

```

#####
;#          Attiny44 LCD-Mini-Funkuhr - Firmware zu Platine TN20140914          #
;# Ressourcen:                                                                    #
;# ATtiny44A @4.096MHz, DCF77-Empfänger an PB2, LCD 2x16 im 4 Bit Modus an Port A #
;# Fusebits: Low=FDh, High, Ext, Lock=FFh. Firmware erstellt mit AVR-Studio 4.17 #
;# Mit DCF77 "1-Level-Dekoder", der keine Absenkungen sondern Signalausfälle abfragt #
;# Zum "Einsammeln" der Daten dient ein Software-Schieberegister.                #
;#                                                                                #
;# V1.01: 02/05/2015 - Initiale Version (c) T.Neveling                        Code: 880 Byte #
;# V1.02: 23/03/2016 - Code Optimierungen bei gleicher Funktion                840 Byte #
;# V1.03: 10/01/2017 - Quarztakt mit DCF-Impuls synchronisiert                  842 Byte #
;# V1.04: 18/08/2018 - Verbesserte Fehlererkennung (Antennenausfall)           846 Byte #
#####
.include "tn44Adef.inc"                ; Definitionsdatei für ATtiny44A CPU einbinden
;
.dseg                                ; Datenbereich im SRAM
.org      SRAM_START
SEK:      .BYTE 1                    ; Zeit/Datum zur Ausgabe durch die interne Uhr
MIN:      .BYTE 1                    ; werden mit den dcf xxx-Variablen (s.u.) bei
STU:      .BYTE 1                    ; erfolgreicher Synchronisation überschrieben
TAG:      .BYTE 1
WTAG:     .BYTE 1
MON:      .BYTE 1
JHR:      .BYTE 1
;
; Register-Zuordnung
.def      P1      = r1                ; Parity-Register
.def      P2      = r2
.def      P3      = r3
.def      shifter = r4                ; Schieberegister für Daten aus Zeit-Telegramm
.def      dcf min = r5                ; aktuell empfangene Werte des Zeit-Telegramms
.def      dcf stu = r6
.def      dcf tag = r7
.def      dcf wtag = r8
.def      dcf mon = r9
.def      dcf jhr = r10
.def      dcf TZ  = r11                ; Timezone = 0 bei MEZ, = 2 bei MESZ
.def      TZ      = r12                ; dto für interne Uhr: 2 Inkremente zur Validierung
.def      NULL    = r13                ; immer = 0
.def      intreg  = r14                ; Zwischenspeicher für SREG
.def      temp1   = r16                ; Universalvariablen (ab hier erweiterte Registernutzung)
.def      temp2   = r17
.def      temp3   = r18
.def      MScount = r19                ; 10ms Zähler für den Interrupt (10ms x 100 = 1s)
.def      BCD out = r20                ; übergibt 2 BCD-Nibbles an die LCD-Zeichenausgabe
.def      Bit nr  = r21                ; 0..59 für Zuordnung Bit => dcf xxx-Variable
.def      high cnt= r22                ; zählt Länge des Signalpegels (high) in ms * 10
.def      Flags   = r23                ; Hier können diverse Stati gespeichert werden
.equ Error = 0                        ; Bit0 gesetzt => Protokollfehler
;
; Konstanten festlegen
.equ XTAL      = 4096000                ; Quarzfrequenz für Zeitschleifen-Berechnung
.equ TIMERwert = 256-160                ; für 10ms Timer bei 4.096MHz Takt und Prescaler 1/256
;
.cseg                                ; Codesegment
;
; Interrupt Vektoren
.org 0      rjmp start                ; Reset Handler, Startadresse 0
.org OVFOaddr rjmp t0int              ; Timer0 Interrupt alle 10ms
;
; Initialisierung
start:
    ldi temp1, LOW(RAMEND)              ; Stackpointer initialisieren
    out SPL, temp1                      ; LOW-Byte der obersten RAM-Adresse
    ldi temp1, HIGH(RAMEND)             ; HIGH-Byte der obersten RAM-Adresse
    out SPH, temp1
    ldi temp1, 0b11111111
    out DDRA, temp1                    ; PortA nur Ausgänge für LC-Display
    out PortB, temp1                  ; Pull-up-R an Empfänger-Eingang
    clr NULL
    out DDRB, NULL                    ; PortB nur Eingänge: Empfänger an PB2
;
; Variablen initialisieren
    sts MIN, NULL
    sts STU, NULL
    sts JHR, NULL                    ; Jahr (20)00
    ldi temp1, 1
    sts SEK, temp1                    ; die RTC läuft erst mal mit 00:00:01 an.

