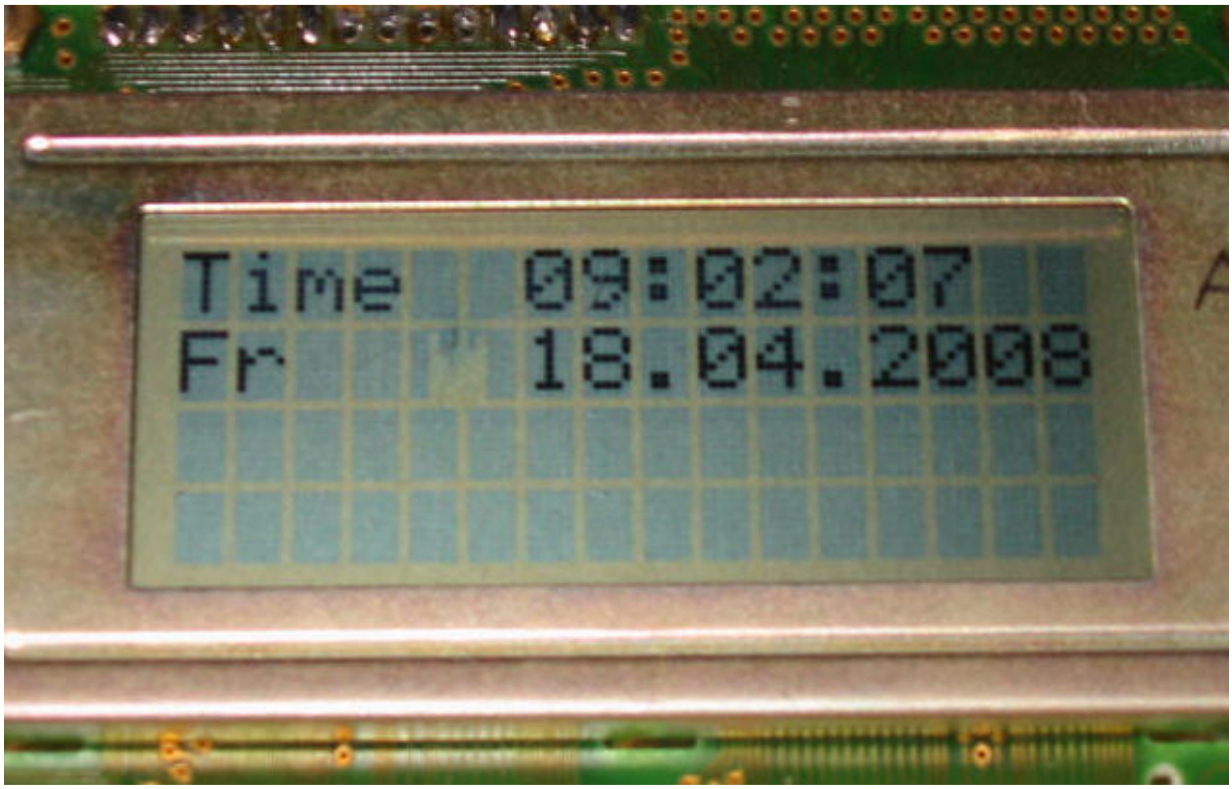
Unterrichtsprojekt: DCF77-Funkuhr



Unter Verwendung eines für ca. 12 Euro käuflichen DCF77-Empfänger-Moduls der Firma Conrad erfolgt die Auswertung der übertragenen Zeit- und Datumsinformationen mit unserem ATMEL-ATM1- Board.

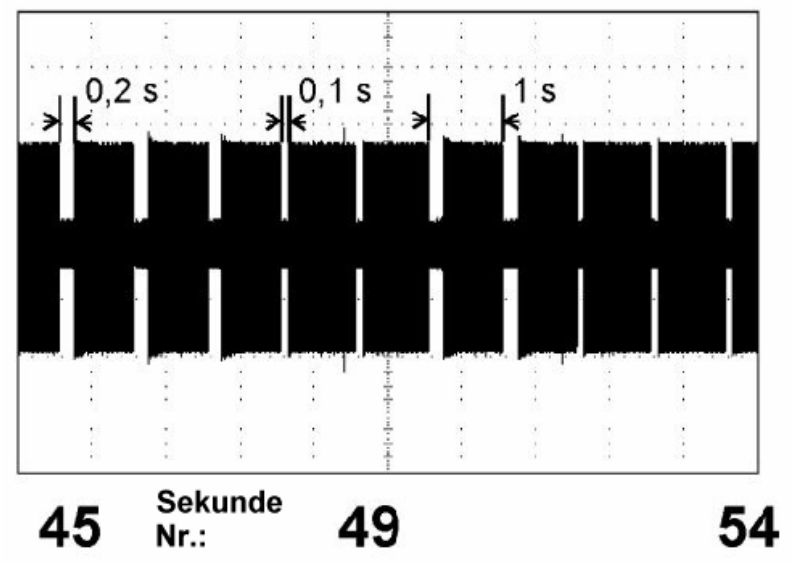
Der Empfänger wird dabei über ein Flachbankkabel und einen Pfostenfeldstecker mit Port P1 des Controllers verbunden und erhält so die an den Portsteckern zur Verfügung gestellte Systemspannung von +5V.

Der Ausgang (hier: „Ausgang – normal“) wird über einen Widerstand von 5,6 Kilo-Ohm mit der Betriebsspannung verbunden. Direkt an der „Ausgang-normal“-Klemme wird Portpin P1^0 angeschlossen und liest damit die Sekundenimpulse der Funkuhr ein.

1. „DCF77“ -- was steckt dahinter?

