

/\*-----\*/

Wetterstation  
(C)2006-2012 Hardy Lau

Microchip PIC18F2550  
ICD2/ICD3  
Externer Oszillator mit 25 MHz

- Sensirion SHT75
- Smartec SMT 160
- Smartec UTI (PT100)
- Impulsiingang mit Zaehler fuer Regenschirm
- Eingang fuer Regenmelder
- Zweiter Impulsiingang fuer Regenschirm

V1.0 9.06.2006 Initialversion  
19.06.2006 Sensor und Linearisierung funktionieren  
4.07.2006 RS485 Kommunikation funktioniert  
18.07.2006 SMT160-30 Sensor zugefügt  
20.07.2006 SMT160-30 Temperaturmessung funktioniert  
26.07.2006 Umbau auf 18F2550 wegen zu wenig RAM  
CRC-Ueberpruefung bei SHT75-Kommunikation  
14.09.2006 Zaehler fuer Regenwippe, Regenmelder, Datenausgabe alle Sekunde  
8.03.2011 Neue Koeffizienten fuer V4-Sensoren, SHT75-Timing etwas langsamer  
Timer2 fuer genauen Sekundentakt der Ausgabe  
14.03.2011 Zeitablauf etwas geaendert und Temperatur- bzw. Feuchtemessung  
jede zweite Sekunde im Wechsel  
17.03.2011 Fehler in Datenkommunikation (ACK) gefunden  
18.03.2011 Anzahl der Messungen reduziert. (Feuchte/Temperatur/Pause/Pause)  
23.03.2011 PT100-Messung mit UTI  
7.11.2011 Timing Aenderung SMT160, Probleme mit SHT75-Schlaf  
3.04.2012 Zusaeztlicher Eingang fuer zweiten Regenschirm KBI0  
16.04.2012 Softreset des SHT75 falls Daten CRC-Richtig aber unplaesibel  
SHT75 wieder im Sekundenwechsel (Feuchte/Temperatur) lesen

-----\*/  
#include <p18f2550.h>  
#include <math.h>  
#include <stdio.h>

/\*-----\*/

enum {TEMP, HUMID};

#define DATA PORTAbits.RA1  
#define DATA\_RICHTUNG TRISAbits.TRSA1  
#define SCK LATAbits.LATA3  
  
#define noACK 0  
#define ACK 1  
  
#define STATUS\_REG\_W 0x06 //adr command r/w  
#define STATUS\_REG\_R 0x07 //000 0011 1  
#define MEASURE\_TEMP 0x03 //000 0001 1  
#define MEASURE\_HUMID 0x05 //000 0010 1  
#define RESET 0x1e //000 1111 0  
  
#define RS485\_SENDEN LATAbits.LATA2  
#define TAKT 25000000.0 //12 MHz fuer BAUDRate  
#define BAUD 19200.0 //Baudrate fuer RS485

#define REGENMELDER PORTAbits.RA5

volatile char uart\_tx\_buffer7[5];  
volatile char uart\_tx\_buffer6[6];  
volatile char uart\_tx\_buffer5[5];  
volatile char uart\_tx\_buffer4[1];  
volatile char uart\_tx\_buffer3[6];  
volatile char uart\_tx\_buffer2[6];  
volatile char uart\_tx\_buffer1[6];

volatile int start\_mitte\_stop\_ok;  
volatile unsigned int start, mitte, stop; //Zaehlerstaende CCP  
volatile unsigned int regenzaehler = 0;  
volatile unsigned char regen = 1;

