// --- [dcf77_readme] --

DCF77-Uhr + TWI-Slave + LCD + Sonnenstandsberechnung für m168 in C

Das Programm besteht aus mehreren Modulen:

dcf.c, dcf_bits.c	Ermittelt die Zeit aus dem CDF-77 Signal und korrigiert Fehler.
alarm.c	Zu Beginn einer jeden Minute wird geprüft, ob zuvor definierte Schaltzeiten erreicht sind. Ggf. werden dann Schaltvorgänge an Pins des Prozessors oder auf TWI-Slave ausgeführt.
sun.c	Berechnet den aktuellen Sonnenstand sowie den Sonnenauf-/untergang
twi_master_slave.c	Lässt den Controller als TWI-Slave auf externe Kommandos reagieren, etwa um Speicherinhalte zu verändern. Gleichzeitig aggiert er als TWI-Master, um Schaltvorgänge auf TWI-Slaves zu initieren.

Überarbeitung 25.Oktober 2009

Neu eingebaut, um Fehler im Programm zu finden:

- Alle Kommandos, die Zeitarrays der Uhrzeit verändern, setzen das Fehlerflag (auf 0xFC).
  Ein gesetztes Fehlerflag erzwingt eine 'manuelle' Fortschreibung der Uhrzeit.
  So ändert das Kommando *TWI_ADR, 0xF0, Minute, Stunde, DOW, Tag, Monat, Jahr*
  die aktuelle Zeit (Die Änderung wird erst in der folgenden Minute im Display sichtbar !)
  In der folgenden Minute wird aufgrund der Fehlerkorrektur (des Vergleiches zweier
  aufeinander folgender DCF-Signale) wieder auf die richige Zeit geschaltet.

  Sinn: zu Testzwecken die Uhrzeit ändern um festzustellen, was zu Beginn der neuen
  Minute geschehen wird (Funktionieren die Schaltzeitpunkte, welcher Sonnenstand wird
  ermittelt ?)

- Ein Logfile kann via RS232 ausgegeben werden:
  Es beinhaltet für jede Minute (ausgegeben wird in der 59. Sekunde):
  - die aktuelle Zeit, s/w für Sommer/Winterzeit,
  - die DCF-Bits der kommenden Minute von Bit.15 bis Bit.58
    die logischen Bit-Gruppen sind jeweils durch einen . separiert,
    die Paritätsbits als ! (wenn erfolgreich) bzw. ? (wenn Fehler) dargestellt
    sollten Zusatzsbits gesetzt sein (A1/A2/R), dann werden diese separat ausgegeben

```
               |<-DCF-Bit 15
                     |<-DCF-Bit 21
                              |<-Paritätsbit                      |<DCF-Bit 58
18:13-s-24.10.09 001001.0010100!.000110!.001001.011.00001.10010000!
18:14-s-24.10.09 001001.1010100!.000110!.001001.011.00001.10010000!
                      Minute   Stunde  Tag    DOW Monat Jahr
```

  Am Anfang einer Minute (in Sekunde 0) werden zusätzlich Informationen zu den gerade
  durchgeführten Schaltvorgängen ausgegeben:
  - der Slot (Speicherplatz),
  - die Regel (also die Filterkriterien Uhrzeit ... etc.)
  - der betroffene Schalter sowie die durchgeführte Aktion (EIN/AUS/TOGGLE)

```
18:15-s-24.10.09 slot: 02, rule: 015.---.---.---.---.---.002.001. -> switch: 02, action: on
```

  Diese Regel wurde definiert (siehe unten)durch die Eingabe

  SLAVE_ADR,2,15,128,128,128,128,128,2,1

  Beim Start des Programmes werden alle Schaltdefinitionen via RS232 ausgeben, die im Eeprom
  gespeichert waren und nun wieder aktiv werden:

```
slot: 00, rule: 015.---.064.---.---.---.001.001. -> switch: 00, action: on
slot: 01, rule: 030.---.064.---.---.---.001.002. -> switch: 00, action: off
slot: 02, rule: 015.---.---.---.---.---.002.001. -> switch: 02, action: on
slot: 56, rule: 000.---.---.---.---.---.011.129. -> switch: 10, action: on, single action
slot: 62, rule: 000.012.---.---.---.---.064.099. -> TWIslave: 64, MSG: 99
               |< Minuten             |<Schalter
      |            |< Stunden             |<Aktion
      |               |<DOW
      |<Speicherplatz     |<Tag
                             |<Monat      --- = wird nicht verglichen (Bit.7 gesetzt)
                                |<Jahr
```

  Das Generieren eines Logfiles kann durch Auskommentieren von #define LOG_RS232
  unterbunden werden.