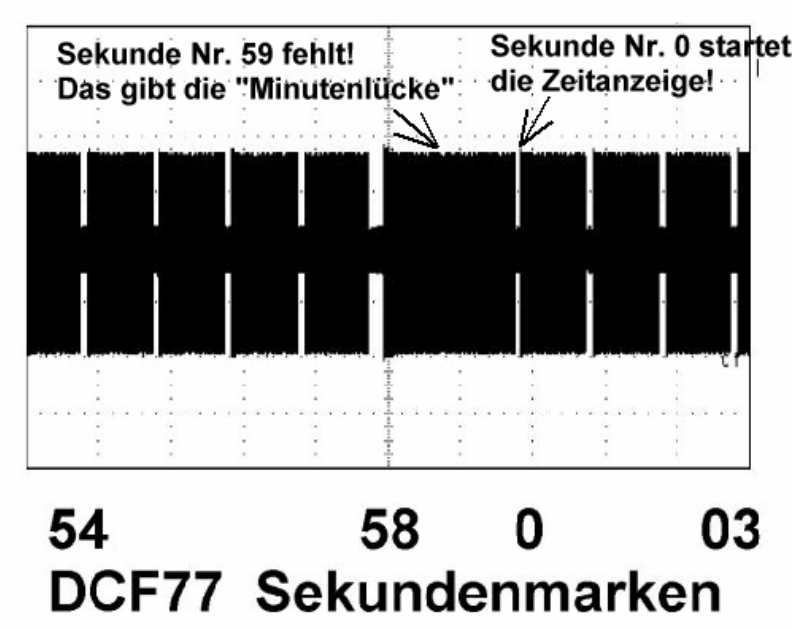
Die PTB (= Physikalisch – Technische Bundesanstalt) erzeugt über ein Cäsium-Normal eine sehr exakte Frequenz. Aus dieser Cäsiumfrequenz wird nicht nur eine „Normalfrequenz“ gewonnen, sondern auch eine äußerst präzise Zeitinformation generiert.

Basierend auf diesen beiden Informationen steuert man einen Langwellensender in Mainflingen (im Taunus, nahe Frankfurt) und versieht das über eine Antennenanlage abgestrahlte Signal zusätzlich mit einer Amplitudenmodulation, in der die Zeit und das Datum übermittelt werden (= ASK = Amplitude Shift Keying mit 1 Bit / Sekunde = 1 Baud).



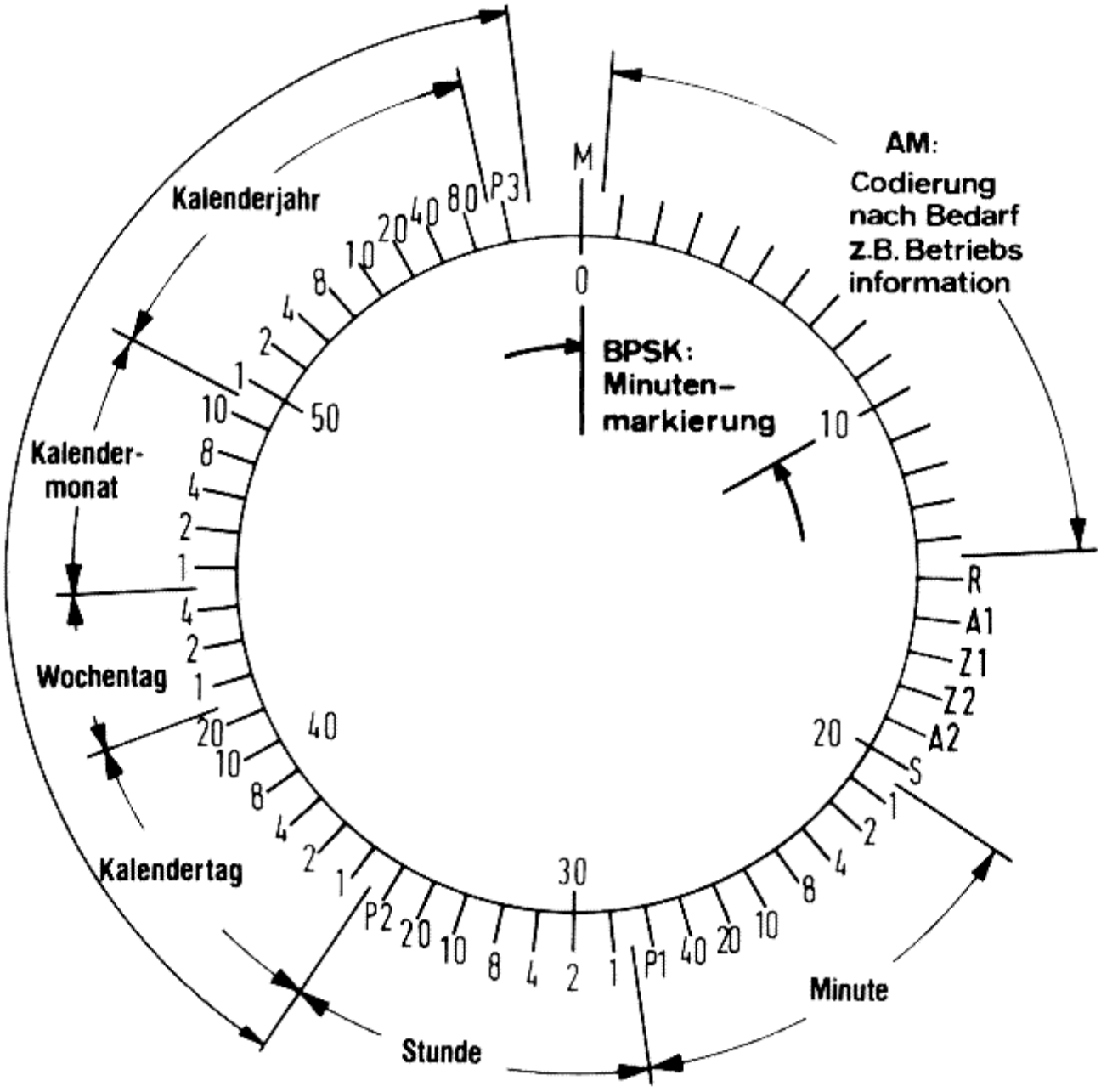
Die Sendefrequenz beträgt 77,5 kHz, die maximale Sendeleistung (PEP = Peak Envelope Power) 50 kW. Bei jeder vollen Sekunde wird die Trägerspannung auf 25% reduziert. Diese Reduktion dauert entweder 100ms (= log. Null) oder 200 ms (= log. Eins).