volatile unsigned int ccp2\_captures\_zeiger = 0; //Ringpuffer mit CCP2-Capturewerte

```

volatile unsigned int ccp2_captures[16];
volatile char sekunde = 0; //Marker wird jede Sekunde gesetzt
volatile unsigned int regenzaehler2 = 0;

//-----
//-----
// ISR Setup

void t2_rb_isr( void);
void ccp_isr( void);

#pragma code high_vector=0x08
void high_interrupt( void)
{
    _asm GOTO ccp_isr _endasm
}

#pragma code low_vector=0x18
void low_interrupt( void)
{
    _asm GOTO t2_rb_isr _endasm
}

//-----
//-----
// CCP1 + CCP2 Interrupt Service Routines
#pragma code
#pragma interrupt ccp_isr
void ccp_isr( void)
{
    //CCP1-Interrupt
    if( PIR1bits.CCP1IF) {
        if( CCP1CON == 0b00000101) { //Capture Mode every rising edge
            CCP1CON = 0b00000100; //Capture Mode every falling edge
            if(start_mitte_stop_ok == 0)
                start = CCPR1;
            else
                stop = CCPR1;
        } else {
            CCP1CON = 0b00000101; //Capture Mode every rising edge
            if( start_mitte_stop_ok == 1)
                mitte = CCPR1;
        }
        start_mitte_stop_ok += 1;
        if( start_mitte_stop_ok > 2) {
            PIE1bits.CCP1IE = 0; //Disable CCP1-Interrupt
        }
        PIR1bits.CCP1IF = 0;
    }

    //CCP2-Interrupt
    if( PIR2bits.CCP2IF) {
        ccp2_captures[ ccp2_captures_zeiger++] = CCPR2;
        ccp2_captures_zeiger &= 15;
        PIR2bits.CCP2IF = 0;
    }
}

//-----
//-----
// Timer2 + RB Port Change Interrupt Service Routines
#pragma code
#pragma interrupt t2_rb_isr
void t2_rb_isr( void)
{
    static char x = 0;
    char y;

    // Timer2 Interrupt
    if( PIR1bits.TMR2IF) {
        if( ++x >= 100) {
            sekunde = 1;
            x = 0;
        }
        PIR1bits.TMR2IF = 0;
    }
}

```



```

    return( error);                //error=1 in case of no acknowledge
}

//-----
char s_read_byte(unsigned char ack)
//-----
// reads a byte form the Sensibus and gives an acknowledge in case of "ack=1"
{
    static unsigned char i, val;

    val = 0;
    DATA_RICHTUNG = 1;            //release DATA-line
    delay();
    for (i=0x80; i>0; i>>=1)      //shift bit for masking
    { SCK=1;                       //clk for SENSI-BUS
      delay();
      if (DATA) val=(val | i);     //read bit
      SCK=0;
      delay();
    }
    DATA=!ack;                   //in case of "ack==1" pull down DATA-Line
    DATA_RICHTUNG = 0;
    delay();
    SCK=1;                         //clk #9 for ack
    delay();                       //pulswith approx. 5 us
    if( !ack) DATA_RICHTUNG = 1;  //Bei ACK-Antwort rechtzeitig Treiber abschalten!
    SCK=0;
    delay();
    DATA_RICHTUNG = 1;
    return( val );
}

//-----
void s_transstart(void)
//-----
// generates a transmission start
//
// DATA: _____|_____|_____
//
// SCK :  ___|__|___|__|___|_____
{
    DATA=1;
    DATA_RICHTUNG = 0;
    delay();
    SCK=0;                          //Initial state
    delay();
    SCK=1;
    delay();
    DATA=0;
    delay();
    SCK=0;
    delay();
    SCK=1;
    delay();
    DATA=1;
    delay();
    SCK=0;
    DATA_RICHTUNG = 1;
    delay();
}

//-----
void s_connectionreset(void)
//-----
// communication reset: DATA-line=1 and at least 9 SCK cycles followed by transstart
//
// DATA: _____|_____|_____
//
// SCK :  ___|__|___|__|___|__|___|__|___|__|___|_____|__|___|__|___|_____
{
    unsigned char i;
    DATA=1;
    DATA_RICHTUNG = 0;
    delay();
    SCK=0;                          //Initial state
    delay();
    for(i=0; i<9; i++)              //9 SCK cycles
    { SCK=1;