- Geänderte ist die Logik des DCF-Empfangs sowie der Synchronisation mit der Quarzuhr. Stark gestörter Empfang konnte die Uhr zeitweise aus dem Takt bringen.
- Diverse Fehler, die beim Betrachten des Logfiles sichtbar wurden, sind beseitigt:
  - einmalige Schaltdefinitionen löschten mehr als nur sich selbst aus dem Speicher,
  - Fehler in der Ermittlung des Übergangs Sommer-/Winterzeit,
  - Fehler beim decrementieren der Zeit

Die geänderte Logik der DCF-Synchonisation

Beim Start des Programms wird im Sekundentakt ein Zähler im Display aktualisiert.
Bei 59 wird er wieder auf 0 zurückgesetzt.
Parallel wird das DCF-Signal mitgelesen und nach dem Erkennen eines 2-Sekunden-Taktabstandes ein (Minuten)Flag gesetzt.
Nun wird die Warteschleife in der Initialisierung verlassen und das Programm läuft in seiner Hauptschleife.

Jede Sekunde wird vom Timer0 ein Flag gesetzt, dass die Aktualisierung der Sekundenanzeige im LCD steuert.

Mit dem Beginn einer neuen Minute werden die DCF-Bits aufgezeichnet.

Nach 58 Sekunden ist der komplette DCF-Datensatz eingelesen.

In der 59. Sekunde wird von der Quarzuhr ein Flag gesetzt, das die Auswertung des DCF-Signals veranlasst.

Das DCF-Signal wird decodiert und die Paritätsbits werden geprüft.
Ist ein Fehler im DCF-Signal der letzten Minute aufgetreten oder stimmt ein Paritätsbit nicht, dann wird die nächste Minute 'rechnerisch' durch durch Addition einer Minute bestimmt.

Ist kein Fehler erkannt worden, dann muss das Signal dennoch nicht korrekt sein (die Paritätsprüfung kann z.B. keine Verdreher erkennen).

Daher wird die decodierte DCF-Zeit der neuen Minute mit der incrementierten letzten Minute der Uhr verglichen.

Sind beide identisch, dann wird davon ausgegangen, das die neue Zeit korrekt ist.

Ein Fehler tritt natürlich nach dem Einschalten der Uhr auf, wenn keine Vergleichszeit vorhanden ist.

Darum wird bei der Initialisierung ein Startflag gesetzt, das den Neustart markiert.
Bei gesetztem Bit wird die erste fehlerfreie DCF-Zeit in die Uhr übernommen.

Gelingt der Vergleich zwischen zwei aufeinanderfolgen Minuten nicht, dann wird die neue DCF-Zeit mit dem letzten DCF-Datensatz (+ 1 Minute) verglichen.

Sind beide gleich, dann wird die Uhrzeit korrigiert, indem die DCF-Zeit in die Uhr kopiert wird.

Stimmen beide nicht überein, dann wird eine Fehlermeldung gesetzt - und anschließend incrementiert die Uhr manuell ohne weitere Prüfung.

Durch diese Logik sollten sich Zeitfehler korrigieren, sobald 2 aufeinander folgende DCF-Zeiten logisch aufeinander folgen.

Die Quarzuhr

Der Timer0 liefert einen 1 Sekunden Takt (oder genauer den 225-fachen 1-Sekundentakt) und steuert damit die Sekundenanzeige.

Bei jedem Aufruf wird ein Counter decrementiert; hat der Counter 0 erreicht, dann ist ein Sekunde um.

Nach 60 Sekunden kann er das Zeichen für den Anbruch einer neuen Minuten geben, sofern das DCF-Signal fehlerhaft ist.

Der Timer1 ist ein 16-Bit Timer, der im 10ms Takt über die COMPA_ISR aufgerufen wird.
Auch hier wird ein Counter incrementiert, der als Zeitzähler für die Impulsdauer des DCF-Signals dient.

Der 8-bit Counter läuft bei 255 (= 2.55 Sekunden) über und wird dann auf 255 gehalten.
Fällt das DCF-Signal aus, dann bleibt der Counter auf 255 stehen und es wird ein Fehlercode gesetzt. Wodurch sich das DCF-Signal für die Zeitmessung disqualifiziert hat.

