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

  PeetBros-Anemometer
  (C)2006,2010 Hardy Lau

  Microchip dsPIC30F4011
  ICD2/ICD3
  Externer Oszillator mit 6,144 MHz und 4x PLL (Fcy = 6.144.000 Hz)
  "Natuerliche"-Interrupt-Prioritaeten; dadurch kein Interruptnesting!

  Anemometer gibt einen Puls pro Umdrehung (=1,07m Windweg) ab.
  Prellen ist moeglich und wird daher abgefangen.
  Peet Bros Anemometer gibt zusaetzlich einen verschobenen Puls pro Umdrehung
  fuer die Windrichtung ab.

  V1.0  6.10.2006 Initialversion
       13.10.2006 Capture,Timer,UART (UART mit 19200 Baud 8n1)
       23.10.2006 Fallende und Steigende Flanke auswerten
       24.10.2006 Fehlerauswertung verbessert,
                  Spitzengeschwindigkeit wird fuer 0,8 Meter Windweg ausgegeben
        1.04.2010  Umbau fuer PeetBros Anemometer PRO
        7.04.2010  Windrichtung wird durch Phasenverschiebung der Flanken registriert.
                   Nur fallende Flanken auswerten.
                   Gelb = Geschwindigkeit (1)
                   Gruen = Windrichtung (2). Fallende Flanke ist um Himmelsrichtung des
                   Windes verschoben. 0=Nord, 90=Ost, 180=Sued, 270=West
        9.04.2010  Test im Labor
       10.04.2010  Rote LED nur bei neu ermittelter Windrichtung
       11.04.2010  Bearbeitung der Fehlermeldungen, Prellzeit verlaengert da Thies-
                   Windmesser bei sehr langsamen drehen "rattert".
       12.04.2010  Windrichtung "0" wenn keine Windrichtung ermittelbar
       26.04.2010  Windrichtung bei fehlenden Windrichtungspulsen auch nach
                   Timerueberlauf auf 0 setzen
                   PLL auf 4fach reduziert. Dadurch weniger Stromverbrauch und
                   Erwaermung des Linearreglers.
 -------------------------------------------------------------------------------*/

#define FCY 6144000UL  //Geschwindigkeit der Zaehler
#define PI 3.141592654
#define WINDWEG 1.07    //PEET BROS.
//#define WINDWEG 0.8   //Thies

#include <p30f4011.h>
#include <math.h>

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

#define CBL 80 //Laenge des Capture-Buffers
volatile unsigned long cbr[CBL];  //Capture buffer (Richtung)
volatile unsigned long cb[CBL];  //Capture buffer (Geschwindigkeit)
volatile int cbz = CBL-1; //Zeiger auf letzten wert in cb
volatile int cbz_shadow = CBL-1;
volatile int cbzr = CBL-1; //Zeiger auf letzten wert in cbr
volatile int cbzr_shadow = CBL-1;
volatile int naechste_sekunde = 0;
volatile unsigned long zeit; //Aktuelle Zeit
volatile int ic1_unvergleichbar = 0; //Mehrfacher ueberlauf moeglich
volatile int ic7_unvergleichbar = 0; //Mehrfacher ueberlauf moeglich

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

/*
 * Gruene LED an RD1, Gelbe LED an RD3
 */
void init_led()
{
 TRISDbits.TRISD1 = 0;  //RD1 Ausgang
 PORTDbits.RD1 = 0;     //RD1 aus
 TRISDbits.TRISD2 = 0;  //RD2 Ausgang
 PORTDbits.RD2 = 0;     //RD2 aus
 TRISDbits.TRISD3 = 0;  //RD3 Ausgang
 PORTDbits.RD3 = 0;     //RD3 aus
 }

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

/*
 * Gelbe LED einschalten
```

```
 */
inline void gelbe_led_ein()
{
 PORTDbits.RD3 = 1;
}

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

/*
 * Gelbe LED ausschalten
 */
inline void gelbe_led_aus()
{
 PORTDbits.RD3 = 0;
}

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

/*
 * Gruene LED einschalten
 */
inline void gruene_led_ein()
{
 PORTDbits.RD1 = 1;
}

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

/*
 * Gruene LED ausschalten
 */
inline void gruene_led_aus()
{
 PORTDbits.RD1 = 0;
}

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

/*
 * Rote LED einschalten
 */
inline void rote_led_ein()
{
 PORTDbits.RD2 = 1;
}

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

/*
 * Rote LED ausschalten
 */
inline void rote_led_aus()
{
 PORTDbits.RD2 = 0;
}


