```
//-------------------------------------------------------------------------------

/*
   Elektronischer Barometer fuer Wetterstation

   (c) 8.02.2007 by Hardy Lau
   V1.0

   PIC 18F4550 mit externem 12 MHz Quarzgenerator

   MCP3551 als 22Bit-Delta-Sigma A/D-Wandler an SPI-Schnittstelle
   Freescale MPXA4100A als Drucksensor

   MCP3551 als 22Bit-Delta-Sigma A/D-Wandler an SPI-Schnittstelle
   Spannung von Rotorsteuerung ARS (0..5V)

   Eingebauter A/D-Wandler (AN1) ueberwacht die Versorgungsspannung
   des Rotors (Einschaltkontrolle).

   Ausgabe alle Sekunde auf Serieller Schnittstelle mit 19200 Baud 8n1

   Luftdruck in hPa und Richtung in Grad und Rotoreinschaltkontrolle

*/

#include <p18f4550.h>
#include <math.h>
#include <string.h>

#define TAKT 12000000.0 //12 MHz fuer BAUDRate
#define BAUD 19200.0 //Baudrate fuer RS232

#define EE_DRUCKKORREKTUR 0
#define EE_ADCCCW 4
#define EE_ADCCW 8
#define EE_WINKEL 12
#define EE_RICHTUNG 16

static double druckkorrektur; // Luftdruckkorrekturwert
static long adcccw; // A/D-Wert bei Linksanschlag CCW = 0Grad!
static long adccw; // A/D-Wert bei Rechtsanschlag CW = xGrad!
static long winkel; //Moeglicher Drehwinkel des Rotors in Grad (450)
static long richtung; //Stellung des Rotors in Grad

volatile unsigned char timer0 = 0;
volatile unsigned int sekundenzaehler = 0;

volatile char uart_rx_buffer[16];
volatile char uart_rx_count = 0;

//-------------------------------------------------------------------------------

// ISR Setup

void uart_isr( void);
void tmr0_isr( void);

#pragma code high_vector=0x08
void high_interrupt( void)
{
 _asm GOTO tmr0_isr _endasm
}

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

//-------------------------------------------------------------------------------

// Timer1 Interrupt Service Routine
#pragma code
#pragma interrupt tmr0_isr
void tmr0_isr( void)
```

```
{
 if( PORTBbits.RB4) //Gruene LED blinkt
   PORTBbits.RB4 = 0;
 else
   PORTBbits.RB4 = 1;
 TMR0H = 256 - ((long)TAKT / 4 / 256 / 256) - 1;
 TMR0L = 256 - (((long)TAKT / 4 / 256) % 256);
 INTCONbits.TMR0IF = 0;

 timer0 = 1; //Sekunde abgelaufen
 sekundenzaehler++; //Anzahl Sekunden seit Einschalten
}

//-------------------------------------------------------------------------------

// UART Interrupt Service Routine (bei Empfangenem Zeichen)
#pragma code
#pragma interrupt uart_isr
void uart_isr( void)
{
 char x;

 x = RCREG;
 if( uart_rx_count < 16) {
   uart_rx_buffer[uart_rx_count++] = x;
 }

 PIR1bits.RCIF = 0;
}

//-------------------------------------------------------------------------------

//LED-Ports Initialisieren (Ausgang, aus)
void init_led( void)
{
 TRISBbits.TRISB3 = 0; //Ausgang Gelbe LED
 TRISBbits.TRISB4 = 0; //Ausgang Gruene LED
 TRISBbits.TRISB5 = 0; //Ausgang Gelde LED
 PORTBbits.RB3 = 1;
 PORTBbits.RB4 = 1;
 PORTBbits.RB5 = 1;
}

//-------------------------------------------------------------------------------

// UART auf "BAUD" Geschwindigkeit, 8N1 Programmieren, Empfangsinterrupt zulassen
void init_uart(void)
{
 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 = 1;  //Empfangsinterrupt!