DCF-Signal und Quarzuhr

Das DCF-Signal und die Quarzuhr arbeiten mit- und nebeneinander:
Das DCF-Signal liefert die absolute Zeitinformation.

Einmal synchronisiert, kann die Quarzuhr diese Zeit fortschreiben.
Der Start einer neuen Minute, wenn er bei fehlerfreiem Empfang aus dem DCF-Signal erkannt wird, setzt ein Flag für den Beginn einer neuen Minute und stellt die Sekundenzähler der Quarzuhr jeweils auf 0 zurück, so dass DCF-Signal und Quarzuhr über dem Minutenstart synchron laufen.

Bei fehlerhaftem DCF-Signal entfällt diese Synchronisation.
Nun erkennt die Quarzuhr in der 60. Sekunde, dass ein Fehler im DCF-Signal vorliegt und setzt ihrerseits das Flag für die neue Minute.

Auch bei (scheinbar) fehlerfreiem DCF-Empfang wird die um 1 Minute incrementierte aktuelle Quarzzeit mit der DCF-Zeit der neuen Minute verglichen.

Bei Gleichheit wird die Zeit als korrekt angenommen.

Bei Ungleichheit wird zunächst der Fall "Programmstart" untersucht.
Hier existiert noch keine Vergleichszeit.
Sofern das Startflag gesetzt ist (wird in der Initialisierung gesetzt)
wird die DCF-Zeit übernommen, sofern keine Fehler aufgetreten sind.

Liegt kein Programmstart vor, dann wird das incrementierte DCF-Signal der aktuellen Minute mit dem DCF-Signal der folgenden Minute verglichen.

Sind beide gleich, dann wird unterstellt, dass die Quarzuhr-Zeit falsch ist.

Sie wird durch die DCF-Zeit überschrieben.

Sonstige Änderungen gegenüber dem Stand 02.05.09

Der Schalter #64 wird auf PIND.7 gemappt.
Die Schalter oberhalb #64 sind deaktiviert (kann geändert werden).
Es sind 62 (nicht 63) Schaltdefinitionen möglich (weil mehr nicht ins Eeprom passt).
Die Schaltdefinition werden von 0 .. 62 durchnummeriert.
Die Schalter werden von 0 .. 64 gezählt, wobei die #0..63 auf TWI-Slaves ausgegeben werden.

Bekannter Fehler:

Beim Wechsel von Sommer- auf Winterzeit änderte sich die Zeit für den Sonnenaufgang
von 07:51 auf 06:36. Das kann nicht sein !
Die Ursache kann ich im Moment nicht nachvollziehen.
Die Änderungen der Zeit für den Sonnenuntergang ist plausibel (von 18:13 auf 17:11).
Der berechnete Sonnenstand stimmt mit den Angaben des Programms ORBITRON.exe überein.

Überarbeitung 02. Mai 2009

Änderungen gegenüber der Version vom Januar:

