8 Temperatur-Messung und Analog-Digital-Wandler

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

8 Temperatur-Messung und Analog-Digital-Wandler 2
Ziele 2
weiterführende Links 2
Video 2
Messsignal-Digitalisierung und Auswertung 2
1.a Umwandlung der physikalischen Größe in ein Messsignal (Sensorwert zu Widerstandswert) 3
Konvertierung des Datenblatts in Excel 3
2. + 3. Relation von Widerstandswert zu ADC-Wert 4
4. Aufbereitung der des ADC-Werts im Code 6
direkte Verwendung der Datenblattwerte 6
direkte Verwendung der Datenblattwerte mit Interpolation 7
Umrechnung der Datenblattwerte 9
Übung 9

8 Temperatur-Messung und Analog-Digital-Wandler

Ziele

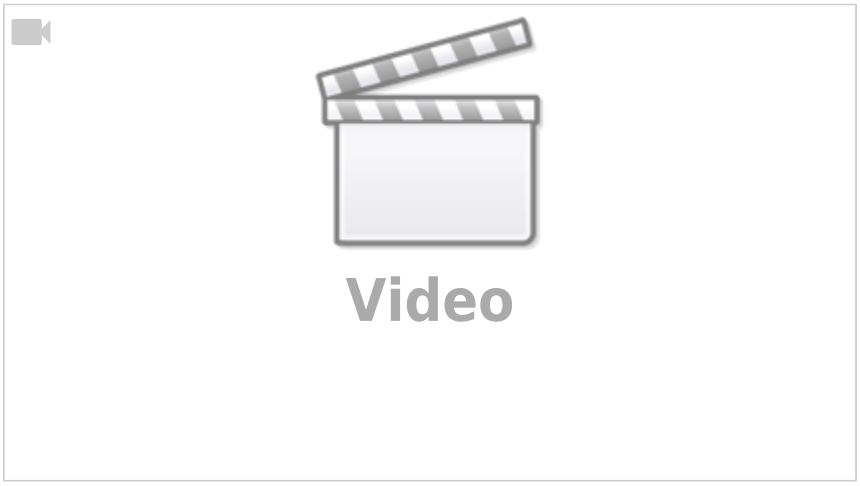
Nach dieser Lektion sollten Sie:

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: $T \rightarrow R_{\rm NTC}$
   Umwandlung der physikalischen Größe in ein Messsignal

2. Sensorschaltung: $R_{\rm NTC} \rightarrow V_{\rm NTC}$
   Aufbereitung des Messsignal (z.B. Filter, Verstärkung oder Umwandlung in Spannungssignal)

3. ADC: $V_{\rm NTC} \rightarrow adcValue$
   Digitalisierung in ein Zahlenwert im Code

4. Aufbereitung im Code $adcValue \rightarrow tValue$
   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 T	Widerstandswert $R_{\rm NTC}$
$5° \rm C$	$11,9~\rm k\Omega$
$10°\rm C$	$9,34~\rm k\Omega$
$15°\rm C$	$7,37~\rm k\Omega$
...	...

Tab. 1: Beispiel für eine Vorgabe im Datenblatt

ADC-Wert $adcValue$	Temperatur T
256	$52,1°\rm C$
272	$49,9°\rm C$
288	$47,9°\rm C$
...	...

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 ein 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.
- Häufig sind im Datenblatt Stützpunkte angegeben, bei welchen der physikalischen Größe ein Messsignal gegenübersteht. Die ist auch hier der fall.
- Suchen Sie dort nach der 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”.

