9 UART und Terminal

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

9 UART und Terminal

Bei der Programmierung wünscht man sich häufig die Möglichkeit Daten des Mikrocontrollers irgendwo darzustellen.

- Wird eine reale Hardware verwendet, kann mit Hilfe des Freeware Programms PuTTY leicht ein Terminal für die Kommunikation mit dem PC geöffnet werden. Zusätzlich wird dann noch ein USB-to-serial Adapter benötigt.
- In der Simulation SimulIDE kann ein Terminalfenster direkt im unteren Bereich angezeigt werden.

1. Tutorial zu UART auf mikrocontroller.net, weiteres Tutorial zu UART auf mikrocontroller.net
2. Datenpaket für RX/TX
3. Leitungslänge vs. Übertragungsrate

die einfache Art der Kommunikation

Ziele

Nach dieser Lektion sollten Sie:

1. wissen, wie eine UART Kommunikation hergestellt wird

Video

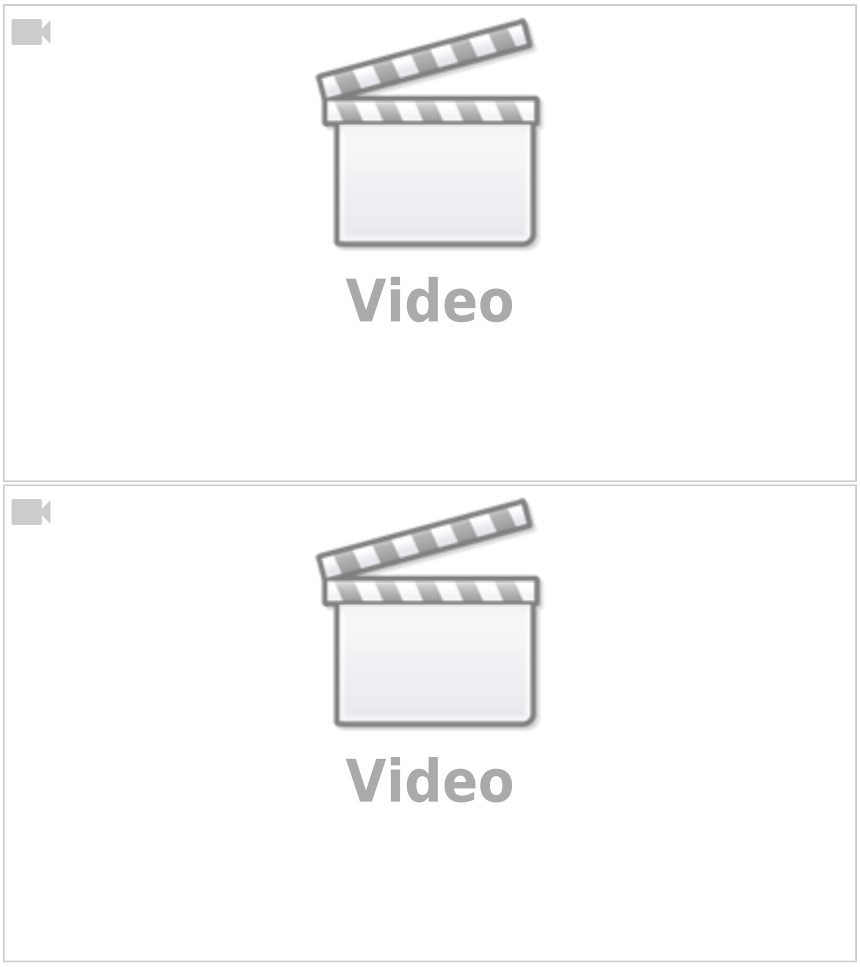


Fig. 1: Zusammenspiel der UART-Register

uC1 - here only as transmitter

I. Vorarbeiten

1. Laden Sie folgende Datei herunter:
    1. 9_temperatureuart_1.0.0.sim1
    2. 9_uart.hex
    3. lcd_lib_de.h
       ACHTUNG: Das Display ist hier an einem anderen Port. Entsprechend müssen zwei Register geändert werden:
        1. `#define DDR_DATA DDR`**`C`**
        2. `#define PORT_DATA PORT`**`C`**

> **Beachten Sie folgendes**
>
> - Es wird nun ein ATmega328 genutzt, d.h. das Programm ist nicht mehr kompatibel mit dem MiniMEXLE!
> - Überprüfen Sie die Pins und Ports des Displays.
> - Überprüfen Sie die Taktfrequenz.