- Hardware: nun ist leider ein mega168 erforderlich.
  Das Programm benötigt ca. 9900 Byte, zuviel für den m8. Wenn man allerdings auf die Sonnenstandsberechnung verzichtet, dann ist das Programm nur noch ca. 5400 Byte lang.
  Und ohne LCD sind nur 4100 Byte erforderlich.
- Der für einen externen Master nutzbare TWI-Speicher (768 Byte) entfällt.
  Dieses SRAM wird genutzt, um zusätzliche Schaltdefinitionen aufnehmen zu können.
- Der Day_Of_Week (DOW) wird bitweise gespeichert.
  Somit können nun unterschiedliche Wochentage mittels Maske gefiltert werden (benutzt werden die Bits DOW.0 .. 6).
  Ein gesetztes Bit.0 entspricht Montag, Bit.6 entspricht Sonntag.
  Die Sommerzeit (bislang als Bit.4 in DOW gespeichert) ist verschoben nach Bit.7
- Jeweils zum Zeitpunkt des Sonnenaufgangs bzw. Sonnenuntergangs wird der Schalter 64 (PD.0) einmalig ein- bzw. ausgeschaltet.
  Kann durch Auskommentieren von #define DAY_LIGHT entfernt werden.
- Jeweils zu vollen 15 Minuten wird via TWI ein Signal an einen Slave gesendet (als Zeitzeichen).
  Kann durch Auskommentieren von #define ZEIT_ZEICHEN entfernt werden.
- Das TWI-Modul arbeitet nun alternativ als Slave (Normalfall) oder als Master (wenn Daten an externe Slaves gesendet werden sollen).
  Im Slave-Modus kann der gesamte Speicher der Uhr gelesen/geschrieben werden.
  Im Master-Modus kann die Uhr Pins an einem Portexpander ansprechen - oder aber einfach nur messages an einen Slave senden
  (z.B. um einen Weckton zu einem definierten Zeitpunkt auszugeben).
- Die Anzahl der definierbaren Schaltzeitpunkte ist auf 63 erweitert (die Beschränkung rührt daher, dass so alle Definitionen im Eeprom gespeichert werden wollen).
  Grundsätzlich ist noch Platz für zusätzliche Definition vorhanden - die sind dann aber nach einem Reset ‘flüchtig’.
- Die Anzahl der adressierbaren Schaltpins ist auf 70 erhöht.
  Davon sind 6 an den freien Portpins von PORTD des ATMEGA verfügbar.
  Die übrigen 64 Pins können auf 8 TWI-Slaves als Portexpander angesprochen werden.
- Alle Daten der Uhr und auch die Schaltdefinitionen sind in einem 2-dimensonalen Array abgelegt. Von außen kann ein Pointer auf jedes Element des Arrays gesetzt werden: alle Daten können von außen gelesen und beschrieben werden (wobei letzteres bei den Daten der Uhr natürlich wenig sinnvoll ist !).

Um Speicherbereiche von aussen auszulesen oder zu beschreiben, ist nur noch 1 Byte als Speicheradresse bzw. Kommandos erforderlich.
In den nachfolgend Beispielen sei 0x08 die TWI-Adresse der Uhr.
Die Reihenfolge ist immer: TWI-Adr, Speicher-Adr, Daten (8 Byte):

```
0x08, 0xF0   positioniert den Pointer auf der aktuellen Uhrzeit
0x09, ...    liest die Bytes der aktuellen Uhrzeit aus
             Die Reihenfolge ist immer:
             Minute, Stunde, DOW, Tag, Monat, Jahr, Sekunde, 0
             12,20,129,1,5,9,45,0
             DOW 129 = 128 für Sommerzeit + 1 für Montag

0x08, 0xF1   positioniert den Pointer auf der DCF-Zeit
0x09, ...    liest die dekodierte DCF-Information der aktuellen Minute:
             Minute, Stunde, DOW, Tag, Monat, Jahr, Zusatzbits, 0
             12,20,129,1,5,9,2,0
```

0x08, 0xF2 positioniert den Pointer auf der UTC-Zeit
0x09, ...   liest die UTC-Zeit der aktuellen Minute:
            Minute, Stunde, DOW, Tag, Monat, Jahr, 0, 0
            12,20,129,1,5,9,2,0

0x08, 0xF3 positioniert den Pointer auf den Sonnenstandsdaten
0x09, ...   liest die Sonnenstandsdaten der aktuellen Minute:
            Elevation (high_byte), Elevation(low_Byte) (-> int16 !),
            Azimut(high_Byte), Azimut (lowbyte) (-> uint16 !),
            Sonnenaufgang (Minute), Sonnenaufgang (Stunde),
            Sonnenuntergang (Minute), Sonnenuntergang (Stunde)
            Elevation und Azimut sind mit 100 multipliziert (für 2 Nachkommastellen),
            Elevation ist als INT16 abgelegt (wegen negativer Werte)

0x08, 0xF4 positioniert den Pointer auf den DCF-Bits der aktuellen Minute
0x09, ...   liest die DCF-Bits der aktuellen Minute.
            Das sind die Bits uninterpretiert so wie per Funk empfangen.

0x08, 0xF5 positioniert auf interne Zeitinformationen (time[CMP])
0x08, 0xF6 positioniert auf interne Zeitinformationen (time[BAK])

0x08, 0xFA ist ein Kommando, das die aktuellen Schaltdefinitionen im Eeprom ablegt.
0x08, 0xFB, 0x10 (oder beliebige andere Adresse)
            ist ein Kommando, das die bisherige TWI-Adresse 0x08 z.B. auf 0x10 ändert.
            Die Änderung wird im Eeprom gespeichert und beim Neustart initialisiert.
0x08, 0xFC ist ein Kommando, das alle Schaltdefinitionen im SRAM mit 0 überschreibt.