```

```

    sts TAG, temp1
    sts MON, temp1                ; 1. Januar
    sts WTAG, temp1              ; Montag
    clr TZ
    clr High cnt
    rcall init_next              ; weitere Variablen initialisieren
;
; Timer0 (8 Bit): Liefert die Zeitbasis für Signalerfassung & Quarzuhr
    ldi temp1, 1<<CS02
    out tccr0b, temp1            ; Prescaler 1/256
    ldi temp1, 1<<toie0         ; Überlauf-Interrupt für Timer0 aktivieren
    out timsk0, temp1           ; 256 - 4,096MHz * 10ms / 256 = 256 - 160 = 96
;
    rcall lcd_init               ; Display initialisieren
    rcall lcd_loadcustom        ; Selbst definierte Zeichen laden (ASCII 0-7)
    rcall lcd_clear             ; Display löschen
    rcall pon_mess              ; Firmwareversion in 2. Zeile schreiben
    sei                        ; Globale Interrupt Freigabe
;
main:; ##### Hauptprogramm - Endlos-Schleife #####
    tst MScount
    brne main0
    rcall ZEITAUSGABE           ; Die Zeit nur 1x / Sekunde ausgeben
main0:
    lds temp1, SEK
    cpi temp1, $00              ; Neue Minute ?
    brne main1                 ; Sprung wenn nicht
    rcall ZEITZONE              ; Zeitzone & Datum nur bei Minutenwechsel ausgeben
    rcall DateOut              ; Datum überschreibt hier die Pon_Mess (Firmware)
main1:
    cpi temp1, $01              ; In Sekunde 1: Fehler für nächste Minute neu vorbesetzen
    brne main2
    sbr Flags, 1<<Error
main2:
    sbrc MScount, 0             ; MScount wechselt alle 8ms
    rcall Show DCF Output       ; Signalpegel nur alle 16ms darstellen
    rjmp main; ##### Hauptprogramm - Endlos-Schleife #####
;
Show DCF Output:               ; Anzeige des Empfängersignals zur Antennenausrichtung
    ldi temp1, $8E              ; Cursor: Zeile 1, Spalte 15
    rcall lcd_command
    ldi temp1, 1                ; Low Signal => runder Punkt = LED-Nachbildung
    sbic PINB, 2
    ldi temp1, ' '              ; High Signal => mit Blank überschreiben
    rcall lcd_data
    ret
;*****
;* Timer0 Interruptroutine, alle 10ms *
;*****
t0int:
    push temp1
    push temp2
    push temp3
    in intreg, sreg             ; CPU-Flags sichern
    ldi temp1, TIMERwert        ; Timer Startwert für 10ms (s. Equate zu Beginn)
    out tcnt0, temp1
; Signal von PB2 lesen: Wenn 0 dann Dauer der vorherigen Anhebung auswerten
; Wenn 1 dann nur Signalausgangszähler "high cnt" inkrementieren
    sbis PINB, 2                ; DCF77 Empfängersignal: Skip next if PINB,2 = 1
    rjmp LEVEL0                 ; Signal ist 0
    inc high cnt                ; Signal ist 1
    rjmp LEVEL1
LEVEL0:
    rcall signal_analyse        ; Zeit-Telegramm dekodieren
LEVEL1:
    inc MScount                 ; 10ms-Zähler um eins erhöhen
    cpi MScount, 100            ; ist eine Sekunde vorbei? 100 x 10ms = 1s
    brne t0jmp0                ; wenn nicht dann Ende
    clr MScount                 ; ansonsten MScount auf null
    rcall RTC                   ; Quarzzeit um 1s erhöhen
t0jmp0:
    out sreg, intreg            ; CPU-Flags wiederherstellen
    pop temp3
    pop temp2
    pop temp1
    reti
;*****
;* Quarzuhr (Real Time Clock) als Gangreserve *
;*****

```

```

RTC:lds temp1, SEK
    rcall BCDINC                ; Sekunden erhöhen
    sts SEK, temp1              ; Sekunde erhöht
    cpi temp1, $60              ; Minute voll ?
    brlo RTx                    ; wenn nicht => springen
    sts SEK, NULL               ; sonst Sekunde löschen
; 1 Minute vergangen
    lds temp1, MIN              ; gleiches Prinzip wie oben
    rcall BCDINC
    sts MIN, temp1
    cpi temp1, $60
    brlo RTx
    sts MIN, NULL
; 1 Stunde vergangen
    lds temp1, STU              ; gleiches Prinzip wie oben
    rcall BCDINC
    sts STU, temp1
    cpi temp1, $24
    brlo RTx
    sts STU, NULL              ; das Datum wird hier nicht aktualisiert.