 PIE1bits.TXIE = 0; //KEIN Sendeinterrupt!
 INTCONbits.GIE = 1;
 INTCONbits.GIEL = 1;
}

//-------------------------------------------------------------------------------
// Interrupt Prioritaeten konfigurieren

void init_interrupts(void)
{
 RCONbits.IPEN = 1;   //Interrupt mit Prioritaeten
 IPR1bits.RCIP = 0;   //USART RX low priority
 IPR1bits.TXIP = 0;   //USART TX low prioroty
 INTCON2bits.TMR0IP = 1; //TMR0 high priority
 INTCONbits.GIEH = 1;
 INTCONbits.GIEL = 1;
}

//-------------------------------------------------------------------------------
```

```
// SPI  und /CS-Leitung(Ausgang) fuer MCP3551 konfigurieren

void init_spi(void)
{
 TRISBbits.TRISB2 = 0; //Ausgang /CS fuer A/D-Wandler Luftdruck
 PORTBbits.RB2 = 1;

 TRISCbits.TRISC0 = 0; //Ausgang /CS fuer A/D-Wandler Rotorrichtung
 PORTCbits.RC0 = 1;

 TRISCbits.TRISC7 = 1; //SDO 0 wenn auch sedendet werden soll - Achtung UART-PIin!
 TRISBbits.TRISB1 = 0; //SCK out (in Master Mode)

 TRISBbits.TRISB0 = 1;

 SSPSTATbits.CKE = 0; //SPI Mode 1,1
 SSPSTATbits.SMP = 0;
 SSPCON1bits.CKP = 1; //SPI Mode 1,1
 SSPCON1bits.SSPM0 = 0; //SPI Master Clocl = FOSC/4
 SSPCON1bits.SSPM1 = 0; //SPI Master Clocl = FOSC/4
 SSPCON1bits.SSPM2 = 0; //SPI Master Clocl = FOSC/4
 SSPCON1bits.SSPM3 = 0; //SPI Master Clocl = FOSC/4
 SSPCON1bits.SSPEN = 1; //Enable SPI
}

//-------------------------------------------------------------------------------
// MCP3552 A/D-Wandler lesen
// Kanal legt /CS-Leitung fest (zwei MCP3551 an SPI)
// Kanal 0 -> Luftdruck
// Kanal 1 -> Antennenrichtung
//
long read_spi_mcp3551( char kanal)
{
 unsigned char x,x1,x2,x3;
 long i;

 x = SSPBUF; //SPI-Buffer leer lesen

 if( kanal == 0) {
   PORTBbits.RB2 = 0; // /CS - Konvertierung A/D-Wandler 0 starten
   PORTBbits.RB2 = 1; // /CS
 }
 else {
   PORTCbits.RC0 = 0; // /CS - Konvertierung A/D-Wandler 1 starten
   PORTCbits.RC0 = 1; // /CS
 }

 do { //Mit /CS Status abfragen bis A/D-Wandler mit SDI=0 "/RDY" anzeigt
   if( kanal == 0)
     PORTBbits.RB2 = 1;
   else
     PORTCbits.RC0 = 1;
   Nop(); Nop(); Nop(); Nop(); Nop();
   if( kanal == 0)
     PORTBbits.RB2 = 0;
   else
     PORTCbits.RC0 = 0;
   Nop(); Nop(); Nop(); Nop(); Nop();
 } while( PORTBbits.RB0);

 while( PORTBbits.RB0);

 SSPBUF = 0;
 while( !SSPSTATbits.BF); //Warten bis erstes Byte empfangen
 x1 = SSPBUF; //Byte lesen
 SSPBUF = 0;
 while( !SSPSTATbits.BF); //Warten bis zweites Byte empfangen
 x2 = SSPBUF; //Byte lesen
 SSPBUF = 0;
 while( !SSPSTATbits.BF); //Warten bis drittes Byte empfangen
 x3 = SSPBUF; //Byte lesen

 if( kanal == 0)
   PORTBbits.RB2 = 1; // CS
 else
```

```
   PORTCbits.RC0 = 1; // CS
 if( x1 < 32)
   return((long)x1 << 16 | (long)x2 << 8 | (long)x3);
 else
   return(-1);
}

//---------------------------------------------------------------------------
//Timer0 als Timer mit 1 Interrupts / Sekunde aufsetzen
void init_timer0( void)
{
 T0CONbits.T0PS0 = 1; // :256
 T0CONbits.T0PS1 = 1;
 T0CONbits.T0PS2 = 1;
 T0CONbits.PSA = 0; //Prescaler