Im BCD-Code werden nun nacheinander (mit 1 Baud) die aktuelle Minute und Stunde, der aktuelle Wochentag und das komplette Datum übertragen. Alles mit einer Minute Vorlauf, denn erst zur nächsten vollen Minute erfolgt die gültige Anzeige. Dabei muss natürlich zuerst die nächste volle Minute über folgenden Trick erkannt werden:



Sekunde Nr. 59 wird NICHT markiert und diese „Minutenlücke“ ermöglicht den Empfängern die Synchronisation. (Diese 59. Sekunde muss also von der Decoderschaltung ergänzt werden!).
Bei der dann folgenden „Sekunde Nr. Null“ erfolgt die neue Zeitanzeige sowie die Ausgabe des Wochentages und des Datums.

Sendeschema für eine volle Minute:



Bei unserem Projekt wird nur die Zeit- und Datumsinformation ausgewertet. Das beginnt bei Sekunde 21 mit der „Einer-Stelle“ der im BCD-Code übermittelten Minutenangabe.

Auf die Übertragung der Minute bzw. Stunde folgt jeweils ein Prüfbit.

Bei der Wochentags-Information steht die „1“ für Montag.

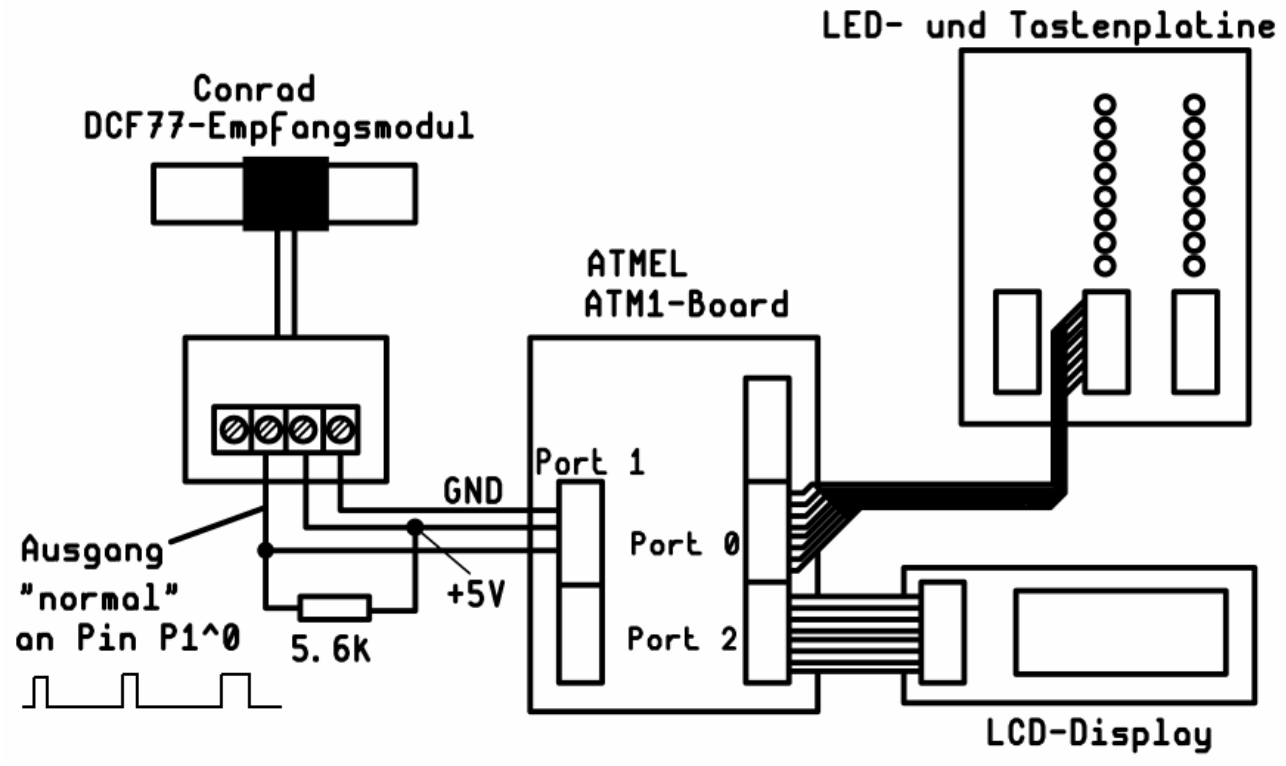
Beim Kalenderjahr werden nur die beiden letzten Stellen gesendet (z. B. „08“ für 2008).

Abgeschlossen wird diese „vorlaufende Übertragung“ durch das Prüfbit P3 und die Minutenlücke.

Hinweis: Sehr ausführliche Informationen finden sich im Internet, speziell in der Homepage der Physikalisch Technischen Bundesanstalt (PTB). Einfach in die Suchmaschine „DCF77“ eingeben!

2. Das Funkuhr-Projekt

2.1. Übersichtsschaltplan



2.2. Programmkonzept

Die Sekundenimpulse der Funkuhr werden über Portpin P1^0 in das Controllerboard eingelesen und direkt in Portpin P0^5 kopiert, an den eine Kontroll-LED angeschlossen ist. Zur Kontrolle werden die Sekundenimpulse, mit denen dann das Auswerteprogramm arbeitet, an einer weiteren LED (P0^7) ausgegeben -- so ist z. B. leicht zu überprüfen, ob die Ergänzung der (nicht übertragenen 59. Sekunde) funktioniert hat.