Code mit Polling

Code mit Polling

Dieses Unterkapitel ist z.Zt. in einem provisorischem Zustand.

Variante mit Polling: 9_uartpolling_temperature.c

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

 Experiment 9:  Temperaturmessung mit UART (Polling)
 =============    ================================

 Dateiname: 9_Temperature.c

 Autoren    : Tim Fischer
 Datum      : 29.10.2020

 Version    : 0.1

 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)
            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 ad_wert gespeichert.

 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-wert           AD-Wandlung => Umrechnung
                        t-wert            Umrechnung => Anzeige
                        tmax-wert        Umrechnung => Anzeige

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

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

// Festlegung der Quarzfrequenz
#ifndef F_CPU                    // optional definieren
#define F_CPU 8000000L //12288000UL     // MiniMEXLE mit 12,288 MHz
Quarz
#endif

// Include von Header-Dateien
#include <avr/io.h>               // Header-Dateien zum ATmega88
#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
#include <string.h>

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

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

#define UART_BUFFER_SIZE 10             // Buffersize für übertrgene
Daten
#define BAUD_RATE 9600L

// Berechnungen für die Baudrate
#define UBRR_VAL  ((F_CPU+BAUD_RATE*8)/(BAUD_RATE*16)-1)    // clever
runden
#define BAUD_REAL (F_CPU/(16*(UBRR_VAL+1)))                 // Reale
Baudrate
#define BAUD_ERROR ((BAUD_REAL*1000)/BAUD_RATE-1000)        // Fehler
in Promille

// Überprüfung der Baudrate und ob Fehler zu groß ist
#if ((BAUD_ERROR>10) || (BAUD_ERROR<-10))
#error Systematischer Fehler der Baudrate zu hoch!
#endif
```

```
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   vorteiler     = VORTEILER_WERT;        // Zaehlvariable
Vorteiler
unsigned char   hundertstel    = HUNDERTSTEL_WERT;
unsigned char   zehntel =       ZEHNTEL_WERT;

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

char wort[UART_BUFFER_SIZE];    // Variable für die Ein- und Ausgabe
von Wörtern und Zahlen


//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);

void initUart(void);            // Initialisierung der UART-Register
void putData(char *daten);      // Daten senden
uint8_t getData(void);              // Daten empfangen

// Hauptprogramm
============================================================
int main ()
{
```

```
    uint8_t bufferFilled = 0;    // Flag für vollen Puffer

    initDisplay();                // Initialisierung LCD-Anzeige
    initTimer0();                // Initialisierung von Timer0
    initAdc();                    // Initialisierung des AD-Wandlers

    initUart();                    // UART initialisieren
    sei();                        // generell Interrupts einschalten
    // Hauptprogrammschleife
_____________________________________________________

    while(1)                    // unendliche Warteschleife mit Aufruf
der
                                // Funktionen abhaengig von
Taktbotschaften
    {
        if(takt10ms)
        {
            takt10ms = 0;        // Taktbotschaft zuruecksetzen

            if(getData()=='r') tValueMax = tValue;    // wenn r
eingegeben, dann ValueMax zurücksetzen

            if (bufferFilled==1) // Bei verfügbarer Zeichenkette
            {
                putData(wort);    // Wort zurück senden
                bufferFilled=0;    // Puffer leeren
            }

        }
        if(takt100ms)            // Durchfuehrung der Funktion einmal
pro 100ms
        {
            takt100ms = 0;        // Taktbotschaft zuruecksetzen
            doAdc();            // Ausfuehrung des Modules der A/D-
Wandlung
            calculateTemp();    // Ausfuehrung des Modules der
Umrechnung
        }

        if(takt1s)                // Durchfuehrung der Anzeige einmal
pro 1s
        {
            takt1s = 0;            // Taktbotschaft zuruecksetzen

            refreshDisplay();    // Ausfuehrung des Modules der Anzeige
            wort[5] = '°';
            wort[6] = 'C';
            wort[7] = '\n';        // ...New Line und...
            wort[8] = '\r';        // ...Carriage Return.
            wort[9] = 0;        // Endzeichen
```