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

void _ISR __attribute__((no_auto_psv)) _OscillatorFail( void)
{
}

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


/*
 * TIMER1 ISR
 */
void _ISRFAST __attribute__((no_auto_psv)) _T1Interrupt( void)
{
 static int i = 0;

 if( i++ == 1) { //Jede volle Sekunde
   zeit = (unsigned long)TMR2;
   zeit |= ((unsigned long)TMR3HLD << 16);  //Verwendete Zeit im Hauptprogramm
   naechste_sekunde = 1;   //Semaphore eZeigt Hauptprogramm den Ablauf einer Sekunde an
   cbz_shadow = cbz;       //Aktuelle Zeiger im Capturespeicher fuer HP
```

```
   cbzr_shadow = cbzr;     //Aktuelle Zeiger im Capturespeicher fuer HP
   gruene_led_ein();
 } else {
   gruene_led_aus();
   rote_led_aus();
   gelbe_led_aus();
 }
 if( i > 1) i = 0;

 // Clear Interrupt flag
 IFS0bits.T1IF = 0;
}

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

/*
 * CAPTURE ISR1 (Geschwindigkeit)
 */
void _ISR __attribute__((no_auto_psv)) _IC1Interrupt( void)
{
 static unsigned long last_zeit;

 unsigned long cb_zeit; //Zeit (32bit) zum "input capture" Ereigniss
 unsigned int lsb_cb_zeit;
 unsigned int lsb_zeit, msb_zeit;
 unsigned int tmp;

 //Aktuelle Zeit festellen
 lsb_zeit = TMR2;
 msb_zeit = TMR3HLD;

 //Capture-Wert lesen
 lsb_cb_zeit = IC1BUF; //Wert der ge-captured wurde
 if( lsb_cb_zeit > lsb_zeit) { //LSB-Zaehler zwischen ISR und capture uebergelaufen
   msb_zeit -= (unsigned long)1;
 }
 cb_zeit = (unsigned long)lsb_cb_zeit | ( (unsigned long)msb_zeit << 16);

 //Falls wert nicht zu gering von altem Wert unterscheidet (=Prellen) abspeichern
 if( ic1_unvergleichbar || (cb_zeit - last_zeit) > (unsigned long)(FCY / 100UL ) ) {
   if( ++cbz > CBL-1) cbz = 0;
   cb[cbz] = cb_zeit;
   last_zeit = cb_zeit; //Zeit festhalten (wegen Prellen)
 }

 ic1_unvergleichbar = 0;

 //Capture buffer leeren (nur bei Fehlerfall)
 while( IC1CONbits.ICBNE)
   tmp = IC1BUF;

 //Clear Interrupt flag
 IFS0bits.IC1IF = 0;
}

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

/*
 * CAPTURE ISR7 (Richtung)
 */
void _ISR __attribute__((no_auto_psv)) _IC7Interrupt( void)
{
 static unsigned long last_zeit;

 unsigned long cb_zeit; //Zeit Jetzt
 unsigned int lsb_cb_zeit;
 unsigned int lsb_zeit, msb_zeit;
 unsigned int tmp;

 //Aktuelle Zeit festellen
 lsb_zeit = TMR2;
 msb_zeit = TMR3HLD;

 //Capture-Wert lesen
 lsb_cb_zeit = IC7BUF; //Wert der ge-captured wurde
 if( lsb_cb_zeit > lsb_zeit) { //LSB-Zaehler zwischen ISR und capture uebergelaufen
   msb_zeit -= (unsigned long)1;
 }
```

```
 cb_zeit = (unsigned long)lsb_cb_zeit | ( (unsigned long)msb_zeit << 16);

 //Falls wert nicht zu gering von altem Wert unterscheidet (=Prellen) abspeichern
 if( ic7_unvergleichbar || (cb_zeit - last_zeit) > (unsigned long)(FCY / 100UL ) ) {
   if( ++cbzr > CBL-1) cbzr = 0;
   cbr[cbzr] = cb_zeit;
   last_zeit = cb_zeit; //Zeit festhalten (wegen Prellen)
}

 ic7_unvergleichbar = 0;

 //Capture buffer leeren (nur bei Fehlerfall)
 while( IC7CONbits.ICBNE)
   tmp = IC7BUF;

 //Clear Interrupt flag
 IFS1bits.IC7IF = 0;
}

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

/*
 * Serielle Schnittstelle initialisieren
 */
void init_uart()
{
 #define BAUD 19200
 U1BRG = ( FCY / 16 / BAUD) -1;
 U1MODEbits.ALTIO = 1;  // Benutze U1ARX,U1ATX Pins
 U1MODEbits.PDSEL = 0;  // 8 Bit, no parity
 U1MODEbits.STSEL = 0;  // 1 Stop-Bis
 U1MODEbits.UARTEN = 1; // Enable;
 U1STAbits.UTXEN = 1; // Enable
 U1STAbits.UTXISEL = 0; //Interrupt bei einem freien TXBUFFER
 U1STAbits.URXISEL = 0; //Interrupt bei jedem empfangenen Zeichen
 IFS0bits.U1TXIF = 0;
 IFS0bits.U1RXIF = 0;
 IEC0bits.U1TXIE = 0; //Disable UART TX-Interrupt
 IEC0bits.U1RXIE = 0; //Disable UART RX-Interrupt
}

