8 Temperatur-Messung und Analog-Digital-Wandler

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

8. Temperatur-Messung und Analog-Digital-Wandler

ADC - wie kommt mehr als nur ein Bit über einen Pin?

Ziele

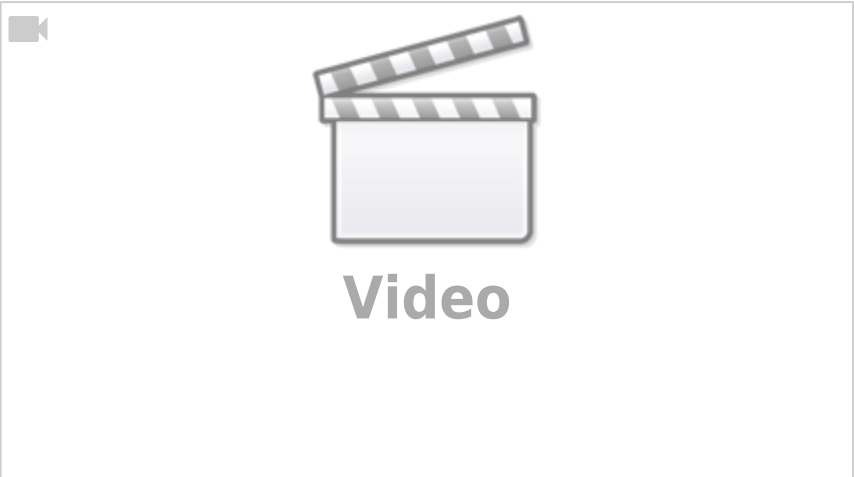
Nach dieser Lektion sollten Sie:

1. wissen, wie ein ADC genutzt wird

weiterführende Links

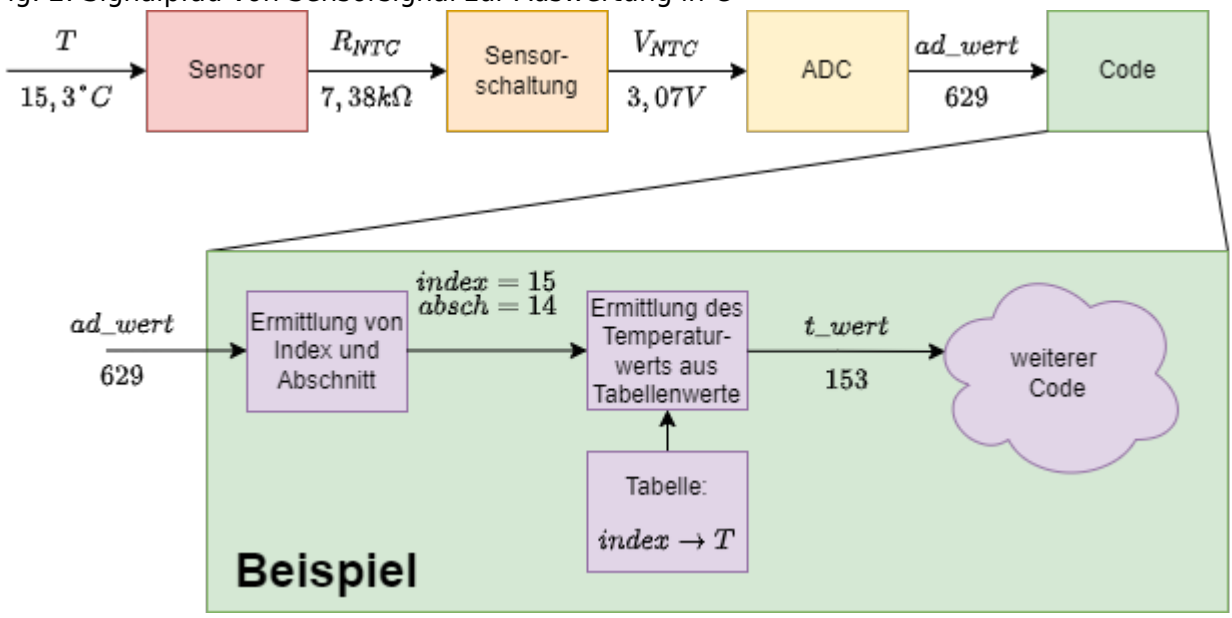
- Eine schnelle und seichte Einführung in AD-Wandler