```
            bufferFilled=1;
        }
    }
}


// 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
}

// Timer-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)
            |  (1<<MUX0)
            |  (1<<MUX2);        // Vref =AVCC; ADC5

    ADCSRA    |= (1<<ADPS0)
              |  (1<<ADPS1)
              |  (1<<ADPS2)
              |  (1<<ADEN);        // Teiler 128; ADC ON
}


// Timer-Initialisierung
================================================================
//
// 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)
{
    --vorteiler;                          // Vorteiler dekrementieren
    if (vorteiler==0)                      // wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler = VORTEILER_WERT;        // Vorteiler auf Startwert
        takt10ms = true;                  // Botschaft 10ms senden
        --hundertstel;                     // Hundertstelzaehler
dekrementieren

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

            if (zehntel==0)                // Zehntelzaehler
dekrementieren
            {
                zehntel = ZEHNTEL_WERT;    // Teiler auf Startwert
                takt1s = 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
    }
}

// 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 '-'
```

```
    {
        wort[0] = ' ';
    }
    else
    {
        wort[0] = '-';
        tempValue = -tempValue;                // Vorzeichenumkehr bei
negativer Zahl
    }
    wort[1] = tempValue/100 + ASC_NULL;
    tempValue = tempValue%100;
    wort[2] = tempValue/10 + ASC_NULL;
    wort[3] = ASC_FULL_STOP;
    wort[4] = tempValue%10 + ASC_NULL;
    wort[5] = 0;
    lcd_putstr   (wort);    // Hunderter ausgeben (°C Zehner)
}

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

// Initialisierung Display-Anzeige
=============================================================
//
void initDisplay()                  // Start der Funktion
{
    lcd_init();                        // Initialisierungsroutine aus
der lcd_lib
    lcd_displayMessage("- Experiment 9b-",0,0); // Ausgabe in erster
Zeile
    lcd_displayMessage("UART-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
```

```
}                                       // Ende der Funktion

void initUart (void)    // Initialisierung der UART-Register
{
    UBRR0H = UBRR_VAL >> 8;         // High-Baudraten-Register
beschreiben
    UBRR0L = UBRR_VAL & 0xFF;    // Low-Baudraten-Register beschreiben
    UCSR0B = (1<<RXEN0 ) | (1<<TXEN0 );                // Senden und
Empfangen ermoeglichen
    UCSR0C = (1<<UCSZ00) | (1<<UCSZ01);                // Setzen der
Datenbits auf 8-Bit
}


/* puts ist unabhaengig vom Controllertyp */
void putData (char *data)
{
    while (*data)
    {   /* so lange *s != '\0' also ungleich dem "String-
Endezeichen(Terminator)" */
        while (!(UCSR0A & (1<<UDRE0))){}  /* warten bis Senden moeglich
*/
        UDR0 = *data;                      /* sende Zeichen */
        data++;
    }
}

uint8_t getData(void)   // Daten empfangen
{
    if  (UCSR0A & (1<<RXC0))   // wenn Zeichen verfuegbar
    {
        return UDR0;           // Zeichen aus UDR an Aufrufer
zurueckgeben
    }else{
        return 0;
    };
}
```

Code mit Interrupt

Code mit Interrupt

Variante mit Interrupt: 9_uart_temperature.c

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

```
 Experiment 9:     Temperaturmessung und UART
 =============    ==============================

 Dateiname  : 9_UART_Temperature.c

 Autoren    : Tim Fischer
 Datum      : 01.05.2020

 Version    : 0.1

 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)
            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 ad_wert gespeichert.