Das Auswerteprogramm gliedert sich in zwei Phasen, die ebenfalls durch LEDs (Phase 1 = LED an Pin P0^0 leuchtet, Phase 2 = LED an Pin P0^1 leuchtet) angezeigt werden:

In **Phase 1** sucht das Programm nach der Minutenlücke (= fehlender 59. Sekundenimpuls). Ist diese Lücke erkannt, beginnt der Sekundenzähler im Display zu arbeiten, die „Phase 1 – LED“ erlischt und die „Phase 2 – LED“ leuchtet auf.

In **Phase 2** werden nicht nur die ankommenden einzelnen Sekunden gezählt und sofort auf dem Display angezeigt. Zusätzlich werden ab Sekunde 21 **die „0/1“-Informationen** der im BCD-Code ausgestrahlten Minuten, Stunden, Wochentage, Kalendermonate usw. in passenden **bitadressierbaren Variablen** gespeichert. Wird die Minutenlücke erreicht, dann wird die fehlende Sekunde ergänzt und bei der darauf folgenden „Sekunde Null“ die komplette Zeit- und Datumsanzeige ausgelöst.

Zusätzlich sind im Programm noch einige Kontrollmechanismen eingebaut, um durch Störungen ausgelöste Fehlanzeigen (Beispiel: mehr als 60 Sekunden gezählt ….) zu erkennen. In solchen Fällen kehrt das Programm in Phase 1 zurück.