```

```

    delay();
    SCK=0;
    delay();
}
DATA_RICHTUNG = 1;
delay();
s_transstart();          //transmission start
}

//-----
char s_softreset(void)
//-----
// resets the sensor by a softreset
{
    static unsigned char error;

    error = 0;
    s_connecti onreset();          //reset communication
    error+=s_wri te_byte(RESET);    //send RESET-command to sensor
    return( error);                //error=1 in case of no response form the sensor
}

//-----
char s_read_statusreg(unsigned char *p_val ue, unsigned char *p_checksum)
//-----
// reads the status register with checksum (8-bit)
{
    static unsigned char error;

    error = 0;
    s_transstart();                //transmission start
    error=s_wri te_byte(STATUS_REG_R); //send command to sensor
    *p_val ue=s_read_byte(ACK);      //read status register (8-bit)
    *p_checksum=s_read_byte(noACK);  //read checksum (8-bit)
    return( error);                //error=1 in case of no response form the sensor
}

//-----
char s_wri te_statusreg(unsigned char *p_val ue)
//-----
// writes the status register with checksum (8-bit)
{
    static unsigned char error;

    error = 0;
    s_transstart();                //transmission start
    error+=s_wri te_byte(STATUS_REG_W); //send command to sensor
    error+=s_wri te_byte(*p_val ue);  //send value of status register
    return( error);                //error>=1 in case of no response form the sensor
}

//-----
char calc_crc(unsigned char val ue, unsigned char ini t)
//-----
// calculates checksum (8-bit)
{
    static unsigned char crc = 0;
    unsigned char i, j;
    static unsigned char returnwert;

    returnwert = 0;
    if( ini t != 0) {
        crc = val ue;
        return( 0);
    }
    else {
        for( i = 0b10000000; i > 0; i>>=1) {
            if( ((val ue & i) && !( crc & 0b10000000)) || ( !(val ue & i) && ( crc &
0b10000000))) {
                crc <<= 1;
                crc ^= 0b00110000;
                crc |= 1;
            } else {
                crc <<= 1;
            }
        }
    }
}
}

```

```

j = 0b00000001;
for( i = 0b10000000; i; i>>=1, j<<=1)
    if( crc & i) returnwert |= j;
return( returnwert);
}

//-----
unsigned int s_measure(unsigned int *p_value, unsigned char mode)
//-----
// makes a measurement (humidity/temperature) with checksum
{
    static unsigned int error;
    unsigned int i;
    unsigned char highbyte, lowbyte, crc;
    unsigned char checksum;

    error = 0;
    calc_crc( 0, 1); //CRC Initialisieren
    s_transstart(); //transmission start
    switch(mode){ //send command to sensor
        case TEMP : error+=s_write_byte(MEASURE_TEMP);
                    crc = calc_crc( MEASURE_TEMP, 0);
                    break;
        case HUMI : error+=s_write_byte(MEASURE_HUMI);
                    crc = calc_crc( MEASURE_HUMI, 0);
                    break;
        default : break;
    }
    for( i = 10; i; i--) {
        delay();
    }
    for ( i = 18000; i; i--) { //wait until sensor has finished the measurement
        if(!DATA) break;
        delay();
    }
    if( DATA) error++; // or timeout (~0,5 sec.) is reached
    highbyte = s_read_byte(ACK); //read the first byte (MSB)
    crc = calc_crc( highbyte, 0);
    lowbyte = s_read_byte(ACK); //read the second byte (LSB)
    crc = calc_crc( lowbyte, 0);
    *p_value = (unsigned int)highbyte << 8 | (unsigned int)lowbyte;
    checksum = s_read_byte( noACK); //read checksum
    if( checksum != crc) error += 10;
    return( error);
}

//-----
void init_uart(void)
//-----
// UART auf "BAUD" Geschwindigkeit, 8N1 Programmieren
{
    TRISCbits.TRISC6 = 1;
    TRISCbits.TRISC7 = 1;
    TXSTA = 0b01100101;
    RCSTA = 0b10010000;
    BAUDCONbits.BRG16 = 1;
    SPBRGH = (unsigned int)( ((TAKT / BAUD) / 4.0) - 1 ) >> 8;
    SPBRG = (unsigned int)( ((TAKT / BAUD) / 4.0) - 1 ) & 0xff;
    PIE1bits.RCIE = 0; //Kein Empfangsinterrupt !!!
    PIE1bits.TXIE = 0;
    INTCONbits.GIE = 1;
    INTCONbits.GIEL = 1;
    RS485_SENDEN = 0;
}

//-----
void double2string( double* wert_f, volatile char string[])
//-----
// Formatiert einen double in einen ASCII-String
//
{
    int i;
    long wert;

    wert = (long)(*wert_f * 100.0);
    if( wert >= 0) string[0] = '+';
    else {
        string[0] = '-';
    }
}