RTx:ret
;
BCDINC:
    subi temp1, -7              ; temp1 -> gepackte BCD-Zahl erhöhen -> temp1
    brhc BCx                    ; inkrementieren und 6 addieren
    subi temp1, 6               ; half-carry wird bei "subi rh, -k" invertiert
BCx:ret
;
;*****
;* DCF77 1-Level-Dekoder: Wertet nur die Pausen zwischen den Absenkungen aus *
;*****
signal analyse:
; Signal ist jetzt bei Aufruf = 0: Wie lange war es vorher auf "1"?
    cpi high cnt, 76            ; Werte hier für 10ms Iteration
    brlo Sx                     ; keine Aktion wenn < 760ms
    cpi high cnt, 84
    brsh SA1                    ; > 840ms => weiter prüfen
; High Signal zwischen 760 und 840ms => 200ms Absenkung => logisch 1
    rcall DCF_1                 ; "1" zu Bit_nr speichern
    rjmp Sx
SA1:cpi high cnt, 86
    brlo Sx                     ; keine Aktion wenn < 860ms
    cpi high cnt, 94
    brsh SA2                    ; > 940ms => weiter prüfen
; High Signal zwischen 860 und 940ms => 100ms Absenkung => logisch 0
    rcall DCF_0                 ; "0" zu Bit_nr speichern
    rjmp Sx
SA2:cpi high cnt, 176
    brlo Sx                     ; keine Aktion wenn < 1760ms
    cpi high cnt, 184
    brsh SA3                    ; > 1840ms => weiter prüfen
; High Signal zwischen 1760 und 1840ms => lange Absenkung 58. Sek => logisch 1
    rcall DCF_1                 ; "1" zu Bit nr 58 speichern
    rcall final_check           ; fehlerfreier Empfang?
    rjmp Sx
SA3:cpi high cnt, 186
    brlo Sx                     ; keine Aktion wenn < 1860ms
    cpi high cnt, 194
    brsh SAx                    ; High Signal zwischen 1860 und 1940ms => kurze Absenkung 58. Sek => logisch 0
; High Signal zwischen 1860 und 1940ms => kurze Absenkung 58. Sek => logisch 0
    rcall DCF_0                 ; "0" zu Bit nr 58 speichern
    rcall final_check           ; fehlerfreier Empfang?
SAx:clr high_cnt                ; Zähler in jedem Fall löschen: Signal ist hier = 0
    ret
;
final check:
; Parity-Prüfung und ggf. Synchronisierung der Uhr
    cpi Bit nr, 59
    brne init_next              ; Fehler Minutenende falsch erkannt
    sbrc P1, 0                  ; gerade Minuten-Parity?
    rjmp init_next              ; Parity-Fehler
    sbrc P2, 0                  ; gerade Stunden-Parity?
    rjmp init_next              ; Parity-Fehler
    sbrc P3, 0                  ; gerade Datums-Parity?