2.3. C-Programm „dcf77_atmel_03.c“

```
/*Decoderprogramm für eine DCF-77-Funkuhr unter Verwendung des ATMEL ATM1 -
Boards.

Das Funkuhr-Signal wird an Portpin P1^0 angelegt und das eingelesene Signal
an Portpin P0^5 zur Kontrolle mit einer LED angezeigt. An Pin P0^7 zeigt
eine weitere LED das für die Decodierung verwendete Signal mit beseitigter
Minutenlücke (zur Kontrolle der korrekten Ergänzung bei Sekunde 59).

Port P2 dient zur Ansteuerung des Displays.

An Port P0 hängt die LED-Kette. Die LEDs an den Pins P0^1 und P0^2
signalisieren die Startphase bzw. den Übergang in den Dauerbetrieb.

Das Funkuhrsignal ist auf HIGH-Pegel für 0,1 bzw. 0,2 Sekunden, für denRest
der Sekunde dagegen auf LOW.  Eine Dauer von 0,1 Sekunden stellt eine log.
Null, eine Dauer von 0,2 Sekunden dagegen eine log. Eins dar.

Es wird ein 11,059MHz-Quarz beim Board eingesetzt.

Programmiert durch G. Kraus am 22.04.2008

Programmname: dcf77_atmel_03.c*/

#include<stdio.h>
#include<t89c51ac2.h>         // Header für AT89C51AC3

void zeit_s(void);            // Display-Prototypen
void zeit_l(void);
void lcd_com(void);
void lcd_ein(void);
void lcd_ini1(void);
void switch_z1(void);
void switch_z2(void);
void show_text(char *ptr);

void detect_bit(void);            // Prototypen für verwendete Funktionen
void fill_Sec_59(void);
void Zeitanzeige(void);
void Datumsanzeige(void);
void Conv_Second(unsigned char a);
void del_10ms(unsigned char delay);
void ISR_Timer1(void);
void Conv_Date(void);
void Conv_Time(void);

sbit PULS_EING=P1^0;      // Funkuhrsignal (LOW-aktiv) an Portpin P1^0

sbit SEK_LED=P0^7;        // Sekunden-LED an P0.5
sbit PULS_LED=P0^5;           // Eingangssignal-Kontrolle
sbit LED_PH1=P0^0;        // LED an P0.0: Startphase
sbit LED_PH2=P0^1;        // LED an P0.1: Phase 2 (1. Minutenlücke erkannt)

bit  phase_1;             // Flag für Phase 1
bit  phase_2;             // Flag für Phase 2

data unsigned char TIME_TICK=0;   // Timer-Ticker (10ms - Intervalle)
data unsigned char SEK_TICK=0;    // Sekundenzähler
data unsigned char SEK_TICK_OLD;  // Sekundenspeicher für Fehlersicherheit
```

```
bdata unsigned char MIN_EINS=0;       // Einer-Stelle der Minutenanzeige
bdata unsigned char MIN_ZEHN=0;       // Zehner-Stelle der Minutenanzeige

bdata unsigned char STD_EINS=0;       // Einer-Stelle der Stundenanzeige
bdata unsigned char STD_ZEHN=0;       // Zehner-Stelle der Stundenanzeige


bdata unsigned char TAG_EINS=0;       // Einer-Stelle des aktuellen Tages
bdata unsigned char TAG_ZEHN=0;       // Zehnerstelle des aktuellen Tages

bdata unsigned char WOCH_TAG=1;       // Wochentag (1 = Montag)

bdata unsigned char MON_EIN=0;        // Einer-Stelle des Monats
bdata unsigned char MON_ZEH=0;        // Zehner-Stelle des Monats

bdata unsigned char JAHR_EIN=0;       // Einer-Stelle des Jahres
bdata unsigned char JAHR_ZEH=0;       // Zehner-Stelle des Jahres

code unsigned char wochent[21]=      {"MoDiMiDoFrSaSo      "};
code unsigned char start_time[21]=   {"Time  xx:xx:xx      "};
code unsigned char start_date[21]=   {"Mo    xx.xx.20xx    "};
unsigned char time_array[21]=        {"Time  xx:xx:xx      "};
unsigned char date_array[21]=        {"Mo    xx.xx.20xx    "};
/*------------------------------------------------------------*/

void main(void)
{   P0=0x00;              // Alle LEDs aus
    P1=0xFF;              // P1 auf "Lesen" schalten
    TMOD=0x10;            // Timer1 im 16 Bit-Betrieb (Mode 1)
    TL1=0xFF;             // Reload-Wert für 11,059MHz-Quarz
    TH1=0xDB;             // ist 0xDBFF für 10ms
    EA=1;                 // Alle Interrupts freigeben
    ET1=1;                // Timer1 - Interrupt freigeben
    TF1=0;                // Timer1-Überlaufflag löschen
    TR1=0;                // Timer1 ausschalten

    lcd_ein();               // LCD-Einschaltroutine
    lcd_ini1();              // Einstellung der LCD-Betriebswerte
    switch_z1();             // Schalte auf Zeile 1
    show_text(start_time);   // Zeige Starttext der Zeitanzeige
    switch_z2();             // Schalte auf Zeile 2
    show_text(start_date);   // Zeige Starttext der Datumsanzeige
    LED_PH1=1;               // Phase 1 beginnt
    phase_1=1;               // ""   "
    LED_PH2=0;               // Phase 2 OFF
    phase_2=0;
    AUXR=AUXR&0xFD;          // Auf internes ERAM umschalten

    while(1)
    {   while(phase_1)               // Minutenlücke erkennen
        {   switch_z1();             // Starttext nach Rückkehr
            show_text(start_time);   // aus der Fehlererkennung zeigen
            switch_z2();
            show_text(start_date);

            while(PULS_EING==1)      // Auf LOW-Pegel warten
            {   SEK_LED=1;           // Sekunden-LED ON
                PULS_LED=PULS_EING;  // LED zeigt Eingangssignal
            }

            TIME_TICK=0;     // TIME_TICK soll 10ms-Einheiten zählen
```

```
    TR1=1;           // Timer starten, um LOW-Dauer zu messen

    while(PULS_EING==0)       // LOW-Ende abwarten
    {    SEK_LED=0;           // Sekunden-LED ausschalten
         PULS_LED=PULS_EING;  // LED = Eingangsssignal
    }

    SEK_LED=1;                // LOW ist vorbei, HIGH erreicht

    TR1=0;                    // Timer stoppen

    if(TIME_TICK>105)         // wenn 1,05 Sekunden vergangen:
    {   phase_1=0;            // Phase_1 - Bit ausschalten
        phase_2=1;            // Phase_2 - Bit einschalten
        LED_PH2=1;            // zugehörige LEDs schalten
        LED_PH1=0;
        SEK_TICK=0;           // Sekunde "NULL" zählen
        Conv_Second(SEK_TICK);    // in ASCII konvertieren
        Zeitanzeige();            // und anzeigen

        while(PULS_EING==1)       // HIGH-Ende abwarten
        {   SEK_LED=1;            // Sekunden-LED ausschalten
            PULS_LED=PULS_EING;   //LED = Eingangssignal
        }
    }
}

while(phase_2)                    // Dauerbetrieb
{    while (SEK_TICK<=57)         // Sekunde 58 noch nicht erreicht
    {   SEK_TICK_OLD=SEK_TICK;    // Sekundenstand speichern