Video



Messsignal-Digitalisierung und Auswertung

Fig. 2: Signalpfad von Sensorsignal zur Auswertung in C



Um Sensorwerte in C sinnvoll nutzen zu können sind einige Schritte zwischen dem passiven Sensor über den ADC bis in den Code umzusetzen (siehe figure 2).

1. Sensor:
   Umwandlung der physikalischen Größe in einen Messsignal
2. Sensorschaltung:
   Aufbereitung des Messsignal (z.B. Filter, Verstärkung oder Umwandlung in Spannungssignal)

3. ADC:
   Digitalisierung in ein Zahlenwert im Code
4. Aufbereitung im Code
   um ein Abbild des Sensorwerts zu erhalten

Häufig ergeben sich hierbei folgende Herausforderungen:

- Der digitalisierte Wert ist nicht linear zur physikalischen Größe
- Die Relation zwischen digitalisiertem Wert und physikalischen Größe muss platzsparend und effizient umgesetzt werden.
- Die Angaben im Datenblatt sind für feste Werte der physikalischen Größe vorgegeben (siehe table 1).
  Im Code werden aber feste digitale Werte benötigt (siehe table 2).

Temperatur	Widerstandswert
$5°C$	$11,9k\Omega$
$10°C$	$9,34k\Omega$
$15°C$	$7,37k\Omega$
...	...

Tab. 1: Beispiel für eine Vorgabe im Datenblatt

Variable t_wert	Temperatur
256	$52,1°C$
272	$49,9°C$
288	$47,9°C$
...	...

Tab. 2: Beispiel für die notwendige Relation im Code

Diese Einzelschritte und Herausforderungen sollen im Folgenden beschrieben werden.

1.a Umwandlung der physikalischen Größe in einen Messsignal (Sensorwert zu Widerstandswert)

- Bei vielen passiven Sensoren wird der Sensorwert in eine Widerstandsänderung umgewandelt. Die Relation von Sensorwert zu Widerstandswert ist im Datenblatt des Sensors angegeben.
- Das Datenblatt der gewünschten Komponente (hier: Thermistors 2381 640 6472) ist im Internet zu finden.
- Suchen Sie dort nach dem gewünschten Komponente (hier Thermistor 2381 640 6472). Beachten Sie, das manche Datenblätter eine andere Sortierung / Benamung nutzen, als sie von den Distributoren genutzt wird. Im konkreten Fall hilft eine Suche nach den letzten drei Ziffern “472”.

1.b Konvertierung des Datenblatts in Excel

Um die Daten aus dem Datenblatt aufzubereiten empfiehlt sich eine Verarbeitung in einem Analyse-Tool, z.B. Matlab. Für kleinere Tabelle kann auch ein Tabellenkalkulationsprogramm wie Excel eine Hilfe sein. Im Folgenden sollen die Schritte anhand von Excel erklärt werden.

- In vielen Fällen kann das Datenblatt über `Daten » Daten abrufen » Aus Datei » Aus PDF` eingelesen werden. über diese Variante ist aber hier in vertretbarer Zeit kein Import möglich, da das Datenblatt viele verschiedene Tabellen enthält.
- Eine andere Variante ist ein Umweg über Word
  - Erstellen Sie ein leeres Dokument in Word
  - Öffnen Sie über `Datei » Öffnen » Durchsuchen` mit dem Filter `*.pdf` das gewünschte Datenblatt. Der Import sollte nur wenige Sekunden dauern
  - Es soll nun die relevante Tabelle ausgewählt werden (hier: im pdf Seite 80, in Word Seite 84). Nutzen Sie hierzu das Auswahltool “$\boxed{{\leftarrow}\mkern-10mu{\rightarrow}\mkern-17mu{\uparrow}\mkern-9mu{\downarrow}\;\;}$” links oben bei der Tabelle.