 T0CONbits.T0CS = 0; // Clock = FOSC/4
 T0CONbits.T08BIT = 0; //16bit

 TMR0H = 256 - ((long)TAKT / 4 / 256 / 256) - 1;
 TMR0L = 256 - (((long)TAKT / 4 / 256) % 256);

 INTCONbits.TMR0IF = 0;
 INTCONbits.TMR0IE = 1;
 T0CONbits.TMR0ON = 1; //Enable Timer
}


//---------------------------------------------------------------------------
void double2string( double wert_f, char string[], char vkstellen, char kstellen)
// Formatiert einen double in einen ASCII-String
// vk = Stellen vor dem Komma (mit fuehrenden Nullen)
// kstellen = Stellen nach dem Komma)
//
{
 char i;
 long wert;
 double multi = 1.0;

 i = kstellen;
 while( i--) multi *= 10.0;

 wert = (long)(wert_f * multi);
 for( i = kstellen ? vkstellen + kstellen : vkstellen - 1; i >= 0; i--) {
   string[i] = (wert % 10) + '0';
   wert /= 10;
   if( kstellen && (i == vkstellen + 1)) i--;
 }
 if( kstellen) string[vkstellen] = '.';
}

//---------------------------------------------------------------------------
void putsUART( char *string)
{
 char *zeichen;

 zeichen = string;
 while( *zeichen) {
   Nop(); Nop(); Nop();
   while(!PIR1bits.TXIF); //Warten bis TXREG leer
   TXREG = *zeichen++;
 }
}

//---------------------------------------------------------------------------
//ADC an AN1
int read_adc( void)
{
 int wert;

 TRISAbits.TRISA1 = 1;
 ADCON1 = 0b00001101; //AN0 und AN1, VREF- = VSS, VREF+ = VDD
 ADCON2 = 0b10100010; //TAD=FOSC/32, ACCTIME = 8*TAD
 ADCON0 = 0b00000100; //AN1
 ADCON0bits.ADON = 1; //ADC on
 ADCON0bits.GO = 1; //Wandlung starten
```

```
 Nop(); Nop(); Nop();
 while( ADCON0bits.GO); //Warten bis Wandlung beendet
 wert = ADRESL + 256 * ADRESH;
 ADCON0bits.ADON = 0; //ADC off
 return(wert);
}

//---------------------------------------------------------------------------
//Luftdruck von Sensor lesen und umrechnen;
double lese_druck( void)
{
 char i;
 long x,drucksumme = 0;
 static double druck;

 drucksumme = 0; //Mittel aus 11 Messwerten bilden
 for( i = 0; i < 11; i++) {
   x = read_spi_mcp3551(0);
   if( x != -1)
   drucksumme += x;
 }

 druck = (((double)drucksumme / 11.0 / 2097152.0) + 0.1518 ) * 944.2870633;
  if( druck > 880.0 && druck < 1051.0)
    PORTBbits.RB3 = 0; //Fehler in Druckmessung
  else {
    PORTBbits.RB3 = 1;
    druck = 0.0;
  }
  druck += druckkorrektur;
  return( druck);
}

//---------------------------------------------------------------------------
//A/D-Wert der Richtung lesen und in Richtung umrechnen;
long lese_richtung( void)
{
 char i;
 long x;
 double richtung;
 static long l_richtung;

 x = read_spi_mcp3551(1);
 if( x > 0x1fffff) x = 0; //Negativer Wert (Rauschen oder Ueberlauf)
 if( x < adcccw) x = adcccw;
 if( x > adccw) x = adccw;
 richtung = (double)(x - adcccw) / (double)(adccw - adcccw) * (double)winkel;
 if( richtung - floor( richtung) > 0.5) richtung += 1;

 l_richtung = (long)richtung;
 return( l_richtung);
}

//---------------------------------------------------------------------------
// Variable aus EEPROM lesen
// Immer 4 Bytes -> nur double oder long !!!