        while(PULS_EING==0)       // Eingang auf LOW, deshalb
        {   SEK_LED=0;            // LOW-Ende abwarten
            PULS_LED=PULS_EING;   // LED = Eingangssignal
        }

        SEK_TICK++;      // HIGH kommt, deshalb Sekunde zählen

        PULS_LED=PULS_EING;   // LED = Eingangssignal
        TR1=1;                // Timer starten
        SEK_LED=1;            // Sekunden-LED einschalten

        Conv_Second(SEK_TICK);    // in ASCII umcodieren
        Zeitanzeige();            // auf dem Display anzeigen
        while(PULS_EING==1)   // Eingang noch auf HIGH, deshalb
        {    SEK_LED=1;       // auf nächstes LOW warten
             PULS_LED=PULS_EING;      // LED = Eingangssignal
        }
        TR1=0;                // LOW da = Timer stoppen
        detect_bit();         // Bit auswerten
        TIME_TICK=0;     // 10ms-Zählerstand wieder auf Null

        SEK_LED=0;            // Sekunden-LED ausschalten
        PULS_LED=PULS_EING;               // LED = Eingangssignal
        if(SEK_TICK - SEK_TICK_OLD!=1)    // Kontrolle: nur 1
                                          // Sekunde
        {   phase_2=0;                    // gezählt? Wenn
                                          // nicht, dann raus
            phase_1=1;                    // aus Phase 2!
            break;
        }
    }
```

```
            fill_Sec_59();        // Ergänze die 59. Sekunde und
        }                         // zeige Zeit und Datum an
    }
}
//---------------------------------------------------------------

void ISR_Timer1(void) interrupt 3     // Timer1 hat Interruptvektor 1B
{
    TF1=0;                    // Überlaufflag löschen
    TIME_TICK++;              // Inkrementiere Timer-Ticker im 10ms-Takt

    TH1=0xDC;                 // Reloadwert ist 0xDBFF für 11,059MHz-Quarz
    TL1=TL1-1;                // und 10ms bis zum Überlauf
}
//---------------------------------------------------------------

void del_10ms(unsigned char delay)    // Verzögerung in 10ms-Einheiten
{
    TIME_TICK=0;                  // Einheitenzähler auf Null
    TR1=1;                        // Zähler starten
    while(TIME_TICK<delay);       // Verzögerung ablaufen lassen
    TR1=0;                        // Zähler stoppen
    TIME_TICK=0;                  // Einheitenzähler wieder auf Null
    TL1=0xFF;                     // Reload-Wert für 11,059MHz-Quarz
    TH1=0xDB;                     // ist 0xDBFF für 10ms bis zum überlauf
}
//---------------------------------------------------------------

void fill_Sec_59(void)            // Ergänzung der 59. Sekunde und Anzeige
{       del_10ms(95);                 // 950ms Verzögerung
        if((SEK_TICK==58)&&(PULS_EING==0))    // Ist das wirklich die
                                              // Minutenlücke?
        {   SEK_LED=1;            // Wenn ja: Sekunden-LED an und 59.
            Sekunde
                                  // zählen
            SEK_TICK=59;
            Conv_Second(SEK_TICK);    // Sekunden in ASCII konvertieren und
            Zeitanzeige();            // Zeit bei 59. Sekunde anzeigen
            del_10ms(20);             // 200ms Pulslänge bei der ergänzten
            SEK_LED=0;                // 59. Sekunde, dann wieder LOW
        }
        else             // Wenn ein Fehler vorliegt: zurück zur Startphase
        {   phase_2=0;
            phase_1=1;
        }


        while(PULS_EING==0)       // Warte auf das Ende der Minutenlücke
        {   SEK_LED=0;            // Sekunden-LED ist OFF
            PULS_LED=PULS_EING;   // Kontroll-LED = Eingangssignal
        }

        SEK_LED=1;                // Sekunde Null erreicht
        PULS_LED=PULS_EING;       // Kontroll_LED = Eingangssignal
        SEK_TICK=0;               // Sekundenzähler auf Null
        Conv_Second(SEK_TICK);    // Sekunde in ASCII konvertieren
        Conv_Time();              // Komplette Zeit ermitteln
        Conv_Date();              // Komplettes Datum ermitteln
        Zeitanzeige();            // Zeit auf Display anzeigen
        Datumsanzeige();          // Datum auf Display anzeigne
        while(PULS_EING==1)       // Ende von HIGH bei Sekunde Null
                                  // abwarten
```

```
        {   SEK_LED=1;                // Sekunden-LED ist ON
            PULS_LED=PULS_EING;       // Kontroll-LED = Eingangssignal
        }

        SEK_LED=0;                    // Sekunden-LED ist OFF
        PULS_LED=PULS_EING;           // Kontroll-LED = Eingangssignal
}
//-----------------------------------------------------

void detect_bit(void)             // Bitauswertung bei jeder Sekunde
{       unsigned char ZER_ONE;    // Variable speichert erkannte Null oder
                                  // Eins

        if(TIME_TICK<=10)         // Pulslänge unter 100ms ergibt Null....
        {   ZER_ONE=0;
        }
        else                      // ...und mehr als 100ms ergeben Eins
        {   ZER_ONE=1;
        }

        if(SEK_TICK==21)                  // Bit von Sekunde 21 gibt
                                          // Minuten-Einer
        {   if(ZER_ONE==1)                // Falls "1" gesendet:
            {   MIN_EINS=MIN_EINS|0x01;   // Einser-Stelle setzen
            }
            else                          // sonst: löschen
            {   MIN_EINS=MIN_EINS&0x0e;
            }
        }

        if(SEK_TICK==22)                      // Bit von Sekunde 22 gibt
                                              // Minuten-Zweier
        {   if(ZER_ONE==1)                    // Falls "1" gesendet:
            {   MIN_EINS=MIN_EINS|0x02;       // Zweier-Stelle setzen
            }
            else                              // sonst: löschen
            {   MIN_EINS=MIN_EINS&0x0d;
            }
        }

        if(SEK_TICK==23)                      // Bit von Sekunde 23 gibt
                                              // Minuten-Vierer
        {   if(ZER_ONE==1)                    // Falls "1" gesendet:
            {   MIN_EINS=MIN_EINS|0x04;       // Vierer-Stelle setzen