0x08, 0xFD ist ein Kommando, das alle Schaltdefinitionen im Eeprom mit FF überschreibt.

0x08, 0x00 ... 0x3F
            positioniert den Pointer auf den Schaltdefinition #0x00 .. #0x3F

0x09, ...   liest die Schaltdefinionen (auf die der Pointer gerade zeigt) aus,
            Reihenfolge der Datenbytes:
            Minute, Stunde, DOW, Tag, Monat, Jahr, Schalter, Modus
            z.B.: 30,12,128,128,128,128,128,1,3
            Bedeutet: Hier wird um 12:30 der Schalter #1 getoggled

            Wenn in einer Zeitangabe AUSSCHLIESSLICH das Bit.7 gesetzt ist, dann wird
            diese Zeit nicht in den Vergleich einbezogen.
            Durch Eingabe von 128 (= Bit.7) werden Minute, Stunde, DOW, Tag, Monat und
      Jahr nicht in den Vergleich einbezogen, die aktuellen Werte sind also nicht
            relevant.
            Im Byte DOW signalisiert das Bit.7 die Sommerzeit.
            Ist nur Bit.7 gesetzt, dann wird DOW nicht ausgewertet.
            Wenn DOW == 129, dann trifft dies an jedem Montag UND Sommerzeit zu,
            Wenn DOW == 255, dann trifft dies an jedem Wochentag UND Sommerzeit zu.
            Wochentage können somit immer nur in Verbindung der Angabe von Sommer- oder
            Winterzeit als Filterbedingung fungieren.

            Schalter 0 - 63 sind Pins an 8 Portexpander nach Art des PCF8574 mit
                  je 8 Pins
                  64 - 69 sind (ggf.) freie Pins an PORTD des Controllers
                  Die Nummerierung der Schalter beginnt mit 0, am ersten Expander sind
                  die Schalter 0 .. 7 vefügbar.
            Modus: Bit.7 wird maskiert. Es veranlasst die Lösung der Definition
            nach einmaliger Ausführung.
                  Es können nun noch die Werte 0 ... 127 vorhanden sein.

                  0 inaktiv
                  1 einschalten
                  2 ausschalten
                  3 togglen
                  4 ... 127 : dieser Wert wird als Message an einen TWI-Slave gesendet,
                  die Adresse des Slave wird aus der (Schalternummer & 0xFE) gebildet.
                  Erforderlich ist das, damit Bit.0 (Read-Bit) NICHT gesetzt ist.
                  Wenn also kein Schalterzustand (1..3) definiert ist, dann greift
                  diese Regel und versendet eine Meldung an einen TWI-Slave!

Diese Option ermöglicht, zu einem definierbaren Zeitpunkt eine Meldung an einen Slave zu sen-
den. Der angegebene "Schalter" wird als TWI-Adresse des Slaves interpretiert.
Die Bits 2,3,4,5,6 sind die Message, die an den Slave gesendet wird:

            0x08,0,30,6,128,128,128,128,128,64,36

Diese Eingabe erreicht: Die Definition wird an Position #0 abgelegt.
Sendet täglich um 6:30 an den TWI-Slave 64 die Message 36.
Sinn z.B.: Einen morgentlichen Weckruf an einen Slave absetzen, der als Wecker arbeitet und
ein akustisches Signal ausgibt.

Noch einen Trick gibt es: wenn Bit.7 im Modusbyte gesetzt ist, dann wird dieser Schaltvorgang nur einmalig ausgeführt - er wird nach seiner Ausführung gelöscht.

0x08,0x02,31,12,128,128,128,128,0x03,0x82

schaltet beim nächsten Zeitpunkt 12:31 den Schalter #3 aus
und löscht anschließend die Schaltdefinition aus dem Speicher.

Um einen Schaltvorgang zu setzen, sieht die Eingabe wie folgt aus:

TWI-Adr, #Schaltdefinition, Minute, Stunde, DOW, Tag, Monat, Jahr, Schalter, Modus

0x08,1,128,128,127,128,128,128,64,3

Legt die Definition an Position #1 ab, toggled zu jeder vollen Minute den Schalter 64, das ist PORTD.0 am Controller.
(Da DOW = 127 bei jeder Abfrage an jedem Wochentag erfüllt ist, wird in jeder Minute geschaltet). Diese Einstellung ist gut für Testzwecke geeignet.