> **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_{\rm NTC}} \over {V_{\rm CC}}} = {{R_{\rm NTC}}\over{R_{\rm NTC} + R_{\rm pullup}}} \tag{8.1} \end{align}

Daraus ergibt sich der digitalisierte Wert zu

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

aus (8.1) und (8.2) ergibt sich

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

Diese Relation muss nun im Analyse-Tool abgebildet werden und wurde in dieser Excel-Datei umgesetzt. Einige der verwendeten Excel-Tricks sollen hier beschrieben werden:

- Benamung von Zellen und Bereichen: Für viele Zellen wurden Namen vergeben, z.B. Zelle J2 in Tabelle 1 erhielt die Benamung V_CC oder der Bereich E7:E45 die Benamung R_NTC.
- Rechnen mit Bereichen: In den Zellen F7 und G7 in Tabelle 1 wurden mit Angabe der Bereichsnamen Formeln eingefügt. Damit entfällt das Auffüllen der weiteren Zeilen
- Einfügen einer Trendlinie: Für das Diagramm wurde ADCval als x-Werte und R_NTC als y-Werte verwendet. Für die Trendlinie wurde ein Polynom 4. Grades genutzt. Mit Anzeige der Trendlinien-Formel wird die Relation von Widerstandswert zu ADC-Wert sichtbar. Die Zahl-Eigenschaft der Trendlinienbeschriftung sollte dabei auf `Wissenschaftlich` mit 2 Dezimalstellen gestellt werden, um die Auflösung zu erhöhen.

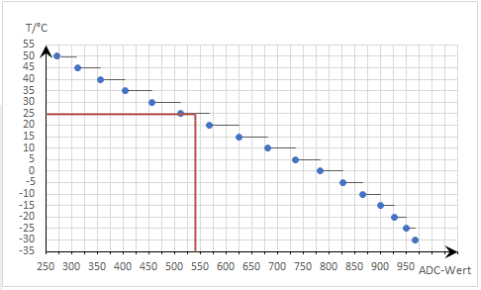
4. Aufbereitung der des ADC-Werts im Code

Mit den bisherigen Betrachtungen ist zwar klar, wie die Relation von Widerstandswert zu ADC-Wert aussieht, aber diese ist noch nicht im Code umgesetzt. Zunächst sollte der notwendige Wertebereich geklärt werden: In welchem Bereich werden Messwerte erwartet? In diesem Konkreten Fall geht es um die Innenraumtemperatur. Damit wird die Temperatur im Bereich $[-30°\rm C, 50°\rm C]$ sein.

direkte Verwendung der Datenblattwerte

Die einfachste Variante wäre die vorhandenen Daten aus dem Datenblatt direkt zu nehmen und zu prüfen, ob der eingelesene tValue zwischen zwei bekannten Werten liegt.

Fig. 4: Bildliche Erklärung für die "direkte Verwendung"




```
#define     MAX_INDEX    17    // Anzahl der
Stuetzstellen

const int TEMP[MAX_INDEX]    ={ -30, -25,
-20, -15, -10,  -5,   0,   5,  10,  15,  20,
25,  30,  35,  40,  45,  50};
const int ADC_VAL[MAX_INDEX] ={ 968, 950,
927, 899, 866, 827, 783, 734, 681, 625, 568,
512, 456, 404, 356, 311, 271};
                            // Datenwerte aus
dem Datenblatt
                            // TEMP[]
```

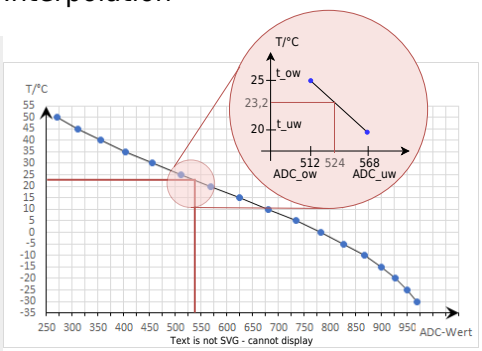
```
enthaelt die Temperaturen
                            // In ADC_VAL[]
sind die den Temperaturen
                            // entsprechenden
ADC-Werte abgespeichert
...
    for(int index=0; index<MAX_INDEX;
index++)
    {                        // suche den
Index, bei dem der gemessene
                            // Wert zum
ersten mal darueber liegt
        if (tValue >= ADC_VAL[index]) break;
// wenn gefunden, breche ab
    };                        // (index zeigt
auf untere Temperatur)
    int tValue = TEMP[index];// Der
Temperaturwert mit dem
...                         // gefundenen
index wird zugewiesen
```