            }
            else                              // sonst löschen:
            {   MIN_EINS=MIN_EINS&0x0b;
            }
        }

        if(SEK_TICK==24)                      // Bit von Sekunde 24 gibt
                                              // Minuten-Achter
        {   if(ZER_ONE==1)                    // Falls "1" gesendet:
            {   MIN_EINS=MIN_EINS|0x08;       // Achter-Stelle setzen
            }
            else                              //sonst: löschen
            {   MIN_EINS=MIN_EINS&0x07;
            }
            MIN_EINS=(MIN_EINS&0x0f) + 48;    // Ergebnis in ASCII
                                              // konvertieren
        }
```

```
if(SEK_TICK==25)         // Bit von Sekunde 25 gibt Minuten-Zehner
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   MIN_ZEHN=MIN_ZEHN|0x01;     // Zehner-Stelle setzen
     }
     else                            // sonst: löschen
     {   MIN_ZEHN=MIN_ZEHN&0x0e;
     }
}

if(SEK_TICK==26)       // Bit von Sekunde 26 gibt Minuten-Zwanziger
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   MIN_ZEHN=MIN_ZEHN|0x02;     // Zwanziger-Stelle setzen
     }
     else                            // sonst: löschen
     {   MIN_ZEHN=MIN_ZEHN&0x0d;
     }
}

if(SEK_TICK==27)       // Bit von Sekunde 27 gibt Minuten-Vierziger
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   MIN_ZEHN=MIN_ZEHN|0x04;     // Vierziger-Stelle setzen
     }
     else                            // sonst: löschen
     {   MIN_ZEHN=MIN_ZEHN&0x0b;
     }
     MIN_ZEHN=(MIN_ZEHN&0x07) + 48;     // Ergebnis in ASCII
                                        // konvertieren
}

if(SEK_TICK==29)          // Bit von Sekunde 29 gibt Stunden-Einer
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   STD_EINS=STD_EINS|0x01;     // Einer-Stelle setzen
     }
     else                            // sonst: löschen
     {   STD_EINS=STD_EINS&0x0e;
     }
}

if(SEK_TICK==30)         // Bit von Sekunde 30 gibt Stunden-Zweier
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   STD_EINS=STD_EINS|0x02;     // Zweier-Stelle setzen
     }
     else                            // sonst: löschen
     {   STD_EINS=STD_EINS&0x0d;
     }
}

if(SEK_TICK==31)         // Bit von Sekunde 31 gibt Stunden-Vierer
{    if(ZER_ONE==1)                  // Falls "1" gesendet:
     {   STD_EINS=STD_EINS|0x04;     // Vierer-Stelle setzen
     }
     else                            // sonst: löschen
     {   STD_EINS=STD_EINS&0x0b;
     }
}
```

```
if(SEK_TICK==32)        // Bit von Sekunde 32 gibt Stunden-Achter
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {    STD_EINS=STD_EINS|0x08;    // Achter-Stelle setzen
     }
     else                           // sonst: löschen
     {    STD_EINS=STD_EINS&0xf7;
     }
STD_EINS=(STD_EINS&0x0f) + 48;    // Ergebnis in ASCII konvertieren
}

if(SEK_TICK==33)         // Bit von Sekunde 33 gibt Stunden-Zehner
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {    STD_ZEHN=STD_ZEHN|0x01;    // Zehner-Stelle setzen
     }
     else                           // sonst: löschen
     {    STD_ZEHN=STD_ZEHN&0x0e;
     }
}

if(SEK_TICK==34)       // Bit von Sekunde 34 gibt Stunden-Zwanziger
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {    STD_ZEHN=STD_ZEHN|0x02;    // Zwanziger-Stelle setzen
     }
     else                           // sonst: löschen
     {    STD_ZEHN=STD_ZEHN&0x0d;
     }
     STD_ZEHN=(STD_ZEHN&0x03) + 48;    // Ergebnis nach ASCII
                                       // konvertieren
}

if(SEK_TICK==36)            // Bit von Sekunde 36 gibt Tag-Einer
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {    TAG_EINS=TAG_EINS|0x01;    // Einer-Stelle setzen
     }
     else                           // sonst: löschen
     {    TAG_EINS=TAG_EINS&0x0e;
     }
}

if(SEK_TICK==37)            // Bit von Sekunde 37 gibt Tag-Zweier
     {    if(ZER_ONE==1)             // Falls "1" gesendet:
     {    TAG_EINS=TAG_EINS|0x02;    // Zweier-Stelle setzen
     }
     else                           // sonst: löschen
     {    TAG_EINS=TAG_EINS&0x0d;
     }
}

if(SEK_TICK==38)            // Bit von Sekunde 38 gibt Tag-Vierer
{    if(ZER_ONE==1)                // Falls "1" gesendet:
     {    TAG_EINS=TAG_EINS|0x04;   // Vierer-Stelle setzen
     }
     else                          // sonst: löschen
     {    TAG_EINS=TAG_EINS&0x0b;
     }
}
```

```
if(SEK_TICK==39)             // Bit von Sekunde 39 gibt Tag-Achter
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   TAG_EINS=TAG_EINS|0x08;   // Achter-Stelle setzen
     }
     else                           // sonst: löschen
     {   TAG_EINS=TAG_EINS&0x07;
     }

     TAG_EINS=(TAG_EINS&0x0f) + 48;    // Ergebnis in ASCII
                                       // konvertieren
}

if(SEK_TICK==40)            // Bit von Sekunde 40 gibt Tag-Zehner
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   TAG_ZEHN=TAG_ZEHN|0x01;   // Zehner-Stelle setzen
     }
     else                           // sonst: löschen
     {   TAG_ZEHN=TAG_ZEHN&0x0e;
     }
}

if(SEK_TICK==41)          // Bit von Sekunde 41 gibt Tag-Zwanziger
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   TAG_ZEHN=TAG_ZEHN|0x02;   // Zwanziger-Stelle setzen
     }
     else                           // sonst: löschen
     {   TAG_ZEHN=TAG_ZEHN&0x0d;
     }
     TAG_ZEHN=(TAG_ZEHN&0x03) + 48;    // Ergebnis in ASCII
                                       // konvertieren
}

if(SEK_TICK==42)       // Bit von Sekunde 42 gibt Wochentag-Einer
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   WOCH_TAG=WOCH_TAG|0x01;   // Einer-Stelle setzen
     }
     else                           // sonst: löschen
     {   WOCH_TAG=WOCH_TAG&0x0e;
     }
}

if(SEK_TICK==43)       // Bit von Sekunde 43 gibt Wochentag-Zweier
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   WOCH_TAG=WOCH_TAG|0x02;   // Zweier-Stelle setzen
     }
     else                           // sonst: löschen
     {   WOCH_TAG=WOCH_TAG&0x0d;