    - Kopieren Sie die ausgewählte Tabelle
    - In Excel muss nun zunächst das Dezimaltrennzeichen geändert werden, da es sich um eine englischsprachiges Datenblatt handelt. Dies geschieht im Menu `Datei » Optionen » Erweitert`. Hier sollte `Trennzeichen von Betriebssystem übernehmen` deaktiviert und als Dezimaltrennzeichen ein Punkt (`.`) gewählt werden.
    - Nun kann die Tabelle eingefügt werden. Es empfiehlt sich nach dem Einfügen das Dezimaltrennzeichen wieder zurückzustellen.
    - Im Anschluss sollten nur die relevanten Zeilen und Spalten gewählt werden (hier nur Zeilen und Spalten für "… 472"). Sinnvoll ist auch mögliche doppelte oder verbundene Zeilen / Spalten zu reduzieren.
- Nun sollten die relevanten Spalten (hier: Temperatur und Widerstandswert) in Excel vorhanden sein

2. + 3. Relation von Widerstandswert zu ADC-Wert

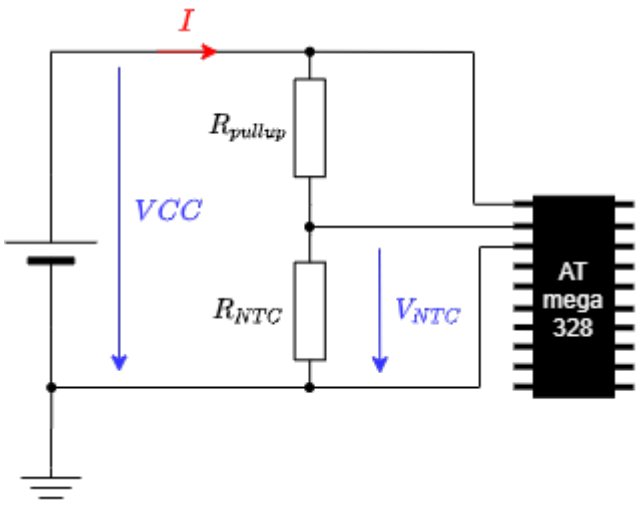


Fig. 3: Schaltung des Sensorwiderstands

Der Analog-Digitalwandler konvertiert die anliegende Spannung in einen Wert, welcher im Code weiter genutzt werden kann. Im ATmega328 ist ein 10-Bit ADC verbaut. Dieser wandelt Spannungen von 0V bis VCC (hier 0..5V) in einen internen ADC-Wert von 0 bis 1023 ($2^{10}-1$) um.

Es muss aber nun auch eine Beziehung zwischen Temperatur und ADC-Wert gefunden werden, um die Temperatur intern aus dem digitalisierten Wert ermitteln zu können. Hierzu ist es notwendig zunächst die Schaltung zu betrachten (siehe figure 3).

Die Spannung am ADC lässt sich leicht über den Widerstandswert berechnen:

$$\begin{align} {{V_{NTC}} \over {V_{CC}}} = {{R_{NTC}}\over{R_{NTC} + R_{pullup}}} \tag{8.1} \end{align}$$

$$\begin{align} ADCwert = round \left({{V_{NTC}}\over{V_{CC}}} \cdot 1024 \right) \tag{8.2} \end{align}$$

aus (8.1) und (8.2) ergibt sich

$$\begin{align} ADCwert = round \left({{R_{NTC}}\over{R_{NTC} + R_{pullup}}} \cdot 1024 \right) \tag{8.3} \end{align}$$

Übung

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 7_mexleclock.simu
    2. 7_mexleclock.hex
    3. lcd_lib_de.h

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE und öffnen Sie dort mittels  die Datei `7_mexleclock.simu`
    2. Laden Sie `7_mexleclock.hex` als firmware auf den 328 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes ist im Display eine Uhr mit dem Format HH:MM:SS Menu zu sehen
    5. Die Tasten 2 und 3 ermöglichen das Einstellen der Stunde und Minute. Werden die Minuten hochgezählt, so werden die Sekunden auf 0 gesetzt.