Nachteil dabei ist, dass die Ausführung unterschiedlich lange benötigt und die Auflösung nur 5°C beträgt (vgl. figure 4).

direkte Verwendung der Datenblattwerte mit Interpolation

In einem weiteren Schritt könnte eine Interpolation zwischen den Werten umgesetzt werden

Fig. 5: Bildliche Erklärung für die "direkte Verwendung mit Interpolation"




```
#define     MAX_INDEX    17    // Anzahl der
Stuetzstellen

const int TEMP[MAX_INDEX]    ={ -30, -25,
-20, -15, -10,  -5,   0,   5,  10,  15,  20,
25,  30,  35,  40,  45,  50};
const int ADC_VAL[MAX_INDEX] ={ 968, 950,
927, 899, 866, 827, 783, 734, 681, 625, 568,
512, 456, 404, 356, 311, 271};
                            // Datenwerte aus
dem Datenblatt
                            // TEMP[]
enthaelt die Temperaturen
                            // In ADC_VAL[]
```

```
sind die den Temperaturen
                              // entsprechenden
ADC-Werte abgespeichert
...
    for(int index=0; index<MAX_INDEX;
index++)
    {                          // suche den
Index, bei dem der gemessene
                              // Wert zum
ersten mal darueber liegt
        if (tValue >= ADC_VAL[index]) break;
// wenn gefunden, breche ab
    };                          // (index zeigt
auf untere Temperatur)
    int t_uW = TEMP[index];    // Der
Temperaturwert mit dem gefundenen index
                              // wird
gespeichert (unterer Wert = uW)
    int t_oW = TEMP[index+1];// Der
darauffolgende Temperaturwert wird
                              // gespeichert
(oberer Wert = oW)

    int ADC_uW = ADC_VAL[index];// Der ADC-
Wert mit dem gefundenen index
                              // wird
zugewiesen (unterer Wert = uW)
    int ADC_oW = ADC_VAL[index+1];// Der
darauffolgende ADC-Wert
                              // wird
zugewiesen (oberer Wert = oW)

    float tValue = TEMP[index] +
                         5 * (tValue - ADC_uW
) / (ADC_oW - ADC_uW ) ;
                              // lineare
Interpolation
                              // zum unteren
Temperaturwert wird ein
                              // Bruchteil
entsprechend der Differenz von
                              // tValue zum
unteren ADC Wert ADC_uW hinzuaddiert
...
```

Die Auflösung ist nun sehr genau (vgl. figure 5). Nachteile sind aber, dass die Ausführung immernoch unterschiedlich lange benötigt und die ein float-Wert relativ viel Speicher und CPU-Zeit bei weiteren Berechnungen benötigt.

Der float-Wert kann umgangen werden, wenn im gesamten Programm die Temperaturwerte in

Zehntel-Grad geschrieben werden. Also $37,5°\rm C \rightarrow$ 375. Dann kann ein tValue als signed integer geschrieben werden.

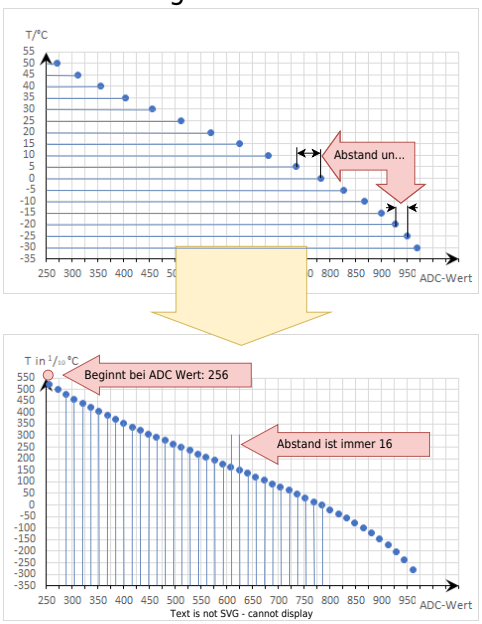
Umrechnung der Datenblattwerte

Eine geschicktere Option ist es, den Temperatur-Array `TEMP[]` bereits so vorauszuberechnen, dass diese über den ADC-Wert `tValue` adressiert werden kann. Dazu würde aber für alle Werte ein Array mit 1024 Werten benötigt (1kB Speicher). Eine Reduktion des Arrays ist also sinnvoll.