    rjmp init_next              ; Parity-Fehler
;
    cbr Flags, 1<<Error         ; Fehlerbit in Flags löschen
    sts SEK, NULL               ; Sekunden auf Null
    clr MScount                 ; Mit "Quarz-Sekunde" synchronisieren

```

```

    sts MIN, dcf min                ; ### Uhr synchronisieren ###
    sts STU, dcf stu
    sts TAG, dcf tag
    sts WTAG, dcf wtg
    sts MON, dcf mon
    sts JHR, dcf jhr
    mov TZ, dcf_TZ
;
init next:                          ; in jedem Fall neue Minute initialisieren
    clr Bit_nr                     ; mit Frame-Bit 0 wieder beginnen
    clr dcf_min                    ; letztes Zeit-Telegramm löschen
    clr dcf_stu
    clr dcf_tag
    clr dcf_wtg
    clr dcf_mon
    clr dcf_jhr
    clr dcf_TZ
    clr P1                         ; Parity-Zähler löschen
    clr P2
    clr P3
    ret
;
DCF 0:                              ; Aktionen für eine empfangene "0"
    lsr shifter                    ; schiebe 0 von links in Bit 7
    rcall DCF_Store                ; weiter schieben und speichern
    cpi Bit_nr, 18                 ; Validierung der Timezone durch Bit 18
    brne dcf0_x
    inc dcf_TZ                     ; MESZ: Bit 18=0 UND Bit 17=1 => 2x inkrementieren
dcf0_x:
    inc Bit_nr
    ret
;
DCF 1:                              ; Aktionen für eine empfangene "1"
    SEC                           ; CARRY setzen
    ror shifter                    ; schiebe 1 aus CARRY von links in Bit 7
    rcall DCF_Store                ; weiter schieben und speichern
    cpi Bit_nr, 17                 ; Sonderbehandlung Timezone
    brne add_Parity
    inc dcf_TZ                     ; MESZ: Bit 18=0 UND Bit 17=1 => 2x inkrementieren
add_Parity:
; Parityzähler P1, P2, P3: Datenbits & Paritybits jeweils aufaddieren
    cpi Bit_nr, 21
    brlo DCF_1Exit
    cpi Bit_nr, 29
    brsh Add_P2
    inc P1                         ; 21..28: Minuten Parity
    rjmp DCF_1Exit
Add_P2:
    cpi Bit_nr, 36
    brsh Add_P3
    inc P2                         ; 29..35: Stunden Parity
    rjmp DCF_1Exit
Add_P3:
    inc P3                         ; >= 36: Datum Parity
DCF_1Exit:
    inc Bit_nr
    ret
;
DCF_Store: ; je nach Bit_nr den Inhalt von Shifter rechtsbündig in Variable speichern
;
; Minuten: 21..27 (7 Bit) => 1x rechts schieben, dann steht Einer Bit0 in Variable Bit0
    cpi Bit_nr, 27                 ; Sekunde 27 = letztes Minutenbit erreicht?
    brne dcf_St1                  ; wenn nicht, letztes Stundenbit abfragen
    lsr shifter                    ; sonst: 1x rechts schieben (0 nach Bit 7)
    mov dcf_min, shifter           ; Minute abspeichern: 00000000
    rjmp dcf_Stx                   ; fertig
; Bit 28 = Minuten-Parity wird separat bearbeitet s.o.
; Stunden: 29..34 (6 Bit) => 2x rechts schieben, dann steht Einer Bit0 in Variable Bit0
dcf_St1:
    cpi Bit_nr, 34                 ; Stunden-Bits abfragen
    brne dcf_St2                  ; Sekunde 34 = letztes Stundenbit erreicht?
    lsr shifter                    ; wenn nicht, letztes Bit Kalendertag abfragen
    lsr shifter                    ; sonst: 2x rechts schieben
    mov dcf_stu, shifter           ; Stunde abspeichern: 00hhhhhh
    rjmp dcf_Stx                   ; fertig
; Bit 35 = Stunden-Parity wird separat bearbeitet s.o.
; Kalendertag: 36..41 (6 Bit) => 2x rechts schieben, dann steht Einer Bit0 in Variable Bit0
dcf_St2:
    ; Kalendertag abfragen

```

```
;
    cpi Bit nr, 41                ; Sekunde 41 = letztes Bit Kalendertag erreicht?
    brne dcf St3                 ; wenn nicht, Wochentag & Kalendermonat abfragen
    lsr shifter                  ; sonst: 2x rechts schieben
    lsr shifter
    mov dcf tag, shifter         ; Kalendertag abspeichern: 00ddddd
    rjmp dcf_Stx                ; fertig
;
; Wochentag: 42..44 (3 Bit) und Kalendermonat: 45..49 (5 Bit) gemeinsam bearbeiten
dcf St3:
    cpi Bit nr, 49                ; Wochentag UND Kalendermonat abfragen
    brne dcf St4                 ; Sekunde 49 = letztes Bit Kalendermonat erreicht?