 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-wert            AD-Wandlung => Umrechnung
                         t-wert            Umrechnung => Anzeige
                         tmax-wert        Umrechnung => Anzeige

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

// Deklarationen
```

```
============================================================

// Festlegung der Quarzfrequenz
#ifndef F_CPU                    // optional definieren
#define F_CPU 8000000L //12288000UL     // MiniMEXLE mit 12,288 MHz
Quarz
#endif

// Include von Header-Dateien
#include <avr/io.h>               // Header-Dateien zum ATmega88
#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
#include <string.h>

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

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

#define UART_BUFFER_SIZE 10             // Buffersize fuer uebertragene
Daten
#define BAUD_RATE 9600L

// Berechnungen für die Baudrate
#define UBRR_VAL  ((F_CPU+BAUD_RATE*8)/(BAUD_RATE*16)-1)    // clever
runden
#define BAUD_REAL (F_CPU/(16*(UBRR_VAL+1)))                 // Reale
Baudrate
#define BAUD_ERROR ((BAUD_REAL*1000)/BAUD_RATE-1000)        // Fehler
in Promille

// Überprüfung der Baudrate und ob Fehler zu groß ist
#if ((BAUD_ERROR>10) || (BAUD_ERROR<-10))
#error Systematischer Fehler der Baudrate zu hoch!
#endif


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

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

volatile uint8_t uart_rx_flag = 0;          // Flag für vollständig
empfangene Daten
volatile uint8_t DataTransferFinished = 1;          // Flag für
vollständig gesendete Daten
char uart_rx_buffer[UART_BUFFER_SIZE];      // Empfangspuffer
char uart_tx_buffer[UART_BUFFER_SIZE];      // Sendepuffer
char wort[UART_BUFFER_SIZE];                // Variable für die Ein-
und Ausgabe von Wörtern und Zahlen


//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);

void initUart (void);           // Initialisierung der UART-Register
void putData(char *data);       // Daten senden

// Hauptprogramm
============================================================
int main ()
{
    uint8_t bufferFilled = 0;    // Flag für vollen Puffer

    initDisplay();                // Initialisierung LCD-Anzeige
    initTimer0();                // Initialisierung von Timer0
    initAdc();                    // Initialisierung des AD-Wandlers
```

```
    initUart();                     // UART initialisieren
    sei();                         // generell Interrupts einschalten
    UDR0 = 0;                     // Work around für Probleme bei
Simulide <= R941 : ohne diese Zeile is kein Senden möglich
    // Hauptprogrammschleife
_____________________________________________________

    while(1)                    // unendliche Warteschleife mit Aufruf
der
                                // Funktionen abhaengig von
Taktbotschaften
    {
        if(takt10ms)
        {
            takt10ms = 0;
            if (DataTransferFinished==1 && bufferFilled==1) // Bei
abgeschlossener Sendung und verfügbarer Zeichenkette
            {
                bufferFilled=0;        // Puffer leeren
                putData(wort);        // Wort zurück senden
            }

        }
        if(takt100ms)            // Durchfuehrung der Funktion einmal
pro 100ms
        {
            takt100ms = 0;        // Taktbotschaft zuruecksetzen
            doAdc();             // Ausfuehrung des Modules der A/D-
Wandlung
            calculateTemp();    // Ausfuehrung des Modules der
Umrechnung
        }

        if(takt1s)                // Durchfuehrung der Anzeige einmal
pro 1s
        {
            takt1s = 0;            // Taktbotschaft zuruecksetzen
            refreshDisplay();    // Ausfuehrung des Modules der Anzeige
            wort[5] = '°';
            wort[6] = 'C';
            wort[7] = '\n';        // ...New Line und...
            wort[8] = '\r';        // ...Carriage Return.
            wort[9] = 0;        // Endzeichen
            bufferFilled=1;
        }
    }
}


// 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
}

// Timer-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)
            |  (1<<MUX0)
            |  (1<<MUX2);        // Vref =AVCC; ADC5

    ADCSRA    |= (1<<ADPS0)
            |  (1<<ADPS1)
            |  (1<<ADPS2)
            |  (1<<ADEN);        // Teiler 128; ADC ON
}