```

```

    wert = -wert;
}
for( i = 4; i >= 0; i--) {
    string[i+1] = (wert % 10) + '0';
    wert /= 10;
    if( i == 3) i--;
}
string[3] = '.';
}

//-----
void uint2string( unsigned int wert, volatile char string[])
//-----
// Formatiert einen Unsigned Int in einen ASCII-String
//
{
    int i;

    for( i = 4; i >= 0; i--) {
        string[i] = (wert % 10) + '0';
        wert /= 10;
    }
}

//-----
void prepare_data( double* temp, double* humi, double* smt_temp, unsigned int
regenzaehler,
                unsigned char regen, double* temp2, unsigned int regenzaehler2)
//-----
// Die formatierten Daten in Puffern zur Verfuegung stellen
//
{
    double2string( temp, uart_tx_buffer1);
    double2string( humi, uart_tx_buffer2);
    double2string( smt_temp, uart_tx_buffer3);
    if( regen) {
        uart_tx_buffer4[0] = 'R';
    } else {
        uart_tx_buffer4[0] = 'T';
    }
    uint2string( regenzaehler, uart_tx_buffer5);
    double2string( temp2, uart_tx_buffer6);
    uint2string( regenzaehler2, uart_tx_buffer7);
}

//-----
void init_porta(void)
//-----
// PORTA konfigurieren
{
    ADCON1 = 0b01111111; //Alle Ports sind Digital-I/O
    TRISAbits.TRI SA2 = 0; //Datenrichtung fuer RS485 Ausgang
    RS485_SENDEN = 0; //Empfangen
    LATAbits.LATA1 = 1;
    LATAbits.LATA3 = 0;
    TRISAbits.TRI SA3 = 0; //SCK Clock fuer SHT75 Kommunikation Ausgang
    TRISAbits.TRI SA1 = 1; //DATA Daten fuer SHT75 Kommunikation Ausgang
    TRISAbits.TRI SA5 = 1; //PORTA. 5 ist Eingang fuer Regenmelder
}

//-----
void init_ccp1(void)
//-----
// CCP1 konfigurieren mit Timer1 als 16 Bit Timer
{
    TRIScbits.TRI SC2 = 1; //CCP1 (RC2) ist Capture-Eingang
    T1CONbits.TMR1CS = 0; //Interner Clock Fosc/4
    T3CONbits.T3CCP1 = 1; //Timer 1 wird fuer CCP1 benutzt / Timer 2 wird fuer CPP2 benutzt
    T3CONbits.T3CCP2 = 0;
    T1CONbits.T1CKPS0 = 0; //1:1 Prescaler
    T1CONbits.T1CKPS1 = 0;
    T1CONbits.RD16 = 1;
    T1CONbits.TMR1ON = 1; //Timer 1 on
    CCP1CON = 0b00000101; //Capture Mode every rising edge
}

//-----
void init_ccp2(void)