//
void* lese_eeprom( unsigned char speicherplatz)
{
 static char wert[4];
 char i;

 for( i = 0; i < 4; i++) {
   EEADR = speicherplatz + i;
   EECON1bits.EEPGD = 0; //Point to DATA memory
   EECON1bits.CFGS = 0 ; //Access EEPROM
   EECON1bits.RD = 1 ; //EEPROM Read
   wert[i] = EEDATA;
 }

 return( (void*)wert);
}

//---------------------------------------------------------------------------
// Variable EEPROM schreiben
```

```
// Immer 4 Bytes -> nur double oder long !!!
//
void schreibe_eeprom( void* variable, char speicherplatz)
{
 char i;

 for( i = 0; i < 4; i++) {
   EEADR = speicherplatz + i; //Adresse
   EEDATA = *(unsigned char*)(variable + i); //Wert
   EECON1bits.EEPGD = 0; //Point to DATA memory
   EECON1bits.CFGS = 0; //Access EEPROM
   EECON1bits.WREN = 1; //Enable writes
   INTCONbits.GIE = 0; //Disable Interrupts
   EECON2 = 0x55; //Required: Write 55h
   EECON2 = 0xAA; //Required: Write 0AAh
   EECON1bits.WR = 1; //Set WR bit to begin write
   INTCONbits.GIE = 1; //Enable Interrupts
   while( EECON1bits.WR); //Warten bis Schreiben beendet
   EECON1bits.WREN = 0; //Disable writes on write complete (EEIF set)
 }
}

//-------------------------------------------------------------------------------

void kalibrierung( void)
{
 static rom char text1[] = "\r\n\r\nKalibrierung\r\n";
 static rom char text2[] = "\r\nOrtsdruck (mit Korrektur) [hPa]: ";
 static rom char text3[] = "\r\nDruckkorrekturwert [hPa]: ";
 static rom char text4[] = "\r\nA/D-Wert [0..1023] Rotorinterface-Spannung : ";
 static rom char text5[] = "\r\nRichtung A/D-Wert momentan: ";
 static rom char text6[] = "\r\nA/D-Wert CCW-Anschlag: ";
 static rom char text7[] = "\r\nA/D-Wert CW-Anschlag: ";
 static rom char text8[] = "\r\nDrehbereich zwischen CCW- und CW-Anschlag in Grad: ";
 static rom char text9[] = "\r\nMomentane Richtung in Grad: ";
 static rom char text10[] = "\r\n\r\n+ Druckkorrektur +0.01 hPa";
 static rom char text11[] = "\r\n- Druckkorrektur -0.01 hPa";
 static rom char text12[] = "\r\nL Rotor ist jetzt in CCW-Anschlag";
 static rom char text13[] = "\r\nR Rotor ist jetzt in CW-Anschlag";
 static rom char text14[] = "\r\nP Drehbereich des Rotors vergroessern";
 static rom char text15[] = "\r\nM Drehbereich des Rotors verringern";
 static rom char text16[] = "\r\nQ Speichern und Kalibrierung beenden";
 static rom char text17[] = "\r\nX NICHT Speichern und Kalibrierung beenden";
 static rom char text18[] = "\r\n\r\nBitte geben Sie ein Zeichen ein [+-LRPMQX ]:\r\n";
 static rom char newline[] = "\r\n";
 static rom char quit[] = "\r\nWerte Abgespeichert\r\n\r\n";
 static rom char exit[] = "\r\nAbgebrochen\r\n\r\n";

 double t_druckkorrektur = druckkorrektur; //Werte zwischenspeichern falls EXIT!