- Betrachtet man die obige Tabelle `ADC_VAL[]`, so ist jeder Wert größer als 256 und kleiner als 976. Diese Werte können also ignoriert werden.
- Statt die restlichen Werte in Einzelschritten abzuspeichern, könnte auch nur jeder 16. Wert gespeichert werden

Der Index kann damit als `index = (tValue - 256)/16;` ermittelt werden. Der Vorteil in den verwendeten Zahlen (Vielfache von 2, hier 16) ist, dass der Microcontroller mit diesen leichter rechnen kann. Dies ist in dem untenstehenden Code umgesetzt. Die Umrechnung des Temperatur-Arrays `TEMP[]` ist in obigen Excel-File beschrieben.

Fig. 6: Bildliche Erklärung für die "Umrechnung der Datenblattwerte"



Übung

I. Vorarbeiten

1. Laden Sie folgenden Dateien herunter:
   1. 8_temperature_1.0.0.sim1
   2. 8_temperature.hex
   3. lcd_lib_de.h

> **Beachten Sie, folgendes**
>
> - Es wird nun ein ATmega328 genutzt, d.h. das Programm ist nicht mehr kompatibel mit dem MiniMEXLE!
> - Das Display ist nunan einem anderen Port um den Analog-Digial-Wandler am Port C zu nutzen.
>   Deshalb muss der Treiber `lcd_lib_de.h` wiefolgt angepasst werden:
>   - Zeile 26: `#define F_CPU 12288000UL`

- Zeile 39: `#define PIN_EN PD4`
- Zeile 40: `#define PIN_RS PD7`
- Zeile 44: `#define DDR_DATA DDRD`
- Zeile 46: `#define PORT_DATA PORTD`

II. Analyse des fertigen Programms

1. Initialisieren des Programms
    1. Öffnen Sie SimulIDE (Version >= 0.5.15) und öffnen Sie dort mittels  die Datei `8_temperature_0.5.15.simu`
    2. Laden Sie `8_temperature.hex` als firmware auf den 328 Chip
    3. Zunächst wird eine Startanzeige mit dem Namen des Programms dargestellt.
    4. Als nächstes sind im Display zwei Temperaturanzeigen zu sehen. Die obere Zeile zeigt den aktuellen Wert, die untere Zeile den Maximalwert seit Start.
    5. Der Drehregler neben dem NTC ermöglicht es die Temperatur des NTC zu verändern.

III. Eingabe in Microchip Studio

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

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

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

 Experiment 8:
Temperaturmessung
 =============
============================
===

 Dateiname: 8_Temperature.c
 Autoren    : Peter
Blinzinger
              Prof. G.
Gruhler  (Hochschule
Heilbronn, Fakultaet T1)
              D. Chilachava
(Georgische Technische
Universitaet)
 Datum      : 01.05.2020
 Version    : 1.1
 Hardware:  MEXLE2020 Ver.
1.0 oder hoeher
```

```
            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
fuer Temperatur  (Takt: 100
ms)
```