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

/*
 * CAPTURE initialieren
 */
void init_capture(void)
{
 int i;

 //IC1 als Eingang
 TRISDbits.TRISD0 = 1; //RD0/IC1 ist Eingang
 PORTDbits.RD0 = 0;

 //Analogports abschalten
 ADPCFG = 0xffff;
 //IC7 als Eingang
 TRISBbits.TRISB4 = 1; //RB4/IC7 ist Eingang

 //Capture Buffer leeren
 for( i = 0; i < CBL; i++) {
   cb[i] = 0;
   cbr[i] = 0;
 }
 cbz = CBL-1;
 cbzr = CBL-1;

 //Timer2 und Timer3 initialisieren als ein 32Bit-Timer
 TMR2 = 0;
 TMR3 = 0;
 T2CONbits.TCKPS = 0; //Kein Vorteiler
 PR2 = 0xffff;
 PR3 = 0xffff;
 T2CONbits.T32 = 1; //32Bit-Timer T2:T3
 T2CONbits.TON = 1; //Timer2 starten

 //Capture IC1 mit Timer2
 IC1CONbits.ICTMR = 1; //Timer2 wird captured
```

```
 IC1CONbits.ICI = 0; //Interrupt bei jedem Ereigniss
 IC1CONbits.ICM = 2; //Jede fallende Flanke (2 = Jede fallende Flanke)
 IFS0bits.IC1IF = 0;
 IEC0bits.IC1IE = 1; //Enable IC1-Interrupt

 //Capture IC7 mit Timer2
 IC7CONbits.ICTMR = 1; //Timer2 wird captured
 IC7CONbits.ICI = 0; //Interrupt bei jedem Ereigniss
 IC7CONbits.ICM = 2; //Jede fallende Flanke (2 = Jede fallende Flanke)
 IFS1bits.IC7IF = 0;
 IEC1bits.IC7IE = 1; //Enable IC7-Interrupt
}

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

/*
 * Timer1 so setzen, dass alle 0.5 Sekunden ein Interrupt ausgeloest wird
 * Jedes zweite mal wird dann die Ausgabe aufgerufen
 */
void init_sekundentimer( void)
{
 TMR1 = 0; //Timer Register leeren
 PR1 = FCY / (256 * 2); //Bei 6,14 MHz * 4 ergibt ca 12000
 T1CONbits.TCS = 0;
 T1CONbits.TGATE = 0;
 T1CONbits.TSYNC = 0;
 T1CONbits.TCKPS = 3; //Teiler 256
 IFS0bits.T1IF = 0;
 IEC0bits.T1IE = 1; //Timer1 Interrupt erlauben
 T1CONbits.TON = 1; //Timer on;
}

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

/*
 * liicb (Last index in cycle buffer)
 *
 * Zugriff auf Element in Ringspeicher
 */
inline int liccb( int index)
{
 if( index < 0) return( CBL + index);
 if( index >= CBL) return( index - CBL);
 return( index);
}

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

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

 //Round simulieren
 wert = (long)(wert_f * 100.0) % 10;
 if( wert > 4) wert_f += 0.05;

 wert = (long)(wert_f * 10.0);
 for( i = 4; i >= 0; i--) {
   string[i] = (wert % 10) + '0';
   wert /= 10;
   if( i == 4) i--;
 }
 string[3] = '.';
}

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

/*
 * Formatiert einen unsigned long in einen ASCII-String
 */
void ulong2string( unsigned long wert, char string[])
{
 int i;
```

```
 for( i = 7; i >= 0; i--) {
   string[i] = wert & 0x0f;
   if( string[i] < 10) string[i] += '0';
   else string[i] += 'A' - 10;
   wert >>= 4;
 }
}

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

/*
 * Formatiert einen int in einen ASCII-String
 */
inline void int2string( int wert, char buffer[])
{
 int i;

 for( i = 2; i >= 0; i--) {
   buffer[i] = (wert % 10) + '0';
   wert /= 10;
 }
 buffer[3] = 0;
 //Fuehrnede Nullen entfernen
 if( buffer[0] == '0') {
   buffer[0] = ' ';
   if( buffer[1] == '0') {
     buffer[1] = ' ';
   }
 }
}