    mov temp1, shifter           ; wenn nicht weiter bei Kalenderjahr
    andi temp1, 0b00000111       ; shifter nicht überschreiben
    mov dcf wtg, temp1           ; 3 LSB = Wochentag isolieren
    lsr shifter                  ; ..und abspeichern
    lsr shifter                  ; 5 MSB = Kalendermonat => shifter 3x rechts schieben
    lsr shifter                  ; dann steht Kalendermonat rechtsbündig in shifter
    mov dcf mon, shifter         ; Monat abspeichern: 000MMMMM
    rjmp dcf_Stx                ; fertig
;
; Kalenderjahr: 50..57 (8 Bit) => nicht schieben, Einer Bit0 ist schon in Variable Bit0
dcf St4:
    cpi Bit nr, 57                ; Jahr abfragen
    brne dcf Stx                 ; Sekunde 57 = letztes Bit Kalenderjahr erreicht?
    mov dcf_jhr, shifter         ; wenn nicht, kein Treffer
    ret                          ; sonst: Jahr abspeichern: YYYYYYYY
dcf Stx:
    ret                          ; Bit 58 = Datum-Parity wird separat bearbeitet s.o.
;
; ### Ende Dekoder ###
singledot:
    ldi temp1, '.'
    rcall lcd_data
    ret
;
doubledot:
    ldi temp1, ':'
    rcall lcd_data
    ret
;
ZEITAUSGABE:
    rcall lcd cursorhome         ; Ausgabe der Uhrzeit (1x pro Sekunde)
    lds BCD out, STU             ; setze Cursor an den Anfang erste Zeile
    rcall timeout2              ; Stunden ausgeben
    rcall doubledot             ; Doppelpunkt auf LCD schreiben
    lds BCD out, MIN             ; Minuten ausgeben
    rcall timeout2              ; Doppelpunkt auf LCD schreiben
    lds BCD out, SEK             ; Sekunden ausgeben
    rcall timeout2
    ret
;
timeout2:
    mov temp1, BCD out           ; Ausgabe von Uhrzeit oder Datum: 1 Byte = 2 BCD-Ziffern
    swap temp1                  ; in Arbeitsregister kopieren
    andi temp1, 0b00001111       ; Nibbles vertauschen (oberes nach unten zuerst)
    subi temp1, -48              ; nur unteres Nibble betrachten
    rcall lcd_data              ; ASCII-Offset (= ADDI $30, ADDI gibt es aber nicht)
    mov temp1, BCD out           ; dieses ASCII-Zeichen auf LCD ausgeben
    andi temp1, 0b00001111       ; ursprüngliches Byte zurück in Arbeitsregister kopieren
    subi temp1, -48              ; unteres Nibble steht schon richtig
    rcall lcd_data              ; ASCII-Korrektur
    ret                          ; Zeichen auf Display schreiben
;
ZEITZONE:
    ldi temp1, $89               ; Cursor auf Zeile 1, Spalte 10 (hinter Uhrzeit)
    rcall lcd command           ; gemeinsamer Vorspann ME
    ldi temp1, 'M'
    rcall lcd_data
    ldi temp1, 'E'
    rcall lcd_data
    sbrs TZ, 1                  ; Variable aus Zeit-Telegramm auswerten: TZ = 2 bei MESZ
    rjmp MEZ
    ldi temp1, 'S'
    rcall lcd_data
MEZ: ldi temp1, 'Z'
    rcall lcd_data
    ldi temp1, ' '
    rcall lcd_data
    ret
;
;
```

```

tage:      .db "MoDiMiDoFrSaSo"
;
dateout:   ; Ausgabe des Datums
           ldi temp1,$C0      ; Display-Cursor auf Anfang der zweiten Zeile
           rcall lcd_command
           ldi ZL, low(tage*2) ; Zeiger auf Beginn des Strings "tage"
           ldi ZH, high(tage*2)
;
; Offset für betreffenden Tag = (WTAG -1) * 2 berechnen und in temp1 ablegen
           lds temp1, WTAG    ; 1 = Mo...7 = So
           dec temp1         ; -1
           lsl temp1         ; Mit 2 multiplizieren, da 2 Buchstaben pro Tag
           add ZL, temp1      ; diesen Offset zu Z aufaddieren
           adc ZH, NULL       ; Z zeigt jetzt auf den 1. Buchstaben des Tages
           lpm temp1, Z+      ; diesen nach temp1 holen, dann Z inkrementieren