III. Eingabe in Atmel Studio

```
/* ------------------------------------------------------------------------

 Experiment 8: Temperaturmessung mit MiniMEXLE
 ==============================================

 Dateiname: 8_Temperature.c

 Autoren    : Peter Blinzinger
              Prof. G. Gruhler     (Hochschule Heilbronn, Fakultät T1)
              D. Chilachava (Georgische Technische Universität)
 Datum      : 01.05.2020

 Version    : 1.1

 Hardware:  MEXLE2020 Ver.
```

```
/*=====================================================================
```

Ändern Sie auch hier wieder die Beschreibung am Anfang des C-Files, je nachdem was Sie entwickeln

```
1.0 oder höher                Deklarationen
           AVR-USB-PROGI      ==================================
Ver. 2.0

 Software:
Entwicklungsumgebung:
AtmelStudio 7.0
           C-Compiler:
AVR/GNU C Compiler 5.4.0

 Funktion : Thermometer mit
Anzeige der aktuellen
Temperatur und der
Maximaltemperatur im
Betriebszeitraum in °C mit
1/10 Grad.
           Keine
Tastenbedienung

 Displayanzeige:    Start
(fuer 2s):        Betrieb:
                     +------
----------+    +------------
----+
                  |-
Experiment 8 -|    |Temp.
18.5°C|
                  |
Temperature   |    |Maximum
21.6°C|
                  +--------
--------+    +-------------
--+

 Tastenfunktion:    keine

 Jumperstellung:    keine

 Fuses im uC:       CKDIV8:
Aus    (keine generelle
Vorteilung des Takts)

 Header-Files:  lcd_lib_de.h
(Library zur Ansteuerung
LCD-Display Ver.1.3)

 Module 1) Taktgenerator
           2) AD-Wandlung
(Takt: 100 ms)
           3) Umrechnung
für Temperatur  (Takt: 100
```

1. Hier wird wieder geprüft ob die Frequenz des Quarz bereits eingestellt wurde und - falls nicht - dessen Frequenz eingestellt.
2. Die Header-Dateien entsprechen denen der letzten Programme.
3. Auch die Makros entsprechen denen der letzten Programme.
4. Die Konstanten entsprechen denen der letzten Programme.
   Zusätzlich wird der Ausdruck ASC_NULL durch den Hexadezimalwert einer '0' in ASCII, also 0x30 und ASC_COLON durch den ASCI-Wert eines Doppelpunkts, also 0x3A, ersetzt
5. Bei den Variablen entsprechen einige denen der letzten Programme.
6. Für die Uhr werden Stunden, Minuten, Sekunden und Zehntelsekunden mit Anfangswerten deklariert.
7. Bei den Funktionsprototypen sind einige bekannte Unterprogramme vorhanden. Details werden weiter unten erklärt.

```
ms)
            4)
Anzeigetreiber (Takt: 1 s)

 1) Das Modul
"Taktgenerator" erzeugt den
Takt von 100 ms für die AD-
Wandlung
      und Umrechnung und
einen zusätzlichen Takt von
1 s für die Anzeige.

        Verwendung von
Hardware-Timer 0 und T0
Overflow-Interrupt.
        Frequenzen:
Quarzfrequenz
12,288 MHz.
        Timer-Vorteiler
/  8    =>   1,536 MHz
        Hardware-Timer
/256    =>   6 kHz / 166 µs
        Software-Vorteiler
/ 60    => 100 Hz  /  10 ms
        Hundertstel-Zaehler
/ 10    =>  10 Hz  / 100 ms
        Zehntel-Zaehler
/ 10    =>   1 Hz  /   1 s

 2) Das Modul "AD-Wandlung"
wird durch den Takt 100 ms
aufgerufen.
    Der AD-Wandler wird mit
einem internen Takt von 96
kHz betrieben.
      Im Modul wird eine
einzelne AD-Wandlung des
Kanals ADC0 mit 10 Bit
    Auflösung gestartet.
Dort ist der NTC des Boards
mit Vorwiderstand
    als temperaturabhängiger
Spannungsteiler bzw.
Potentiometer angeschlossen.
    Als Referenzspannung
wird die 5V-Versorgung
verwendet.
    Das Ergebnis wird in der
globalen Variable ad_wert
gespeichert.
```