```
Deklarationen
==================================
```

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. Die Konstanten entsprechen denen der letzten Programme.

4. Es wird eine zusätzliches Array `TEMP[]` angelegt, in denen die Temperaturen in 1/10 °C abgespeichert sind. Beispielsweise entspricht der erste Eintrag521 einer Temperatur von $52,1°\rm C$. Der erste Eintrag wird bei einem ADC-Wert von 256 benötigt. Der zweite Eintrag entspricht $49,9° \rm C$ und wird bei einem ADC-Wert von 256+16 = 272 benötigt. Die weiteren entsprechend. Die Herleitung erfolgt wie unter Messsignal-Digitalisierung und Auswertung vorgegeben.

5. Bei den Variablen entsprechen einige denen der letzten Programme.

6. Die Variable `adcValue` wird mit dem digitalisierten ADC-Wert befüllt.
7. Die Variable `tValue` beinhaltet die aktuelle Temperatur in 1/10°C und mit Vorzeichen.
8. Die Variable `tValueMax` beinhaltet die höchsten seit Start gemessene Temperatur in 1/10°C und mit Vorzeichen (entspricht einem Schleppzeiger).

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

 1) Das Modul
"Taktgenerator" erzeugt den
Takt von 100 ms fuer die AD-
Wandlung
      und Umrechnung und
einen zusaetzlichen Takt von
1 s fuer 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
    Aufloesung gestartet.
Dort ist der NTC des Boards
mit Vorwiderstand
    als
temperaturabhaengiger
Spannungsteiler bzw.
Potentiometer angeschlossen.
    Als Referenzspannung
wird die 5V-Versorgung
verwendet.
    Das Ergebnis wird in der
globalen Variable tValue
gespeichert.
```

9. Bei den Funktionsprototypen sind einige bekannte Unterprogramme vorhanden. Details werden weiter unten erklärt.

Hauptprogramm ========================

1. Das Hauptprogramm ähnelt sehr stark dem Up/Down Counter, wobei die Initialisierung des Timers in eine Unterfunktion `initTimer0()` ausgegliedert wurde. Die Unterfunktion `initAdc` initialisiert den ADC.

2. In der Endlosschleife sind auf der ersten Ebene wieder nur If-Abfragen zu den Flags `cycle100msActive` und `cycle1sActive` zu finden.

    1. Alle $100~\rm ms$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `doAdc()` sowie `calculateTemp()` aufgerufen

    2. Alle $1~\rm s$ (bzw. wenn das entsprechende Flag gesetzt wird) wird das Flag zurückgesetzt und das Unterprogramm `refreshDisplay()` aufgerufen

Timer Initialisierung ============

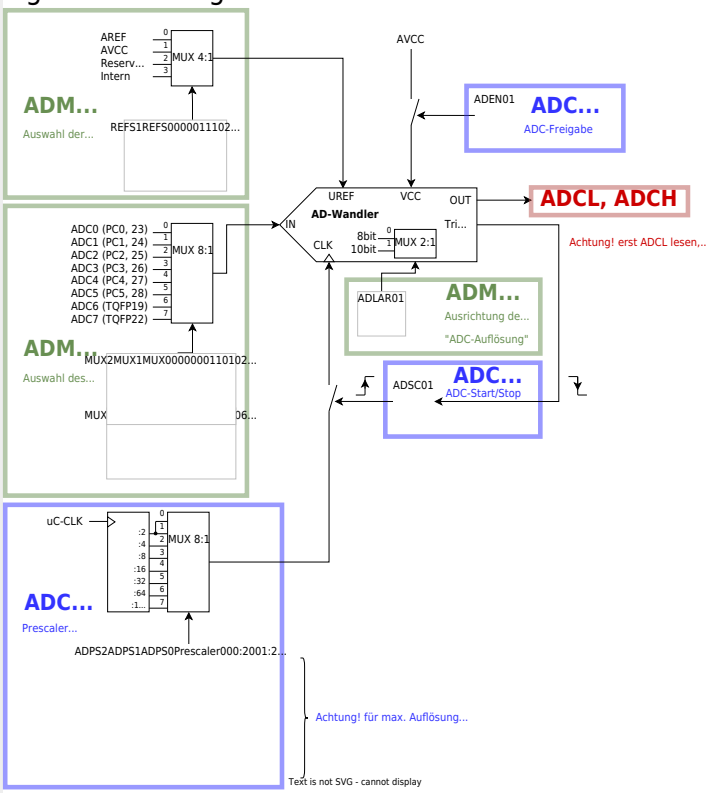
1. Die Timer Initialisierung ist dem Programm Up/Down Counter entlehnt und wird hier nicht weiter erklärt.

```
 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 ueber
aequidistante (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
spaeter noch eine
      Mittelwertsbildung mit
10 Werten eingefuegt werden.
 Die Kopplung der Module
wird ueber global definierte
Variable realisiert:

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

      16-Bit-Variable:  ad-
```