           rcall lcd_data    ; 1. Buchstabe auf LCD ausgeben
           lpm temp1, Z      ; 2. Buchstabe aus der oben inkrementierten Adresse
           rcall lcd_data    ; 2. Buchstabe auf LCD ausgeben
;
           ldi temp1, ' '    ; Blank im Display zwischen Wochentag und Datum
           rcall lcd_data
           lds BCD out, TAG   ; Kalendertag ausgeben
           rcall timeout2
           rcall singledot   ; Punkt auf LCD schreiben
           lds BCD out, MON   ; Kalendermonat ausgeben
           rcall timeout2
           rcall singledot   ; Punkt auf LCD schreiben
           ldi BCD out, $20   ; Präfix 20 ergänzen (nicht im Datenstrom enthalten)
           rcall timeout2
           lds BCD out, JHR   ; Kalenderjahr ausgeben
           rcall timeout2
;
           ldi temp1, $CE    ; ### Anzeige des Fehlerstatus' ###
           rcall lcd_command ; Cursor nach Zeile 2, Spalte 15
           ldi temp1, 0      ; bisher kein Fehler => selbst definiertes Häkchen
           sbrc Flags, Error
           ldi temp1, '?'    ; Fehler: Error-Bit war gesetzt
           rcall lcd_data
           ret
;
pon mess:  ; PowerOn Message senden
           ldi temp1,$C0      ; Cursor auf Anfang 2. Zeile
           rcall lcd_command
           ldi ZL, LOW(text1*2) ; Adresse des Strings in den
           ldi ZH, HIGH(text1*2) ; Z-Pointer laden
           rcall lcd_flash_string ; Unterprogramm gibt String aus, der
           ret                ; durch den Z-Pointer adressiert wird
;
text1:     .db "DCF77->FA1.04", $FF ; String durch FFh abgeschlossen, gerade Bytezahl!
;
;=====
; Die folgenden Routinen dienen der Ansteuerung des LCDs mit HD44780 Controller
; im 4 Bit-Modus. Benutzte Variablen: temp1, temp2, temp3, Register Z (r30, 31)
; Hardware-Anbindung gemäß Platine TN20140914:
; PA0 - PA3 => D4 - D7, PA4 = Enable = Display Pin 6, PA5 = RS = Display Pin 4
lcd_data:  ; Ein Datenbyte an das LCD senden
           push temp2
           mov temp2, temp1   ; "Sicherungskopie" für 2. Durchgang
           swap temp1         ; für die Übertragung des 2.Nibbles vertauschen
           andi temp1, 0b00001111 ; oberes Nibble auf Null setzen
           ori temp1, 0b00100000 ; es sind Daten => RS=1 (PA5)
           out PORTA, temp1   ; ausgeben
           rcall lcd_enable   ; Enable-Routine aufrufen
           andi temp2, 0b00001111 ; obere Hälfte des 2. Nibble auf Null setzen
           ori temp2, 0b00100000 ; es sind Daten => RS=1 (PA5)
           out PORTA, temp2   ; ausgeben
           rcall lcd_enable   ; Enable-Routine aufrufen
           rcall delay50us    ; 50us warten
           pop temp2
           ret
;
lcd_command: ; Einen Befehl an das LCD senden
            ; wie lcd_data, nur ohne RS zu setzen
           push temp2
           mov temp2, temp1
           swap temp1
           andi temp1, 0b00001111
           out PORTA, temp1
           rcall lcd_enable

```

```
    andi temp2, 0b00001111
    out PORTA, temp2
    rcall lcd enable
    rcall delay50us
    pop temp2
    ret

;
lcd enable:                                ; Enable-Puls erzeugen
    sbi PORTA, 4                          ; Enable high
    nop                                  ; etwas warten
    nop
    cbi PORTA, 4                          ; Enable wieder low
    ret

;
delay50us:                                ; 50us Pause nach jeder Übertragung
    ldi temp1, (XTAL * 50 / 3) / 1000000
delay50us :
    dec temp1
    brne delay50us_
    ret

;
delay5ms:                                 ; 5ms Pause für manche Befehle