```

```

//-----
// CCP2 konfigurieren mit Timer3 als 16 Bit Timer (:8)
{
  TRISCbits.TRISC1 = 1; //CCP2 (RC1) ist Capture-Eingang
  T3CONbits.TMR3CS = 0; //Interner Clock Fosc/4
  T3CONbits.T3CCP1 = 1; //Timer 1 wird fuer CCP1 benutzt / Timer 2 wird fuer CPP2 benutzt
  T3CONbits.T3CCP2 = 0;
  T3CONbits.T3CKPS0 = 1;
  T3CONbits.T3CKPS1 = 1; //Prescaler :8
  T3CONbits.RD16 = 1;
  T3CONbits.TMR3ON = 1; //Timer 3 on
  CCP2CON = 0b00000101; //Capture Mode every rising edge
  PIR2bits.CCP2IF = 0;
  PIE2bits.CCP2IE = 1; //Enable CCP2-Interrupt
}

//-----
void init_interrupts(void)
//-----
// Interruptskonfigurieren
{
  RCONbits.IPEN = 1; //Interrupt mit Prioritaete
  IPR1bits.CCP1IP = 1; //CCP1 high priority
  IPR2bits.CCP2IP = 1; //CCP2 high priority
  IPR1bits.TMR2IP = 0; //Timer2 low priority;
  INTCONbits.RBIP = 0; //RB Port Change Interrupt Priority low
  INTCONbits.GIEH = 1;
  INTCONbits.GIEL = 1;
}

//-----
void init_kb(void)
//-----
// Port B fuer Interrupt on Change KIB1
{
  int x;
  TRISBbits.TRISB7 = 0; //RB5 KIB0 ist Ausgang
  TRISBbits.TRISB6 = 0; //RB5 KIB0 ist Ausgang
  TRISBbits.TRISB5 = 1; //RB5 KIB0 ist Eingang
  TRISBbits.TRISB4 = 0; //RB5 KIB0 ist Ausgang
  x = PORTBbits.RB5;
  INTCONbits.RBIF = 0;
  INTCONbits.RBIE = 1; //Enable KIB-Interrupt
}

//-----
void init_timer0(void)
//-----
// Timer0 fuer Regenwippe initialisieren
{
  TOCONbits.T08BIT = 0; //16Bit-Zaehler
  TOCONbits.TOCS = 1; //Zaehler fuer TOCKI-Eingang
  TOCONbits.TOSE = 1; //Fallende Flanke wird gezaehlt
  TOCONbits.PSA = 1; //Kein Prescaler
  TMROH = 0;
  TMROL = 0;
  TOCONbits.TMROON = 1; //Zaehler starten
}

//-----
void init_timer2(void)
//-----
// Timer2 gibt jede 10ms Sekunde einen Interrupt
{
  T2CONbits.T2OUTPS0 = 1; //: 16
  T2CONbits.T2OUTPS1 = 1;
  T2CONbits.T2OUTPS2 = 1;
  T2CONbits.T2OUTPS3 = 1;
  T2CONbits.T2CKPS1 = 1; //: 16
  TMR2 = 0;
  PR2 = (unsigned char)((long)TAKT / 4 / 16 / 16 / 100); // 100 Interrupts pro Sekunde
  PIR1bits.TMR2IF = 0;
  PIE1bits.TMR2IE = 1;
  T2CONbits.TMR2ON = 1; //Zaehler starten
}

//-----
double messe_smt160(void)

```

```

//-----
// CCP1 Modul zur Messung mit SMT160-30 benutzen
{
  int i;
  int timeout;
  unsigned long l_on = 0, l_periode = 0;

  for( i = 0; i < 400; i++) {
    TMR1L = 0; //Timer 1 auf Null setzen
    TMR1H = 0;
    TMR1L = 0;
    start_mitte_stop_ok = 0;
    start = mitte = stop = 0;
    CCP1CON == 0b00000101; //Capture steigende Flanke
    PIR1bits.CCP1IF = 0;
    PIE1bits.CCP1IE = 1; //Enable CCP1-Interrupt

    timeout = 600; //War 500; Verlaengert
    while(start_mitte_stop_ok < 3) { //Warten bis alle drei Flanken gemessen oder Timeout!
      if(!--timeout) {
        PIE1bits.CCP1IE = 0; //Disable CCP1-Interrupt (zur Sicherheit)
        return( -99.99); //Timeout - Kein Sensorsignal!
      }
    }
    if( (start < mitte) && (mitte < stop)) {
      l_on += (mitte-start);
      l_periode += (stop-start);
    }
  }
  return ((double)(l_on) / (double)(l_periode) - 0.3200) * 212.7659574;
}

```

```

//-----
double pt100_to_temp( double pt100)
//-----
// Berechnet Temperatur aus PT100-Widerstandswert
// Tabelle (4te Ordnung) und Interpolation
{