ADC Initialisierung ==========

1. Über den Multiplexer ADMUX wird die Referenzspannung für die AD-Wandlung vorgegeben. Hier wird auf die interne Referenz, gegeben durch Avcc, geschalten
2. ADCSRA ist das “ADC Control and Status Register A”.
   Die Bits ADPS0,ADPS1,ADPS2 geben den Teiler für die Wandlerfrequenz vor (hier 1/128 von F_CPU)
   ADEN aktiviert den Wandler. Die eigentliche Wandlung muss aber immer wieder gestartet werden.
3. WICHTIG: der erste digitalisierte Wert nach Initialisierung und auch nach Kanalwechsel muss ignoriert werden. Bei Änderungen der ADC-Konfiguration entspricht diese bei AVR-Microcontrollern noch keinem validen Wert!

Fig. 1: Schaltung des AD-Wandlers im 328



Timer-Service Routine ==========

1. Die Timer-Service Routine ist dem Programm Up/Down Counter entlehnt und wird hier nicht weiter erklärt.

```
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
// Atmega328 mit 12,288 MHz
Quarz
#endif

// Include von Header-
Dateien
#include <avr/io.h>
// Header-Dateien zum
ATmega328
#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 PRESCALER_VAL
60          // Faktor
Vorteiler = 60
#define CYCLE10MS_MAX
10          // Faktor
Hundertstel = 10
#define CYCLE100MS_MAX
10          // Faktor
Zehntel = 10
```

ADWandlung
==========

1. Das Bit ADSC startet die AD-Wandlung
2. Während die AD-Wandlung läuft bleibt das Bit ADSC=1. Ist die Wandlung beendet schaltet das Bit auf 0.
3. Das Register ADCL enthält die untersten (8) Bits. Das Register ADCH enthält die oberen (2) Bits, welche hier um 8 Bitt nach links verschoben werden (also mit 256 multipliziert wird).
   Wichtig:
   1. immer erst ADCL auslesen, dann ADCH.
   2. immer auch ADCH auslesen, selbst wenn es nicht benötigt wird.

Umrechnung ==========

1. Die Variable `index` gibt die Position in dem Array `TEMP[]` an. In `TEMP[]` sind nur Stützpunkte für jeden 16. Wert angegeben.
2. Die Variable `steps` gibt den Zwischenschritt zwischen zwei Stützpunkte in Sechzehntel an
3. `tValue` ermittelt die Lineare Interpolation zwischen zwei Schritten
4. In `tValue` wird der Schleppzähler gespeichert

Anzeigefunktion
==========

1. Da zwei Temperaturen ausgegeben werden müssen, ruft die Anzeigefunktion eine weitere Unterfunktion auf, welche mit unterschiedlichen Werten gespeist wird

```
#define ASC_NULL            0x30        // Das Zeichen '0' in ASCII
#define ASC_FULL_STOP       0x2E        // Das Zeichen ':' in ASCII

const int   TEMP[45]    = {521,499,479,459,440,422,404,388,371,354,
338,323,308,293,279,264,250,236,221,207,
193,179,165,151,137,122,108,93,78,63,
48,32,15,-1,-19,-38,-56,-77,-97,-121,
-145,-173,-202,-237,-278};

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

// Variable
unsigned char softwarePrescaler = PRESCALER_VAL;    // Zaehlvariable Vorteiler
unsigned char cycle10msCount = CYCLE10MS_MAX;    // Zaehlvariable Hundertstel
unsigned char cycle100msCount   = CYCLE100MS_MAX;   // Zaehlvariable Zehntel

unsigned int  adcValue    = 0;    // Variable fuer den AD-Wandlungswert
int           tValue      = 0;    // Variable fuer die Temperatur (in 1/10 °C)
int           tValueMax =-300;  // Variable fuer maximale Temperatur (1/10 °C)

bool cycle10msActive;    // Bit-
```