Zu beachten ist, dass Schaltdefinitionen beginnend mit der Position #0 bis zur höchsten Position #63 abgearbeitet werden.
Alle Änderungen werden zunächst in ein Array geschrieben.
Am Ende wird die neue Schalterstellung mit der vorherigen verglichen.
Nur dann, wenn Veränderungen aufgetreten sind, wird eine Message an den/die Portexpander geschickt.
Wird ein und derselbe Pin innerhalb dieser Definitionen mehrfach geändert, dann überlebt nur die letzte Definition !

Wird eine fehlerhafte (nicht definierte) Speicheradresse angeben, dann wird der interne Pointer auf das Array TWI_MA_buf[] gestellt. Hier landen dann weitere Eingaben, sie können so nicht versehentlich Daten im Array time[][] ändern.

Diese Feature kann man nutzen, um den Sendepuffer des TWI-Masters auszulesen:

0x08, 0xFF adressiert einen nicht definierten Speicherbereich,
der Pointer wird auf den Beginn des TWI_MA_buf[] gestellt.

Wenn man nun 10 Byte ausliest, dann zeigen die beiden ersten Byte den Inhalt der letzten TWI-Sendung der Uhr, das 10. Byte den letzten Wert des TWSR Registers.
Der Inhalt des TWSR-Registers zeigt an, mit welchem Erfolg die letzte Sendung abgeschlossen wurde.
Die zugehörigen Codes stehen im Manual und in der Headerdatei TWI_master_slave.h.

Anmerkung:

Zum Entwickeln und Testen des Programmes war ein RS232 nach TWI Interface sehr hilfreich. Vom PC aus wird der TWI-Bus z.B. mit HTERM gefüttert und ausgelesen.

Als TWI-Portexpander kann man den PCF8574 oder eine Emulation auf Basis eines USI-TWI-Slaves verwenden.
Der ist preiswerter und lässt sich auf beliebige TWI-Adressen einstellen.

Eine Sonderform dieses USI-Slaves gibt bei mir die 'Musik' aus.

(M)eine Software zu den genannten drei Tools/USI-Slaves hatte ich vor einigen Zeit hier in der Codesammlung ins Netz gestellt.

Nachfolgend die Beschreibung vom Januar 09 (mit Korrekturen versehen)

Hardware: mega168, 4-Zeilen-LCD (optional), 3.686400 MHz XTAL, DCF77-Empfänger
Speicherbedarf: mit Sonnenstand ca. 9600 Byte (AVR_gcc 4.3.0)
Der DCF77-Empfänger ist an INT0 und INT1 angeschlossen !
Das Signal ist nicht invertiert (Ruhelevel high).
Durch Vertauschen der Interrupts INT0 und INT1 sollte es auch mit
dem invertierten DCF-Signal (Ruhelevel low) funktionieren.
Im Zweifelsfalle kann man einen Transistor zum Invertieren spendieren.

Implementiert sind:

DCF77-Uhr mit

- Überprüfung der DCF-Paritätsbits
- Plausibilätscheck der neuen DCF-Minute mit der aktuellen Minute
  (die aktuelle Minute wird incrementiert und mit dem neuen DCF-Signal verglichen).
  Bei einem Fehler wird das letzte DCF-Signal mit dem neuen verglichen:
  Unterscheiden sich nur die Minuten um genau 1 Minute, dann wird die DCF77-Zeit
  übernommen.
- Ist immer noch ein Fehler erkannt, wird solange durch den internen XTAL synchron-
  isiert, bis wieder ein gültiges Signal empfangen wird.
- Die interne Uhr kalkuliert alle Zeitüberläufe, die Sommerzeit und die Schaltjahre,
  und zwar nach Bedarf vorwärts und rückwärts.
- Fehler werden im Display angezeigt (die Uhrzeit natürlich auch !)
- 63 Alarmzeiten sind über TWI programmierbar (Auflösung: 1 Minute):

  Es sind 70 Pins zum Schalten vorgesehen: 6 davon am ATMEGA an PORTD,

  der Rest ist via TWI etwa auf einem PCF 8574 (Portexpander) ansprechbar.

  Zu jedem Zeitpunkt lässt sich einer der freigegebenen Portpins ein-/ausschalten oder
  togglen.
- Die Anzahl der 'SchaltSlots' ist z.Z. auf 63 beschränkt, damit die Schaltdefinitionen
  im Eeprom gespeichert und bei einem Neustart neu geladen werden können.