Hauptprogramm ==========

1. Das Hauptprogramm ähnelt sehr stark dem Up/Down Counter. Entsprechend werden die Zeilen 143-157 hier nicht weiter erklärt.

2. In der Endlosschleife sind auf der ersten Ebene wieder nur If-Abfragen zu den Flags `takt10ms` und `takt100ms` zu finden.

    1. Alle $10ms$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrStellen()` aufgerufen

    2. Alle $100ms$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrAnzeigen()` aufgerufen

    3. Alle $1s$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `uhrZaehlen()` aufgerufen

Interrupt Routine ==========

1. Auch die Interrupt Routine ist dem Programm Up/Down Counter entlehnt und wird hier nicht weiter erklärt.

```
 3) Das Modul "Umrechnung"
wird nach der AD-Wandlung
alle 100 ms gestartet.
    Der Ergebniswert des
Moduls "AD_Wandlung" wird
mit Hilfe einer Tabelle in
    einen entsprechenden
Temperaturwert umgerechnet.
In der Tabelle sind
    Temperaturwerte über
äquidistante (Abstand = 16)
AD-Werte aufgetragen.
      Die Werte dazwischen
werden mit linearer
Interpolation ermittelt.
    Weiterhin wird im Modul
jede aktuelle Temperatur mit
der gespeicherten
      maximalen Temperatur
verglichen und der
Maximalwert optional
angepasst.

 4) Das Modul
"Anzeigetreiber" ist an den
1 s-Takt gekoppelt. Damit
wird ein
    zu schnelles Umschalten
der Anzeigewerte vermieden.
Das Modul gibt die
    Werte der aktuellen und
der maximalen Temperatur in
1/10 °C aus.
      Zwischen AD-Wandlung /
Umrechnung und Anzeige kann
später noch eine
      Mittelwertsbildung mit
10 Werten eingefügt werden.

 Die Kopplung der Module
wird über global definierte
Variable realisiert:

      1-Bit-Variable:
Takt 100 ms:
Taktgenerator => AD-Wandlung
=> Umrechnung
                        Takt
1  s:    Taktgenerator =>
Anzeigetreiber
```

```
Taster initialisieren ==============
```

1. Auch das Einstellen des Data Direction Registers und der Pullups wurde bereits in vorherigen Programmen erklärt.

```
Stellfunktion ==============
```

1. In dieser Funktion werden zunächst die Stellungen aller Taster eingelesen (vgl. `counterCounting(void)` bei Up/down Counter).

2. Neu hier ist, dass die Bedienung der Schalter die Variablen für Stunden, Minuten um eins hochsetzen, bzw. bei Überlauf wider zurück auf 0 setzen. Zusätzlich wird bei eine Änderung des Minuten-Werts der Sekunden-Wert auf 0 gesetzt.

```
Anzeigefunktion Uhr
========================
```

1. Hierüber wird die Uhrzeit in der ersten Zeile im

```
      16-Bit-Variable:
ad-wert            AD-
Wandlung => Umrechnung
                        t-
wert            Umrechnung
=> Anzeige
tmax-wert        Umrechnung
=> Anzeige

// -------------------------
---------------------------
----------------------*/

// Deklarationen
============================
============================
======

// Festlegung der
Quarzfrequenz
#ifndef F_CPU
// optional definieren
#define F_CPU 12288000UL
// MiniMEXLE mit 12,288 MHz
Quarz
#endif

// Include von Header-
Dateien
#include <avr/io.h>
// Header-Dateien zum
ATmega88
#include <stdbool.h>
// Header-Datei fuer Bit-
Berechnung
#include <avr/interrupt.h>
// Header-Datei fuer
Interrupts
#include <util/delay.h>
// Header-Datei fuer
Wartezeit
#include "lcd_lib_de.h"
// Header-Datei fuer LCD-
Anzeige

// Konstanten
#define VORTEILER_WERT
60
#define HUNDERTSTEL_WERT
10
#define ZEHNTEL_WERT
```