Anzeigetreiber fuer Temperaturanzeige

1. Auf der ersten Position wird gegebenenfalls ein Vorzeichen angezeigt. Falls die Temperatur negativ ist, muss für den nächsten Schritt der Betrag gebildet werden

2. Für die Temperatur wird zunächst die Zehner und die Einer ermittelt und ausgegeben. Im nächsten Schritt folgt das Dezimaltrennzeichen und die Zehntel Grad.

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 die Display-Vorlage für die Temperatur angezeigt.

```
Botschaft alle 10 ms
bool
cycle100msActive;   // Bit-
Botschaft alle 100 ms
bool          cycle1sActive;
// Bit-Botschaft alle 1s

//Funktionsprototypen
void initTimer0 (void);
void initAdc (void);
void initDisplay (void);
void doAdc (void);
void calculateTemp (void);
void refreshDisplayTemp(int
tempValue, char line, char
pos);
void refreshDisplay (void);

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

    sei();
// generell Interrupts
einschalten
    // Hauptprogrammschleife
_____________________________
_________________________

    while(1)
// unendliche Warteschleife
mit Aufruf der
// Funktionen abhaengig von
Taktbotschaften
    {
        if(cycle100msActive)
// Durchfuehrung der
Funktion einmal pro 100ms
        {
```

```
            cycle100msActive
= 0;// Taktbotschaft
zuruecksetzen
            doAdc();
// Ausfuehrung des Modules
der A/D-Wandlung
            calculateTemp();
// Ausfuehrung des Modules
der Umrechnung
        }

        if(cycle1sActive)
// Durchfuehrung der Anzeige
einmal pro 1s
        {
            cycle1sActive =
0;  // Taktbotschaft
zuruecksetzen
refreshDisplay();   //
Ausfuehrung des Modules der
Anzeige
        }
    }
}


// Timer Initialisierung
============================
============================
======
//
// Initialisierung des
Timer0 zur Erzeugung eines
getakteten Interrupts.
// Er dient dazu, die
benoetigten 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
}
```

```
// ADC-Initialisierung
============================
============================
======
//
// Initialisierung des A/D-
Wandlers:
// Vorteiler = 128 =>
interner Takt = 96 kHz
// Abfrage des ADC0 (NTC-
Spannungsteiler)
// Referenzspannung =
analoge Versorgung Avcc
void initAdc()
{
    ADMUX   |= (1<<REFS0);
// Vref =AVCC; ADC0

    ADCSRA  |= (1<<ADPS0)
            |  (1<<ADPS1)
            |  (1<<ADPS2)
            |  (1<<ADEN);
// Teiler 128; ADC ON
    (void) ADCH;
// erster Wert ist
"wegzuwerfen"
}


// Timer-Service Routine
============================
============================
======
//
// In der Interrupt-Routine
sind die Softwareteiler
realisiert, die die Takt-
// botschaften (10ms, 100ms,
1s) fuer die Module
erzeugen. Die Interrupts
// werden von Timer 0
ausgeloest (Interrupt Nr. 1)
//
// Veraenderte Variable:
vorteiler
//
hunderstel
//
zehntel
//
// Ausgangsvariable:
```

```
takt10ms
//
takt100ms
//
takt1s

ISR (TIMER0_OVF_vect)
{
    --softwarePrescaler;
// Vorteiler dekrementieren
    if
(softwarePrescaler==0)
// wenn 0 erreicht: 10ms
abgelaufen
    {
        softwarePrescaler =
PRESCALER_VAL;      //
Vorteiler auf Startwert
        cycle10msActive =
true;                 //
Botschaft 10ms senden
        --cycle10msCount;
// Hundertstelzaehler
dekrementieren

        if
(cycle10msCount==0)
// wenn 0 erreicht: 100ms
abgelaufen
        {
            cycle10msCount =
CYCLE10MS_MAX;     // Teiler
auf Startwert