Anzeige

Die Anzeige der Daten erfolgt auf einem 4-zeiligen LCD.
Die Routinen dazu stammen von Peter Fleury.

Alarm/Schaltzeiten

Es können Schaltzeiten via TWI programmiert werden zu denen jeweils ein auswählbarer
Portpin

- gesetzt, gelöscht oder getoggled wird.

Dazu sind alle verfügbaren Zeitinformation als Maske programmierbar:

- Minuten, Stunden, Wochentag, Tag, Monat, Jahr
- Felder, die nicht geprüft werden sollen, werden mit Bit.7 (128) markiert

Es sind 63 dieser 'Slots' vorhanden, die unabhängig voneinander schalten können.

Sonnenstandsberechnung

Eine Funktion ermittelt für die Ortszeit Azimut und Elevation der Sonne.
Sonnenaufgang und -untergang werden ungefähr ermittelt (durch Iteration).
Die Daten werden im LCD angezeigt und lassen sich über den TWI-Slave auslesen.
Die Ortskoordinaten können z.Z. nur im Quellcode verändert werden (sun.h).
Den Rechenweg habe ich hier im Forum gefunden (Hinweis in sun.c).
Die Werte stimmen gut mit denen überein, die das Programm orbitron.exe (www.stoff.pl)
ermittelt.

TWI-Master

Im Normalfall wird im Slave-Modus gearbeitet.
So können die Schaltdefinitionen von einem Master auf die Uhr als Slave übertragen werden.

Wenn die Uhr jedoch Schaltvorgänge auf den Schaltern 0 .. 63 ausführen soll, dann wird in den Master-Modus gewechselt und an einen (hoffentlich) angeschlossenen Slave gesendet.

TWI_slave

- Die Grundlage für den TWI-Zugang sind die Beispielroutinen von atmel (AVR311).
- Der mega8 ist als interruptgesteuerter TWI-Slave konfiguriert.
- Die TWI-Adresse ist im Programm auf 8 'verdrahtet' (siehe TWI_slave_d.h)
  Kann aber geändert werden durch die Eingabe:

  0x08, 0xFB, 0xaa -> aktuelle TWI_ADR, 0xF5, neue Adresse.

  Von nun an hört der Slave auf 0xaa.

  Die neue Adresse wird im Eeprom abgelegt und auch nach einem Reset neu geladen.

  Durch Auskommentieren von #define TWI_ADR_CHANGE wird diese Option entfernt.

- entfallen .....
  Er stellt seinen nicht benötigtes SRAM als zum Schreiben und
  Lesen über TWI zur Verfügung (vergleichbar mit dem RAM des PCF8583),
  verfügbar sind z.Z. 768 Byte -> diese Option ist entfallen !!

- Via TWI kann der Speicher der Uhr gelesen und beschrieben werden.
  Alle Zeit- und Schaltdaten sind in einem 2-dimensionalen Array abgelegt.
  Es kann ein interner Pointer auf jedes dieser Array gesetzt werden.
  Danach können alle Daten gelesen und geschrieben werden.

  Um einen definierten Speicherbereich zu lesen, muss einmalig der Pointer positioniert werden:

  0x08, Speicherbereich (TWI-Adresse, Speicherbereich)

- Über TWI können so Daten aus dem Speicher der DCF-Uhr ausgelesen (und beschrieben werden) für:
- die DCF77-Bits der aktuellen Minute (von Bit.0 bis Bit.58),
- die DCF-Zeit der aktuellen Minute (ohne Sekunde verständlicherweise),
- das Zeitarray mit der aktuellen Uhr-Zeit einschl. der Sekundenangabe
- sowie der UTC-Zeit (benötigt für die Berechnung des Sonnenstandes),
- die Alarmregister mit den Zeitdaten,
- die Sonnendaten mit Azimut, Elevation sowie Sonnenaufgang/untergang
- (jede public-Variable kann man zugänglich machen - wenn man will)
- Aber Vorsicht: die Einhaltung von Arraygrenzen wird nicht geprüft,
  (insbesondere) beim Schreiben kann man das Programm auch Abschießen.