Format hh:mm:ss ausgegeben.
2. Ähnlich zum Counter werden die zweistelligen Werte mit Division durch 10 und dessen Rest in zwei einzelne Ziffern gewandelt

`Initialisierung Display-Anzeige`
==========

1. Die Funktion `initDisplay()` wird zu Beginn des Programms aufgerufen und führt zunächst die Initialisierung des Displays aus.
2. Danach wird der erste Text auf den Bildschirm geschrieben und damit der Programmname dargestellt.
3. Nach zwei Sekunden wird der Auswahlbildschirm angezeigt.

`Zaehlfunktion Uhr`
==========

1. Die Zähl-Funktion `uhrZaehlen()` ist ganz ähnlich aufgebaut zur Interrupt-Service-Routine
2. Zunächst wird ein Schaltwechsel am Ausgang mit dem Lautsprecher ausgegeben, um einen Knackton zu erzeugen
3. Dann werden die Sekunden hochgezählt
4. ist das Maximum erreicht, so wird der Sekunden-Wert zurückgesetzt und der

```
10

const int   TEMP[45]     =
{521,499,479,460,441,423,405
,388,371,355,
339,324,309,294,279,265,250,
236,222,208,
194,180,166,152,137,123,108,
93,78,63,
48,32,15,-01,-19,-37,-55,-76
,-96,-120,
-137,-168,-200,-236,-276};

// Die Tabellenwerte sind in
1/10 °C angegeben
// Der erste Tabellenwert
entspricht einem AD-Wert
// von 256. Die Abstände der
AD-Werte sind 16

// Variable
unsigned char   vorteiler
= VORTEILER_WERT;        //
Zählvariable Vorteiler
unsigned char   hundertstel
= HUNDERTSTEL_WERT;
unsigned char   zehntel =
ZEHNTEL_WERT;

unsigned int    ad_wert = 0;
// Variable für den AD-
Wandlungswert
int             t_wert=0;
// Variable für die
Temperatur (in 1/10 °C)
int
tmax_wert=-300;    //
Variable für maximale
Temperatur (1/10 °C)

bool            takt10ms;
// Bit-Botschaft alle 10 ms
bool            takt100ms;
// Bit-Botschaft alle 100 ms
bool            takt1s;
// Bit-Botschaft alle 1s

//Funktionsprototypen
void initTimer0 (void);
void adWandlerInit (void);
void adWandlung (void);
```

