns zero flag clear if even
```

```
Parity:
  clr rmp
Parity1:
  lsr ZL ; Shift lowest bit to carry
  breq Parity2 ; No ones left
  brcc Parity1 ; Bit not one
  inc rmp ; Count ones
  rjmp Parity1
Parity2:
  brcc Parity3 ; Last bit a zero?
  inc rmp ; Count last bit
Parity3:
  andi rmp,0x01 ; Test lowest bit even
  ret
;
; Shift bit in carry to DCF bit shift registers
ShiftBit:
  ror rDcf7 ; Rotate into 7
  ror rDcf6 ; dto., into 6
  ror rDcf5 ; dto., into 5
  ror rDcf4 ; dto., into 4
  ror rDcf3 ; dto., into 3
  ror rDcf2 ; dto., into 2
  ror rDcf1 ; dto., into 1
  ror rDcf0 ; dto., into 0
  inc rDcfBits ; Count bits
  .if (cDcfMoniBits == 1)||(cDcfMoniBitCount == 1)
    ldi ZH,0 ; Set display position
    ldi ZL,DisplDcf ; to DCF position on LCD
    rcall LcdPos ; Set position
    .endif
  .if cDcfMoniBits == 1
    ; Monitor bits binary
    mov ZL,rDcf7 ; Read the last eight bits
    ldi rmp,'0' ; A zero
    sbrc ZL,7 ; Last incoming bit
    ldi rmp,'1' ; A one
        rcall LcdChar ; Write char to LCD
        ldi rmp,'0' ; A zero
    sbrc ZL,6 ; Pre-last bit
    ldi rmp,'1' ; A one
    rcall LcdChar ; Display the last
    ldi rmp,'0'
    sbrc ZL,5 ; Pre-pre-last bit
    ldi rmp,'1' ; A one
    rjmp LcdChar ; Display the third bit
    .endif
   ; Monitor the incoming bits
  .if cDcfMoniBitCount == 1
    ; Monitor the bit count
    mov rmp,rDcfBits ; Copy bit count
    rjmp LcdDec3 ; Display the bit count
    .endif
  ret
  .endif
;
; DCF signal timeout
DcfTimeOut:
  cbr rFlag,1<<bDcfTO ; Clear flag
  clr rDcfErr
  rjmp DcfE ; Error 0: Time-out on the DCF input pin
```

```
;
; Clear error field on LCD
DcfClr:
  ldi ZH,High(2*DcfClearTxt)
  ldi ZL,Low(2*DcfClearTxt)
  rjmp DcfTxt
;
; DCF ok message
;   requires 433 clock cycles = 13.21 ms
DcfOkMsg:
  ldi ZH,High(2*DcfOkTxt) ; +1=1
  ldi ZL,Low(2*DcfOkTxt) ; +1=2
DcfTxt:
  push ZH ; Save text position, +2=4
  push ZL ; +2=6
  ldi ZH,0 ; +1=7
  ldi ZL,DisplDcf ; +1=8
  rcall LcdPos ; +3+109=120
  pop ZL ; +2=122
  pop ZH ; +2=124
  lpm rmp,Z+ ; +2=126
  rcall LcdChar ; +3+99=228
  lpm rmp,Z+ ; +2=230
  rcall LcdChar ; +3+99=332
  lpm rmp,Z ; +2=334
  rjmp LcdChar ; +2+99=443
;
DcfClearTxt:
  .db "    " ; Clear error message
DcfOkTxt:
  .db " ok " ; DCF ok message
;
; *********************************
;     L C D  R O U T I N E S
; *********************************
;
; LCD text masks, 16 chars per line
InitText:
 .db " DCF77 clock V3 ",0x0D,0xFF
 .db " (C)2019 DG4FAC ",0xFE,0xFF
ClockMask:
.if DebugPulseDur == Yes
  .db "Pulse lo = 12345",0x0D,0xFF
  .db "      hi = 12345",0xFE,0xFF
  ;    0123456789012345
  .else
  .if cDateFormat == 0
    .db "hh:mm:ss utc dcf",0x0D,0xFF
    .db " wd, DD.MM.20YY ",0xFE,0xFF
    .else
    .db "hh:mm:ss utc dcf",0x0D,0xFF
    .db " wd, MM/DD/20YY ",0xFE,0xFF
    ;    0123456789012345
    .endif
  .endif
;
; Display positions, line 1
.equ DisplSec = 6
.equ DisplMin = 3
.equ DisplHour = 0
.equ DisplMode = 9
```

```
.equ DisplDcf = 13
; Display positions, line 2
.equ DisplWd = 1
.if cDateFormat == 0
  .equ DisplDay = 5
  .equ DisplMonth = 8
  .else
  .equ DisplDay = 8
  .equ DisplMonth = 5
  .endif
.equ DisplYear = 13
.if DebugPulseDur == Yes
  .equ DisplPulseDur = 11
  .endif
;
; LCD include routines
.include "lcd.inc"
;
; End of source code
```

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