So sieht die Anzeige auf dem LCD (z.Zeit) aus:

```
Fr 02.01.09 12:11.09  -> Zeile 1 Datum und Uhrzeit
_____________cet_____  -> Zeile 2 kein Fehler, Winterzeit
     +15.11 07:52     -> Zeile 3 Elevation, Sonnenaufgang
     177.82 16:49     -> Zeile 4 Azimut, Sonnenuntergang

Fr 02.01.09 12:11.09  -> Zeile 1 Datum und Uhrzeit
_____________er:FF___  -> Zeile 2 Fehler (Fehlercode für: kein Signal)
xx   +15.11 07:52     -> Zeile 3 Elevation, Sonnenaufgang
02   177.82 16:49     -> Zeile 4 Azimut, Sonnenuntergang (02 = kumulierte Fehlerzahl)
```

xx -> wenn in den Zusatzbits des DCF-Signals Ankündigungsbits gesetzt sind, dann werden alle Zusatzbits (incl. Sommer/Winterzeit) hexadezimal angezeigt.

Die kumulierte Fehlerzahl (hexadezimal) wird um Mitternacht auf 0 gesetzt.

```
Sa 02.05.09 12:11.09  -> Zeile 1 Datum und Uhrzeit
a:_2________cest____  -> Zeile 2 Alarm programmiert, kein Fehler, Sommerzeit
     +15.11 07:52     -> Zeile 3 Elevation, Sonnenaufgang
     177.82 16:49     -> Zeile 4 Azimut, Sonnenuntergang
```

a: 2 -> zeigt, dass 2 Alarm/Schaltdefinitionen aktiv sind.

Fehlercodes

Wenn während der Paritätsprüfung und Prüfung der Signaldauer/Impulsabstände Fehler erkannt werden, dann wird für diese Fehler ein definiertes Bit in Error gesetzt.

Die Bitnummern sowie die Fehler sind in dcf_77.h beschrieben.

Die Fehlerbits 'summieren' sich innerhalb einer Minute möglicherweise auf, so dass mehrere Bits gesetzt sein können.

Zusätzlich werden zwei fatale Fehler mit festen Werten codiert:

0xFF - innerhalb von 250ms wurde kein neuer Signalanfang erkannt (fehlendes Signal)
0xFE - der Plausibilitätscheck (s.unten) ist fehlgeschlagen.

Die Zusatzbits des DCF-Signals sind binär codiert:
Bit.7 - Bit.5 ist immer 0, dann folgen ab Bit.4 - Bit.0: A1, A2, R, Z1, Z2.

Der Plausibilitätscheck

In der 59. Sekunde einer Minute wird das DCF-Signal (der neuen, nächsten Minute) decodiert und in das Array time[DCF][] geschrieben.

Im Array time[CLK][] steht die aktuelle Zeit, die auch im LCD angezeigt wird.

1. Test:
Vorab wird eine Kopie der DCF-Zeit der laufenden Minute in time[BAK][] gespeichert.
Die aktuelle Zeit wird nach time[CHK][] kopiert, um 1 Minute incrementiert und mit time[DCF][] der künftigen Minute verglichen.

Sind beide gleich, dann die neue Zeit als korrekt angesehen und übernommen.

Falls dieser Test fehlschlägt:

2. Test:
Dann wird das neue DCF_Signal mit dem der letzten Minute aus time[BAK][] verglichen:
Unterscheiden sich beide nur genau 1 Minute, dann wird die DCF-Zeit übernommen.

Unterscheiden sich beide um mehr als 1 Minute, dann wird die DCF-Zeit als fehlerhaft eingestuft (was aber bei Überläufen nicht wirklich sein muss) und die weitere Synchronisation dem Quarztakt überlassen.

In der nächsten Minute wiederholt sich der Test (und gelingt dann hoffentlich).

Ziel der Übung ist:

Auch bei einem Programmfehler (!) soll sich die Uhr wieder die DCF-Zeit einstellen, wenn zwei aufeinanderfolgende Zeiten plausibel sind.

Da mir nicht bekannt ist, ob dieses Projekt überhaupt irgendjemanden interessiert (schließlich hat vermutlich jeder von euch mit DCF77-Uhren, RC5-Sendern/Empfängern begonnen ..) beende ich hier weitere Beschreibungen - denn ich weiß ja, wie’s funktioniert bzw. funktionieren sollte.

Und die Beschreibung diente mir letztlich selbst dazu, verquerte/unlogische Denkansätze zu erkennen und zu beseitigen.

Michael S.

// --- [eof] ---