 long t_adcccw = adcccw;
 long t_adccw = adccw;
 long t_winkel = winkel;
 char zeile[60];
 int i;
 long t_ad_richtung;

 uart_rx_count = 0;
 do {
   for( i = 0; i < 60; i++) zeile[i] = 0; //Begruessung
   strcpypgm2ram( zeile, text1);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text2);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0; //Luftdruck Anzeigen
   double2string( lese_druck(), zeile, 4, 2);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text3);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0; //Luftdruck-Korrektur Anzeigen
```

```
if( druckkorrektur < 0) {
  zeile[0] = '-';
  double2string( -druckkorrektur, zeile+1, 2, 2);
}  else {
  zeile[0] = '+';
  double2string( druckkorrektur, zeile+1, 2, 2);
}
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text4);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; //A/D-Wert der Rotorversorgung Anzeigen
double2string( (double)read_adc(), zeile, 4, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text5);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; //A/D-Wert der Richtung Anzeigen
t_ad_richtung = read_spi_mcp3551(1);
double2string( (double)t_ad_richtung, zeile, 7, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text6);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; // A/D-Wert CCW-Anschlag
double2string( (double)adcccw, zeile, 7, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text7);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; // A/D-Wert CW-Anschlag
double2string( (double)adccw, zeile, 7, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text8);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; // Drehbereich Rotor
double2string( (double)winkel, zeile, 3, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text9);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0; //Richtung Anzeigen
richtung = lese_richtung();
double2string( (double)richtung, zeile, 3, 0);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text10);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text11);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text12);
putsUART( zeile);

for( i = 0; i < 50; i++) zeile[i] = 0;
strcpypgm2ram( zeile, text13);
putsUART( zeile);
```

```
   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text14);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text15);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text16);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text17);
   putsUART( zeile);

   for( i = 0; i < 50; i++) zeile[i] = 0;
   strcpypgm2ram( zeile, text18);
   putsUART( zeile);

   //Zeichen lesen
   while( !uart_rx_count);
   uart_rx_count = 0;
   switch( uart_rx_buffer[0]) {
     case '+': druckkorrektur += 0.01;
               break;
     case '-': druckkorrektur -= 0.01;
               break;
     case 'l':
     case 'L': adcccw = t_ad_richtung;
               break;
     case 'r':
     case 'R': adccw = t_ad_richtung;
               break;
     case 'p':
     case 'P': if( winkel < 600) winkel++;
               break;
     case 'm':
     case 'M': if( winkel > 0) winkel--;
               break;
     case 'q':
     case 'Q': schreibe_eeprom( (void*)&druckkorrektur, EE_DRUCKKORREKTUR);
               schreibe_eeprom( (void*)&adcccw, EE_ADCCCW);
               schreibe_eeprom( (void*)&adccw, EE_ADCCW);
               schreibe_eeprom( (void*)&winkel, EE_WINKEL);
               for( i = 0; i < 50; i++) zeile[i] = 0;
               strcpypgm2ram( zeile, quit);
               putsUART( zeile);
               break;
     case 'x':
     case 'X': druckkorrektur = t_druckkorrektur; //Werte nicht aendern!