```

```

  static double temp;
  static rom double rpt100[] = {
/* -30 */ 88.22278618,
/* -29 */ 88.61706639,
/* -28 */ 89.01122689,
/* -27 */ 89.40526795,
/* -26 */ 89.79918985,
/* -25 */ 90.19299285,
/* -24 */ 90.58667722,
/* -23 */ 90.98024319,
/* -22 */ 91.37369101,
/* -21 */ 91.7670209,
/* -20 */ 92.16023307,
/* -19 */ 92.55332774,
/* -18 */ 92.94630509,
/* -17 */ 93.33916531,
/* -16 */ 93.73190859,
/* -15 */ 94.12453507,
/* -14 */ 94.51704493,
/* -13 */ 94.9094383,
/* -12 */ 95.30171533,
/* -11 */ 95.69387613,
/* -10 */ 96.08592082,
/* -9 */ 96.47784951,
/* -8 */ 96.86966229,
/* -7 */ 97.26135925,
/* -6 */ 97.65294046,
/* -5 */ 98.04440599,
/* -4 */ 98.43575589,
/* -3 */ 98.82699022,
/* -2 */ 99.21810899,
/* -1 */ 99.60911225,
/* 0 */ 100.0,
/* 1 */ 100.3907723,
/* 2 */ 100.781429,
/* 3 */ 101.1719703,
/* 4 */ 101.562396,
/* 5 */ 101.9527063,
/* 6 */ 102.342901,
/* 7 */ 102.7329803,

```

```

/* 8 */ 103.122944,
/* 9 */ 103.5127923,
/* 10 */ 103.902525,
/* 11 */ 104.2921423,
/* 12 */ 104.681644,
/* 13 */ 105.0710303,
/* 14 */ 105.460301,
/* 15 */ 105.8494563,
/* 16 */ 106.238496,
/* 17 */ 106.6274203,
/* 18 */ 107.016229,
/* 19 */ 107.4049223,
/* 20 */ 107.7935,
/* 21 */ 108.1819623,
/* 22 */ 108.570309,
/* 23 */ 108.9585403,
/* 24 */ 109.346656,
/* 25 */ 109.7346563,
/* 26 */ 110.122541,
/* 27 */ 110.5103103,
/* 28 */ 110.897964,
/* 29 */ 111.2855023,
/* 30 */ 111.672925,
/* 31 */ 112.0602323,
/* 32 */ 112.447424,
/* 33 */ 112.8345003,
/* 34 */ 113.221461,
/* 35 */ 113.6083063,
/* 36 */ 113.995036,
/* 37 */ 114.3816503,
/* 38 */ 114.768149,
/* 39 */ 115.1545323,
/* 40 */ 115.5408,
/* 41 */ 115.9269523,
/* 42 */ 116.312989,
/* 43 */ 116.6989103,
/* 44 */ 117.084716,
/* 45 */ 117.4704063,
/* 46 */ 117.855981,
/* 47 */ 118.2414403,
/* 48 */ 118.626784,
/* 49 */ 119.0120123,
/* 50 */ 119.397125 };
static int i;

```

```

//Tabelle durchsuchen
for( i = 0; i < 80; i++) {
    if( pt100 < rpt100[i]) break;
}
i -= 1;

```

```

//Fehler?
if( i < 0 || i > 79)
return( -99.99);

```

```

//Interpolation
temp = (double)(i - 30) + ( pt100 - rpt100[i]) / ( rpt100[i+1] - rpt100[i]);
return( temp);
}

```

```

//-----
double messe_uti (void)
//-----
// CCP2 Modul zur Messung mit UTI (PT100 mit UTI Slowmode 5) benutzen
{
    static unsigned int i, j;
    static unsigned int t_bc_v0, t_cd_v0, t_ab_v0, t_v0;
    static double r_pt100;

    //Wenn Ringspeicher nicht gefuehlt - kein Sensor!
    if( !ccp2_captures[0] && !ccp2_captures[1]) {
        return( -99.99);
    }
}