Minuten-Wert um eins hochgezählt.
5. ebenso wird beim Maximum des Minuten-Serts dieser zurückgesetzt und der Stundenwert hochgezählt.
6. beim Maximum des Stunden-Werts wird dieser wieder auf Null gesetzt

```
void umrechnung (void);
void anzeigeTreiber (void);
void initDisplay (void);

// Initialisierung und
Hauptprogramm
//
============================
============================
====================
int main ()
{
    initDisplay();
// Initialisierung LCD-
Anzeige
    initTimer0();
// Initialisierung von
Timer0
    adWandlerInit();
// Initialisierung des AD-
Wandlers

    sei();
// generell Interrupts
einschalten

    // Hauptprogrammschleife
_____________________________
_____________________

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhängig von
Taktbotschaften
    {
        if(takt100ms ==
true)    // Durchführung der
Funktion einmal pro 100ms
        {
            takt100ms =
false;    // Taktbotschaft
zurücksetzen
            adWandlung();
// Ausführung des Modules
der A/D-Wandlung
            umrechnung();
// Ausführung des Modules
der Umrechnung
        }
```

```
        if(takt1s == true)
// Durchführung der Anzeige
einmal pro 1s
        {
            takt1s = false;
// Taktbotschaft
zurücksetzen
anzeigeTreiber();    //
Ausführung des Modules der
Anzeige
        }
    }
}

// Funktionen
============================
============================
======

// Timer-Initialisierung:
// --------------------
//
// Initialisierung des
Timer0 zur Erzeugung eines
getakteten Interrupts.
// Er dient dazu, die
benötigten Taktbotschaften
zu erzeugen.
void initTimer0()
{
    TCCR0A |= (0<<WGM00) |
(0<<WGM01);    // Timer 0
auf "Normal Mode" schalten
    TCCR0B |= (0<<WGM02) |
(1<<CS01);    // mit
Prescaler /8 betreiben
    TIMSK0 |= (1<<TOIE0);
// Overflow-Interrupt
aktivieren
}

// A/D-Wandler-
Initialisierung:
// ------------------------
--
// Initialisierung des A/D-
Wandlers:
// Vorteiler = 128 =>
interner Takt = 96 kHz
// Abfrage des ADC0 (NTC-
Spannungsteiler)
```

```
// Referenzspannung =
analoge Versorgung Avcc

void adWandlerInit ()
{
    ADMUX  |= (1<<REFS0);
// Vref =AVCC; ADC0

    ADCSRA |=
(1<<ADPS0)|(1<<ADPS1)|(1<<AD
PS2)|(1<<ADEN); // Teiler
128; ADC ON
}


// Interrupt-Routine
//
============================
============================
====================
ISR (TIMER0_OVF_vect)
// In der Interrupt-Routine
sind die Softwareteiler
realisiert, die die Takt-
// botschaften (10ms, 100ms,
1s) für die Module erzeugen.
Die Interrupts
// werden von Timer 0
ausgelöst (Interrupt Nr. 1)
//
// Veränderte Variable:
vorteiler
//
hunderstel
//
zehntel
//
// Ausgangsvariable:
takt10ms
//
takt100ms
//
takt1s
{
    --vorteiler;
// Vorteiler dekrementieren
    if (vorteiler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler =
```

```
VORTEILER_WERT;        //
Vorteiler auf Startwert
        takt10ms = true;
// Botschaft 10ms senden
        --hundertstel;
// Hundertstelzähler
dekrementieren

        if (hundertstel==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            hundertstel =
HUNDERTSTEL_WERT; // Teiler
auf Startwert
            takt100ms =
true;            //
Botschaft 100ms senden
            --zehntel;

            if (zehntel==0)
// Zehntelzähler
dekrementieren
            {
                zehntel =
ZEHNTEL_WERT;    // Teiler
auf Startwert
                takt1s =
true;            //
Botschaft 1s senden
            }
        }
    }
}

// Modul ADWandlung:
// ----------------
// Durchführung einer
Einzelwandlung der am NTC-
Spannungsteiler anstehenden
// Spannung in einen
digitalen 10-bit-Wert
(einmal pro 100 ms).
void adWandlung()
{
    ADCSRA |= (1<<ADSC);
// Wandlung starten