    ldi temp1, (XTAL * 5 / 607) / 1000
WLOOP0:
    ldi temp2, $C9
WLOOP1:
    dec temp2
    brne WLOOP1
    dec temp1
    brne WLOOP0
    ret

;
lcd init:                                ; Initialisierung: Muss als Erstes erfolgen
    ldi temp3, 60                          ; etwas warten
powerupwait:
    rcall delay5ms
    dec temp3
    brne powerupwait
    ldi temp1, 0b00000011                  ; muss 3x hintereinander gesendet
    out PORTA, temp1                      ; werden zur Initialisierung
    rcall lcd enable                      ; 1
    rcall delay5ms                        ; 2
    rcall lcd enable                      ; 2
    rcall delay5ms                        ; 3
    rcall lcd enable                      ; 3
    ldi temp1, 0b00000010                  ; 4 Bit-Modus einstellen
    out PORTA, temp1
    rcall lcd enable
    rcall delay5ms
    ldi temp1, 0b00101000                  ; 4 Bit, 2 Zeilen, 5x8 Punkte je Zeichen
    rcall lcd command
    ldi temp1, 0b00001100                  ; Display on, Cursor off, kein Blinken
    rcall lcd command
    ldi temp1, 0b00000100                  ; inkrement / kein Scrollen
    rcall lcd_command
    ret

;
lcd clear:                                ; Display löschen
    ldi temp1, 1
    rcall lcd_command
    rcall delay5ms
    ret

;
lcd cursorhome:                           ; Cursor auf Zeile 1, Spalte 1 setzen
    ldi temp1, 2
    rcall lcd_command
    rcall delay5ms
    ret

;
lcd flash string:                          ; Stringausgabe ab Adresse ZL/ZH, Ende-Zeichen=FFh
    lpm temp1, Z+                          ; Buchstabe laden und post-increment Adresse
    cpi temp1, $FF                          ; Ende-Kennung
    breq lcd_flash_string_1
    rcall lcd_data
    rjmp lcd_flash_string
lcd_flash_string_1:
    ret
```

```

;
; Selbst definierte Zeichen in das CG-RAM des Displays laden: Muss 1x
; nach lcd init aufgerufen werden. Die selbst definierten Zeichen
; können dann über die ASCII-Codes 0-7 via lcd_data ausgegeben werden.
LCD_LOADCUSTOM:
    LDI z1, LOW (lcd user char * 2) ; Adresse der Zeichentabelle
    LDI zh, HIGH(lcd user char * 2) ; in den Z-Pointer laden
    CLR temp3 ; aktuelles Zeichen = 0
;
LCD_LOADCUSTOM2:
    CLR temp2 ; Linienzähler = 0
;
LCD_LOADCUSTOM1:
    LDI temp1, 0b01000000 ; Kommando: 0b01aaa111
    add temp1, temp3 ; + akt. Zeichen(aaa)
    add temp1, temp2 ; + akt. Lin ie (111)
    rcall LCD_COMMAND ; Kommando schreiben
;
    lpm temp1, Z+ ; Zeichenline laden
    rcall LCD_DATA ; ... und ausgeben
;
    ldi temp1, 0b01001000 ; Kommando: 0b01aa1111
    add temp1, temp3 ; + akt. Zeichen (aaa)
    add temp1, temp2 ; + akt. Linie (111)
    rcall LCD_COMMAND
;
    lpm temp1, Z+ ; Zeichenline laden
    rcall LCD_DATA ; ... und ausgeben
;
    inc temp2 ; Linienzähler + 1
    cpi temp2, 8 ; 8 Linien fertig?
    brne LCD_LOADCUSTOM1 ; nein, dann nächste Linie
;
    subi temp3, -$10 ; zwei Zeichen weiter (addi $10)
    lpm temp1, Z ; nächste Linie laden
    cpi temp1, $FF ; Tabellenende erreicht?
    brne LCD_LOADCUSTOM2 ; nein, dann die nächsten zwei Zeichen
    RET
;
lcd user char: ; bis zu 8 selbst definierte Zeichen
; 0 1
    .db 0b00000, 0b00000 ; ,
    .db 0b00001, 0b00000 ; @,
    .db 0b00010, 0b01110 ; @ , @@@
    .db 0b10100, 0b11111 ; @ @ , @@@@
    .db 0b01000, 0b11111 ; @ , @@@@
    .db 0b00000, 0b01110 ; , @@@
    .db 0b00000, 0b00000 ; ,
    .db 0b00000, 0b00000 ; Zeichen: Häkchen & runder LED-Punkt
    .db $FF, $FF ; Tabellenende - je Zeile immer gerade Anzahl von Bytes!

```