```

```

INTCONbits.GIE = 0; //Interrupts sperren

```

```

//CCP2-Ringpuffer durchsuchen

```

```

j = (ccp2_captures_zeiger - 1) & 15;
for( i = 0; i < 16; i++, j = (j - 1) & 15) {
    if( (ccp2_captures[ j] - ccp2_captures[ (j-1) & 15]) < 10000 &&
        (ccp2_captures[ (j-1) & 15] - ccp2_captures[ (j-2) & 15]) < 10000) { //Neuer
Zyklus gefunden
        //t_bc_v0 = ccp2_captures[ (j-2) & 15] - ccp2_captures[ (j-3) & 15];
        t_cd_v0 = ccp2_captures[ (j-3) & 15] - ccp2_captures[ (j-4) & 15];
        t_ab_v0 = ccp2_captures[ (j-4) & 15] - ccp2_captures[ (j-5) & 15];
        t_v0 = ccp2_captures[ (j-5) & 15] - ccp2_captures[ (j-7) & 15];
        break;
    }
}

//Capture-Ringspeicher loessen damit Funktion des UTI erkannt werden kann;
for( i = 0; i < 16; i++)
    ccp2_captures[i] = 0;

INTCONbits.GIE = 1; //Interrupts erlauben

//Berechnung des Widerstandswertes des PT100 aus der UTI-Gleichung (Mode 5, 4Leiter)
r_pt100 = (double)( t_cd_v0 - t_v0) * 100.0 / (double)( t_ab_v0 - t_v0);

//Temperatur aus PT100-Widerstandswert berechnen
if( r_pt100 > 88.0 && r_pt100 < 120.0) {
    return( pt100_to_temp( r_pt100));
} else {
    return ( -99.99);
}
}

//-----
-
void calc_sth75(double *p_humidity ,double *p_temperature)
//-----
-
// calculates temperature [°C] and humidity [%RH]
// input :  humi [Ticks] (12 bit)
//          temp [Ticks] (14 bit)
// output:  humi [%RH]
//          temp [°C]
{ //const double C1=-4.0;           // for 12 Bit
  static const double C1=-2.0468;           // for 12 Bit V4
  //const double C2=+0.0405;           // for 12 Bit
  static const double C2=0.0367;           // for 12 Bit V4
  //const double C3=-0.0000028;        // for 12 Bit
  static const double C3=-1.5955E-6;       // for 12 Bit V4
  static const double T1=+0.01;          // for 14 Bit @ 5V
  static const double T2=+0.00008;       // for 14 Bit @ 5V

  static double rh;                       // rh:      Humidity [Ticks] 12 Bit
  static double t;                         // t:       Temperature [Ticks] 14 Bit
  static double rh_lin;                    // rh_lin:  Humidity linear
  static double rh_true;                   // rh_true: Temperature compensated
humidity
  static double t_C;                       // t_C      : Temperature [°C]

  rh=*p_humidity;
  t=*p_temperature;

  t_C = t*0.01 - 40.1 - (2.0e-8 * (t-7000.0) * (t-7000.0)); //calc. temperature from
ticks to [°C] 5V

  rh_lin=C3*rh*rh + C2*rh + C1;            //calc. humidity from ticks to [%RH]
  rh_true=(t_C-25)*(T1+T2*rh)+rh_lin;      //calc. temperature compensated humidity [%RH]
  if(rh_true>99.9) rh_true=99.9;           //cut if the value is outside of
  if(rh_true<0.1) rh_true=0.1;            //the physical possible range

  *p_temperature=t_C;                      //return temperature [°C]
  *p_humidity=rh_true;                     //return humidity[%RH]
}

//-----
void send_data( void)
//-----
{
  int i;

  //RS485 auf Senden schalten

```

```

RS485_SENDEN = 1;
for( i = 0; i < 1000; i++); //Warten bis RS485-Treiber bereit

//5 X-Zeichen
for( i = 0; i < 5; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = 'X';
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//SHT75-Temperatur
for( i = 0; i < 6; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer1[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//SHT75-Feuchtigkeit
for( i = 0; i < 6; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer2[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//SMT160-Temperatur
for( i = 0; i < 6; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer3[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//Regenmel der "R" oder "T"
for( i = 0; i < 1; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer4[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//Zaehlerstand des Regenmessers
for( i = 0; i < 5; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer5[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//PT100-Temperatur
for( i = 0; i < 6; i++) {
    Nop(); Nop(); Nop();