               adcccw = t_adcccw;
               adccw = t_adccw;
               winkel = t_winkel;
               for( i = 0; i < 50; i++) zeile[i] = 0;
               strcpypgm2ram( zeile, exit);
               putsUART( zeile);
               break;
     default: break;
   }
 } while( uart_rx_buffer[0] != 'q' && uart_rx_buffer[0] != 'Q' &&
          uart_rx_buffer[0] != 'x' && uart_rx_buffer[0] != 'X');
}

//-------------------------------------------------------------------------------

void main( void)
{
 int i;
 double druck;
 char ausgabestring[20];
 long adc = 0, adc1alt = 0;
 long richtung_alt = 0;
 char richtung_einmal_gelesen = 0; //Zeigt an ob Richtung nach Einschalten einmal
                                   //"live" gelesen
```

```
 init_interrupts();
 init_led();
 init_uart();
 init_spi();
 init_timer0();

 //Eeprom-Werte leesen und falls Eeprom zum ersten mal genutzt auf Startwerte
 //setzen
 winkel = *(long*)lese_eeprom( EE_WINKEL);
 if( winkel > 600 || winkel < 0) {
   druckkorrektur = 3.9;
   schreibe_eeprom( (void*)&druckkorrektur, EE_DRUCKKORREKTUR);
   adcccw = 0;
   schreibe_eeprom( (void*)&adcccw, EE_ADCCCW);
   adccw = 2097151L;
   schreibe_eeprom( (void*)&adccw, EE_ADCCW);
   winkel = 450L;
   schreibe_eeprom( (void*)&winkel, EE_WINKEL);
   richtung = 0L;
   schreibe_eeprom( (void*)&richtung, EE_RICHTUNG);
 }

 druckkorrektur = *(double*)lese_eeprom( EE_DRUCKKORREKTUR);
 adcccw = *(long*)lese_eeprom( EE_ADCCCW); // A/D-Wert bei Linksanschlag CCW = 0Grad!
 adccw = *(long*)lese_eeprom( EE_ADCCW); // A/D-Wert bei Rechtsanschlag CW = xGrad!
 winkel = *(long*)lese_eeprom( EE_WINKEL); //Moeglicher Drehwinkel des Rotors in Grad
(450)
 richtung = *(long*)lese_eeprom( EE_RICHTUNG); //Stellung des Rotors in Grad
 richtung_alt = richtung;

 for( i = 0; i < 30000; i++); //Einfach warten (A/D-Wandler Start)

 while( 1) {
  druck = lese_druck();
  double2string( druck, ausgabestring, 4, 2);
  ausgabestring[7] = ' ';

  //Einfache AD-Werte fuer Versorgungsspannung des Rotors an AN0
  adc = 0; //Versorgungsspannung als Einschaltkontrolle
  for( i = 0; i < 32; i++)
    adc += read_adc();
  adc /= 32;

  double2string( richtung_alt > 359 ? (double)(richtung_alt-360) :
                 (double)richtung_alt, ausgabestring+8, 3, 0);
  ausgabestring[11] = ' ';

  if( adc > 950) { //Rotor an -> Werte live
    ausgabestring[12] = '1';
    PORTBbits.RB5 = 0;  //Rotor ist an
    if( adc >= adc1alt - 50) { //Nur Speichern falls Versorgungsspannung konstant
      richtung_alt = richtung;
      richtung = lese_richtung();
      richtung_einmal_gelesen = 1;
    }
  } else { //Rotor ist aus
    ausgabestring[12] = '0';
    if( richtung_einmal_gelesen) {
      //Richtung nach Ausschalten des Rotors abspeichern nur falls geaendert!
      if( richtung_alt != *(long*)lese_eeprom( EE_RICHTUNG)) {
        schreibe_eeprom( (void*)&richtung_alt, EE_RICHTUNG);
      }
      PORTBbits.RB5 = 1;  //Rotor ist aus
    } else {
      PORTBbits.RB5 = ~PORTBbits.RB5; //Blinken
    }
  }
  adc1alt = adc; //Letzter Stromversorungswert;

  ausgabestring[13] = '\n';
  ausgabestring[14] = 0;

  while( !timer0); //Auf volle Sekunde warten
  timer0 = 0;
```

```
  //Kalibrierung durch "kk" angefordert?
  if( uart_rx_count == 2 && uart_rx_buffer[0] == 'k' && uart_rx_buffer[1] == 'k') {
    kalibrierung();
  }
  uart_rx_count = 0;

  //Werte senden
  putsUART( ausgabestring);
 }
}

//---------------------------------------------------------------------------------
```