    while (ADCSRA &
(1<<ADSC));            //
Ende der Wandlung abwarten
```

```
    ad_wert = ADCL +
(ADCH<<8);            // 10-
Bit-Wert berechnen
// ADCL muss vor ADCH
stehen!!
// siehe Datenblatt des
ATmega 328
}

// Modul Umrechnung:
// --------------------
// (wird alle 100 ms
aufgerufen)
void umrechnung ()
{
    unsigned char index;
// Tabellenindex für
Temperaturtabelle
    unsigned char abschnitt;
// Abstand zum
nächstkleineren Wert
// der AD-Werte der
Temperaturtabelle
    index =
(ad_wert-256)/16;
// Indexberechnung (Zeiger
in Tabelle)
    abschnitt =
(ad_wert-256)%16;        //
Rest für Tabellen-
Interpolation
    t_wert = abschnitt *
(TEMP[index+1] -
TEMP[index])/16 +
TEMP[index];
// Temperaturwert berechnen

    if(t_wert>=tmax_wert)
// aktueller Wert mit
Maximalwert
    {
        tmax_wert = t_wert;
// vergleichen und ggf.
ersetzen
    }
}

// Modul Anzeigetreiber:
// --------------------
//
// Beschreiben der Anzeige
```

```
mit dem erstellten
Temperaturwert
// und mit dem maximalen
Wert (wird alle 1 s
aufgerufen).
//
// Umrechnung der
Zahlenwerte (1/10 °C) in
Anzeigewerte wie folgt:
// Hunderter: einfache
Integer-Teilung (/100).
// Zehner: Modulo-Ermittlung
(%100), d.h. Rest bei der
Teilung durch 100
//         dann nochmals
Integer-Teilung (/10) dieses
Restes.
// Einer:  Modulo-Ermittlung
(%10), d.h. Rest bei der
Teilung durch 10.
//
// Umrechnung in ASCII-Werte
für die Anzeige durch
Addition von 0x30.
void tempAnzeige(int
ausgabewert, char zeile,
char spalte)
{
    int i;
    i = ausgabewert;
    lcd_gotoxy(zeile,
spalte);        //
Startposition für
Temperatur-Wert
    if (i>=0)
// zuerst Vorzeichen: ' '
oder '-'
    {
        lcd_putc(' ');
    }
    else
    {
        lcd_putc('-');
        i = -i;
// Vorzeichenumkehr bei
negativer Zahl
    }
    lcd_putc(i/100 + 0x30);
// Hunderter ausgeben (°C
Zehner)
    i = i%100;
```

```
    lcd_putc(i/10 + 0x30);
// Zehner ausgeben (°C
Einer)
    lcd_putc(0x2E);
// Punkt ausgeben
    lcd_putc(i%10 + 0x30);
// Einer ausgeben (°C
Zehntel)
}

// Anzeigefunktion
// ---------------
//
// Der aktuelle Temperatur
und die maximale Temperatur
werden ausgegeben
 void anzeigeTreiber()
{
    tempAnzeige(t_wert, 0,
9);         // aktuelle
Temperatur ab Position 0,9
    tempAnzeige(tmax_wert,
1, 9);    // maximale
Temperatur ab Position 1,9
}

// Initialisierung Display-
Anzeige
void initDisplay()
// Start der Funktion
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("- Experiment
8 -"); // Ausgabe Festtext:
16 Zeichen

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("
Temperature   ");    //
Ausgabe Festtext: 16 Zeichen

    _delay_ms(2000);
// Wartezeit nach
Initialisierung
```

```
    lcd_gotoxy(0,0);
// Cursor auf 1. Zeile, 1.
Zeichen
    lcd_putstr("Temp.
ßC"); // Ausgabe Festtext:
16 Zeichen
// "ßC" wird als °C
dargestellt

    lcd_gotoxy(1,0);
// Cursor auf 2. Zeile, 1.
Zeichen
    lcd_putstr("Maximum
ßC");    // Ausgabe
Festtext: 16 Zeichen
// "ßC" wird als °C
dargestellt
}
// Ende der Funktion
```

IV. Ausführung in Simulide

1. Geben Sie die oben dargestellten Codezeilen ein und kompilieren Sie den Code.
2. Öffnen Sie Ihre hex-Datei in SimulIDE und testen Sie, ob diese die gleiche Ausgabe erzeugt

Bitte arbeiten Sie folgende Aufgaben durch:

Aufgabe

1. Bauen Sie einen “ewigen Kalender” ein
    1. Es sollen nicht nur Stunde, Minute, Sekunde dargestellt werden und veränderbar sein, sondern auch Tag, Monat und Jahr im Format dd:mm:yy
    2. Achten Sie auf die Schaltjahrermittlung
2. Erweitern Sie den Kalender auf vierstellige Jahresangabe.
3. Überlegen Sie sich, wie man Ihre Programme testen kann.
4. BONUS: Wie kann der Wochentag bestimmt werden?