// Timer-Initialisierung
=================================================================
//
// 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)
{
    --vorteiler;                          // Vorteiler dekrementieren
    if (vorteiler==0)                      // wenn 0 erreicht: 10ms
abgelaufen
    {
        vorteiler = VORTEILER_WERT;        // Vorteiler auf Startwert
        takt10ms = true;                  // Botschaft 10ms senden
        --hundertstel;                     // Hundertstelzaehler
dekrementieren

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

            if (zehntel==0)                // Zehntelzaehler
dekrementieren
            {
                zehntel = ZEHNTEL_WERT;    // Teiler auf Startwert
                takt1s = 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
    }
}

// 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 '-'
    {
        wort[0] = ' ';
    }
    else
    {
        wort[0] = '-';
        tempValue = -tempValue;               // Vorzeichenumkehr bei
negativer Zahl
    }
```

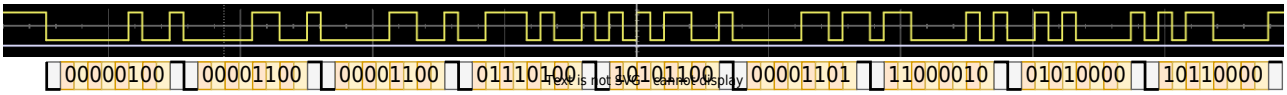
```
    wort[1] = tempValue/100 + ASC_NULL;
    tempValue = tempValue%100;
    wort[2] = tempValue/10 + ASC_NULL;
    wort[3] = ASC_FULL_STOP;
    wort[4] = tempValue%10 + ASC_NULL;
    wort[5] = 0;
    lcd_putstr   (wort);    // Hunderter ausgeben (°C Zehner)
}

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

// Initialisierung Display-Anzeige
=============================================================
//
void initDisplay()                  // Start der Funktion
{
    lcd_init();                        // Initialisierungsroutine aus
der lcd_lib
    lcd_displayMessage("- Experiment 9 -",0,0); // Ausgabe in erster
Zeile
    lcd_displayMessage("UART-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
}                                   // Ende der Funktion

void initUart (void)    // Initialisierung der UART-Register
{
    UBRR0H = UBRR_VAL >> 8;        // High-Baudraten-Register
beschreiben
    UBRR0L = UBRR_VAL & 0xFF;    // Low-Baudraten-Register beschreiben
    UCSR0B = (1<<TXEN0) | (1<<RXEN0) | (1<<RXCIE0);    // Aktivierung
des UART Senders, Empfaengers und der Interrupts
```

```
    UCSR0C = (1<<UCSZ00) | (1<<UCSZ01);                // Setzen der
Datenbits auf 8-Bit
}


ISR(USART_RX_vect) // Interrupt Routine für das Empfangen
{
    char data = UDR0;        // Daten auslesen, um Interruptflag zu
loeschen
    if (data=='r') tValueMax = tValue;
}

ISR(USART_UDRE_vect) // Interrupt Routine für das Senden
{
    static char* uart_tx_p = uart_tx_buffer;    // Zeiger auf
Sendepuffer
    char data = *uart_tx_p++;                 // Daten einlesen und
Pointer erhöhen
    if (data==0 )                          // Am Ende der Eingabe
    {
        UCSR0B &= ~(1<<UDRIE0);             // UDRE Interrupt
ausschalten
        uart_tx_p = uart_tx_buffer;         // Pointer zurücksetzen
        DataTransferFinished = 1;         // Flag für Senden setzen
    }
    else UDR0 = data;                      // Daten senden
}

void putData(char *data)    // Daten senden
{
    if (DataTransferFinished==1)         // Wenn Daten vollstaendig
gesendet
    {
        DataTransferFinished = 0;         // Flag für Senden löschen
        strcpy(uart_tx_buffer, data);     // Daten in Sendepuffer
kopieren
        UCSR0B |= (1<<UDRIE0);             // UDRE Interrupt einschalten
    }
}
```

Unvollständige Falstad Simulation

Bitte arbeiten Sie folgende Aufgaben durch:

Aufgabe



1. Versuchen Sie die obenstehende Zeitverlauf des UART Signals zu entschlüsseln - es kam vom vorliegenden Programm.
   Welche neun Zeichen wurden gesendet?