     }
}

if(SEK_TICK==44)      // Bit von Sekunde 44 gibt Wochentag-Vierer
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   WOCH_TAG=WOCH_TAG|0x04;   // Vierer-Stelle setzen
     }
     else                           // sonst: löschen
     {   WOCH_TAG=WOCH_TAG&0x0b;
     }
WOCH_TAG=WOCH_TAG&0x07;         // Ergebnis in ASCII konvertieren
}
```

```
if(SEK_TICK==45)           // Bit von Sekunde 45 gibt Monat-Einer
{    if(ZER_ONE==1)                // Falls "1" gesendet:
     {   MON_EIN=MON_EIN|0x01;     // Einer-Stelle setzen
     }
     else                          // sonst: löschen
     {   MON_EIN=MON_EIN&0x0e;
     }
}

if(SEK_TICK==46)           // Bit von Sekunde 46 gibt Monat-Zweier
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   MON_EIN=MON_EIN|0x02;    // Zweier-Stelle setzen
     }
     else                         // sonst: löschen
     {   MON_EIN=MON_EIN&0x0d;
     }
}

if(SEK_TICK==47)           // Bit von Sekunde 47 gibt Monat-Vierer
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   MON_EIN=MON_EIN|0x04;    // Vierer-Stelle setzen
     }
     else                         // sonst: löschen
     {   MON_EIN=MON_EIN&0x0b;
     }
}

if(SEK_TICK==48)          // Bit von Sekunde 48 gibt Monat-Achter
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   MON_EIN=MON_EIN|0x08;    // Achter-Stelle setzen
     }
     else                         // sonst: löschen
     {   MON_EIN=MON_EIN&0x07;
     }
MON_EIN=(MON_EIN&0x0f) + 48; // Ergebnis in ASCII konvertieren

}

if(SEK_TICK==49)          Bit von Sekunde 49 gibt Monat-Zehner
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   MON_ZEH=MON_ZEH|0x01;    // Zehner-Stelle setzen
     }
     else                         // sonst: löschen
     {   MON_ZEH=MON_ZEH&0x0e;
     }
MON_ZEH=(MON_ZEH&0x01) + 48; // Ergebnis in ASCII konvertieren
}

if(SEK_TICK==50)          // Bit von Sekunde 50 gibt Jahr-Einer
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   JAHR_EIN=JAHR_EIN|0x01;  // Einer-Stelle setzen
     }
     else                         //   sonst: löschen
     {   JAHR_EIN=JAHR_EIN&0x0e;
     }
}
```

```
if(SEK_TICK==51)         // Bit von Sekunde 51 gibt Jahr-Zweier
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   JAHR_EIN=JAHR_EIN|0x02;  // Zweier-Stelle setzen
     }
     else                         // sonst: löschen
     {   JAHR_EIN=JAHR_EIN&0x0d;
     }
}

if(SEK_TICK==52)        // Bit von Sekunde 52 gibt Jahr-Vierer
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   JAHR_EIN=JAHR_EIN|0x04;  // Vierer-Stelle setzen
     }
     else                         // sonst: löschen
     {   JAHR_EIN=JAHR_EIN&0x0b;
     }
}

if(SEK_TICK==53)        // Bit von Sekunde 53 gibt Jahr-Achter
{    if(ZER_ONE==1)               // Falls "1" gesendet:
     {   JAHR_EIN=JAHR_EIN|0x08;  // Achter-Stelle setzen
     }
     else                         // sonst: löschen
     {   JAHR_EIN=JAHR_EIN&0x07;
     }
JAHR_EIN=(JAHR_EIN&0x0f) + 48;    // Ergebnis in ASCII konvertieren
}

if(SEK_TICK==54)             // Bit von Sekunde 54 gibt Jahr-Zehner
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   JAHR_ZEH=JAHR_ZEH|0x01;    // Zehner-Stelle setzen
     }
     else                           //sonst: löschen
     {   JAHR_ZEH=JAHR_ZEH&0x0e;
     }
}

if(SEK_TICK==55)          // Bit von Sekunde 55 gibt Jahr-Zwanziger
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   JAHR_ZEH=JAHR_ZEH|0x02;    // Zwanziger-Stelle setzen
     }
     else                           // sonst: löschen
     {   JAHR_ZEH=JAHR_ZEH&0x0d;
     }
}

if(SEK_TICK==56)          // Bit von Sekunde 56 gibt Jahr-Vierziger
{    if(ZER_ONE==1)                 // Falls "1" gesendet:
     {   JAHR_ZEH=JAHR_ZEH|0x04;    // Vierziger-Stelle setzen
     }
     else                           // sonst: löschen
     {   JAHR_ZEH=JAHR_ZEH&0x0b;
     }
}
```

```
        if(SEK_TICK==57)        // Bit von Sekunde 57 gibt Jahr-Achtziger
        {   if(ZER_ONE==1)                // Falls "1" gesendet:
            {   JAHR_ZEH=JAHR_ZEH|0x08;   // Achtziger-Stelle setzen
            }
            else                          // sonst: löschen
            {   JAHR_ZEH=JAHR_ZEH&0x07;
            }
        JAHR_ZEH=(JAHR_ZEH&0x0f) + 48; // Ergebnis nach ASCII konvertieren
        }
}
//-------------------------------------------------------

void Conv_Second(unsigned char a) // Konvertiert Sekunden nach ASCII
{   unsigned char x;
    x=a;                              // Kopie anfertigen
    time_array[13]=x%10 +0x30;        // Sekunden-Einer konvertieren
    x=x/10;
    time_array[12]=x%10 +0x30;        // Sekunden-Zehner konvertieren
}
//-------------------------------------------------------

void Conv_Date(void)      // Legt das Datum-Array für das Display an
{   date_array[15]=JAHR_EIN;              // Jahres-Einer
    date_array[14]=JAHR_ZEH;              // Jahres-Zehner
    date_array[10]=MON_EIN;               // Monats-Einer
    date_array[9]=MON_ZEH;                // Monats-Zehner
    date_array[7]=TAG_EINS;               // Tages-Einer
    date_array[6]=TAG_ZEHN;               // Tages-Zehner
    date_array[0]=wochent[(WOCH_TAG-1)*2];    // Wochentage von Mo bis Fr
    date_array[1]=wochent[(WOCH_TAG-1)*2+1];
}
//-------------------------------------------------------

void Conv_Time(void)              // Legt das Zeit-Array für das Display an
{   time_array[10]=MIN_EINS;      // Minuten-Einer
    time_array[9]=MIN_ZEHN;       // Minuten-Zehner
    time_array[7]=STD_EINS;       // Stunden-Einer
    time_array[6]=STD_ZEHN;       // Stunden-Zehner
}
//-------------------------------------------------------

void Zeitanzeige(void)            // zeigt die Zeit auf dem Display an
{       switch_z1();
        show_text(time_array);
}
//-------------------------------------------------------

void Datumsanzeige(void)          // zeigt das Datum auf dem Display an
{       switch_z2();
        show_text(date_array);
}
//-------------------------------------------------------
```