```

```

    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer6[i];
}

//Leerzeichen
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = ' ';

//Zaehlerstand des zweiten Regenmessers (RB5)
for( i = 0; i < 5; i++) {
    Nop(); Nop(); Nop();
    while(!PIR1bits.TXIF); //Warten bis TXREG leer
    TXREG = uart_tx_buffer7[i];
}

//Newline als Zeilenende
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
TXREG = '\n';

//RS485 Sender aus
Nop(); Nop(); Nop();
while(!PIR1bits.TXIF); //Warten bis TXREG leer
for( i = 0; i < 2000; i++);
RS485_SENDEM = 0;
}

//-----
void main(void)
//-----
{
    static unsigned int humi_val_i = 0, temp_val_i = -99;
    static double humi_val_f = 0, temp_val_f = -99.99;
    static double smt_temp_f = 0;
    static double pt100_temp_f = -99.99;
    static unsigned int i;
    static unsigned int error = 0;
    static char sekundenwahl = 0;

    init_interrupts(); //Interrupts mit Prioritaeten konfigurieren
    init_porta(); //PORTA konfigurieren
    init_uart(); //UART konfigurieren
    init_ccp1(); //Capture-Port fuer SMT160 Initialisieren
    init_ccp2(); //Capture-Port fuer UTI Initialisieren
    init_timer0(); //Zaehler fuer Eingang TOCKI (Regenwippe)
    init_timer2(); //Timer fuer Sekudentakt
    init_kb(); //Interrupt on Change an RB5/KBI 1

    for( i = 500; i; i--) delay();
    s_softreset(); //reset sht75 device
    for( i = 500; i; i--) delay();

    regenzaehler2 = 0;

    while(1)
    {
        //Sensirion SHT75 im 2-Sekunden-Rhythmus bearbeiten
        switch( sekundenwahl ) {
            case 0: error = 0;
                error += s_measure( &humi_val_i, HUMI ); //measure humidity
                if( error != 0 ) {
                    s_softreset(); //reset sht75 device
                    for( i = 500; i; i--) delay();
                }
                break;
            case 1: error += s_measure( &temp_val_i, TEMP ); //measure temperature
                if( error != 0 ) {
                    s_softreset(); //reset sht75 device
                    for( i = 500; i; i--) delay();
                }
                break;
            case 2: break; //Nicht genutzt
            case 3: break; //Nicht genutzt
            default: break; //Nicht genutzt
        }
        sekundenwahl = ++sekundenwahl % 2; //Wieder im Wechsel jede Sekunde lesen
    }
}

```

```

//Probleme mit Korrosion/Wasser - Notloesung
//Pullup reicht nicht - Wert "schimmt" und daher kann Sensor nicht schlafen
DATA=1;
DATA_RICHTUNG = 0; //DATA ist Ausgang

if( error == 0) {
    humi_val_f=(double)humi_val_i; //converts integer to double
    temp_val_f=(double)temp_val_i; //converts integer to double
    calc_sht75(&humi_val_f, &temp_val_f); //calculate humidity, temperature

    //Plausibilitaetspruefung
    if( humi_val_f < 5.0 || temp_val_f > 60.0 || temp_val_f < -40.0) {
        temp_val_f = -99.99;
        humi_val_f = 0.5;
        s_softreset(); //reset sht75 device
        for( i = 500; i; i--) delay();
    }
} else {
    temp_val_f = -99.99;
    humi_val_f = (double)error;
}

//Smartec SMT160 lesen
smt_temp_f = messe_smt160();

//Regenmelder einlesen
if( REGENMELDER) regen = 0;
else regen = 1;

//Zaehler fuer Regenmesser einlesen
regenzaehler = TMR0L;
regenzaehler += TMR0H * 256;

//Smartec UTI mit PT100 einlesen
pt100_temp_f = messe_uti();

//Daten in ASCII umwandeln und fuer Serielle Schnittstelle bereitstellen
prepare_data( &temp_val_f, &humi_val_f, &smt_temp_f, regenzaehler, regen,
&pt100_temp_f, regenzaehler2);

//Auf volle Sekunde warten (Timmer2 erledigt das)
while( !sekunde);
sekunde = 0;

//Daten auf RS485 senden
send_data();
}
}
/*-----*/

```