            cycle100msActive
= true;            //
Botschaft 100ms senden
            --
cycle100msCount;

            if
(cycle100msCount==0)
// Zehntelzaehler
dekrementieren
            {
cycle100msCount =
CYCLE100MS_MAX;// Teiler auf
Startwert
cycle1sActive = true;
// Botschaft 1s senden
            }
        }
```

```
    }
}

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

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

// Umrechnung
============================
============================
======
//
// (wird alle 100 ms
aufgerufen)
void calculateTemp ()
{
    unsigned char index;
// Tabellenindex fuer
Temperaturtabelle
    unsigned char steps;
// Abstand zum
naechstkleineren Wert
// der AD-Werte der
Temperaturtabelle
    index   =
(adcValue-256)/16;        //
Indexberechnung (Zeiger in
```

```
Tabelle)
    steps   =
(adcValue-256)%16;        //
Rest fuer Tabellen-
Interpolation
    tValue = steps *
(TEMP[index+1] -
TEMP[index])/16 +
TEMP[index];
// Temperaturwert berechnen

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

// Anzeigefunktion
============================
============================
======
//
// Der aktuelle Temperatur
und die maximale Temperatur
werden ausgegeben
 void refreshDisplay()
{
refreshDisplayTemp(tValue,
0, 9);      // aktuelle
Temperatur ab Position 0,9
refreshDisplayTemp(tValueMax
,   1, 9);      // maximale
Temperatur ab Position 1,9
}

// Anzeigetreiber fuer
Temperaturanzeige
============================
============================
======
//
// 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
fuer die Anzeige durch
Addition von 0x30.
void refreshDisplayTemp(int
tempValue, char line, char
pos)
{
    lcd_gotoxy(line, pos);
// Startposition fuer
Temperatur-Wert
    if (tempValue>=0)
// zuerst Vorzeichen: ' '
oder '-'
    {
        lcd_putc(' ');
    }
    else
    {
        lcd_putc('-');
        tempValue = -
tempValue;             //
Vorzeichenumkehr bei
negativer Zahl
    }
    lcd_putc
(tempValue/100 + ASC_NULL);
// Hunderter ausgeben (°C
Zehner)
    tempValue =
tempValue%100;
    lcd_putc   (tempValue/10
+ ASC_NULL);  // Zehner
ausgeben (°C Einer)
    lcd_putc
(ASC_FULL_STOP);
```

```
// Punkt ausgeben
    lcd_putc   (tempValue%10
+ ASC_NULL);  // Einer
ausgeben (°C Zehntel)
}

// Initialisierung Display-
Anzeige
============================
============================
======
//
void initDisplay()
// Start der Funktion
{
    lcd_init();
// Initialisierungsroutine
aus der lcd_lib
    lcd_displayMessage("-
Experiment 8 -",0,0); //
Ausgabe in erster Zeile
    lcd_displayMessage("
Temperature   ",1,0); //
Ausgabe in zweiter Zeile

    _delay_ms(2000);
// Wartezeit nach
Initialisierung

lcd_displayMessage("Temp.
ßC",0,0); // Ausgabe in
erster Zeile
lcd_displayMessage("Maximum
ßC",1,0); // Ausgabe in
zweiter Zeile
// "ßC" wird als °C
dargestellt
}
```

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. Die Anzeige am Display scheint nicht immer genau zum eingestellten Temperaturwert zu passen. Um dies zu verstehen, sollten Sie betrachten, welche Parameter in Simulide beim NTC eingestellt werden können. Weiterhin gibt es dazu Tipps unter dem Elektrotechnik 1-Kapitel: Temperaturabhängigkeit von Widerständen
2. Entwickeln Sie ein Programm, welches mit einem Poti über den ADC des Atmega328 den Cursor bzw. ein Zeichen auf dem Display bewegt.