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

/*
 * Arcustangenz
 * Gibt Winkel in Grad (0...45) zurueck
 * Gueltig nur fuer +0..+1
 */
inline int atan_i( float tangenz)
{
 unsigned int tan;
 int i;

 //return( (int)(180.0 * atan( tangenz) / PI));

 tan = (unsigned int)(tangenz * 65535);

 const unsigned int tan_liste[] = {
   572, 1716, 2861, 4008, 5158,
   6310, 7467, 8628, 9794, 10967,
   12146, 13333, 14529, 15734, 16949,
   18174, 19412, 20663, 21928, 23207,
   24503, 25815, 27145, 28495, 29866,
   31259, 32675, 34115, 35583, 37078,
   38603, 40160, 41750, 43377, 45041,
   46746, 48493, 50287, 52129, 54023,
   55972, 57980, 60052, 62190, 64401
 };

 for( i = 0; i < 45; i++) {
   if( tan < tan_liste[i]) return( i);
 }

 return( 45);
}

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

/*
 * x_y_to_winkel( x, y)
 *
 * Berechnet Winkel (in Grad) aus X- und Y- Windkomponenten
 *
 * Windvektor        0
 *                   X
 *                   ^
 *                   |
 *                   |
```

```
 *            270 ---+----> Y 90
 *                   |
 *                   |
 *
 *                  180
 *
 */
inline int x_y_to_winkel( float vx, float vy)
{
 int w;

  if( vx == 0.0) {
    if( vy > 0.0) return( 270);
    else return( 90);
  }
  if( vy == 0.0) {
    if( vx > 0.0) return( 180);
    else return( 0);
  }
  if( vy > 0.0 && vx > 0.0) {
    if( vy >= vx) {
      return( 270 - atan_i( vx / vy));
    } else {
      return( 180 + atan_i( vy / vx));
    }
  }
  if( vy < 0.0 && vx > 0.0) {
    if( -vy >= vx) {
      return( 90 + atan_i( vx / -vy));
    } else {
      return( 180 - atan_i( -vy / vx));
    }
  }
  if( vy > 0.0 && vx < 0.0) {
    if( vy >= -vx) {
      return( 270 + atan_i( -vx / vy));
    } else {
      w = 360 - atan_i( vy / -vx);
      return( w == 360 ? 0 : w);
    }
  }
  if( vy < 0.0 && vx < 0.0) {
     if( -vy >= -vx) {
       return( 90 - atan_i( -vx / -vy));
     } else {
       return( atan_i( -vy / -vx));
     }
  }
  return( 0); // Fehler!
}

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

/*
 * Hauptprogramm
 */
main()
{
 static int i,j;
 static int cbz_alt = CBL-1;  //Zeiger fuer "abgearbeitete" Werte aus Speicher
 static int cbzr_alt = CBL-1; //Zeiger fuer "abgearbeitete" Werte aus Speicher
 static int cb_events = 0;    //Anzahl neuer zu bearbeitender Werte
 static int cbr_events = 0;   //Anzahl neuer zu bearbeitender Werte
 static unsigned long pulse = 0; //Pulslaenge ( => Windgeschwindigkeit in Km/h)
 static int sekunden_ohne_event = 0;
 static int sekunden_ohne_event_r = 0;
 static char buffer[50];      //Textbuffer fuer serielle Ausgabe
 static char error = 0;       //Anzeige fuer Fehler in Berechnung !!!

 unsigned long richtung_pulse_summe;
 unsigned long richtung_summe;
 static int richtung;
 static int neue_richtung;    //Flag das neue Windrichtung ermittelt worden ist
 float sin_richtung_summe, cos_richtung_summe;

 init_led();
 init_uart();
 init_capture();
```

```
 init_sekundentimer();

 while(1) {
   while(!naechste_sekunde); //Auf Ablauf einer Sekunde warten
   naechste_sekunde = 0;

   error = 0;

   //Anzahl der neuen capture-werte (Geschwindigkeit)
   cb_events = cbz_shadow >= cbz_alt ? cbz_shadow - cbz_alt : cbz_shadow - cbz_alt + CBL;
   cbz_alt = cbz_shadow;

   //Anzahl der neuen capture-werte (Richtung)
   cbr_events = cbzr_shadow >= cbzr_alt ? cbzr_shadow - cbzr_alt : cbzr_shadow - cbzr_alt
+ CBL;
   cbzr_alt = cbzr_shadow;

   //Nicht mehr als 50 werte in Capture Buffer (das waeren 50*1,07 m/s = 192,6 Km/h !!!)
   if( cb_events > 50 || cbr_events > 50) { //Fehler !!!
     pulse = 0;
     error = 1;
   }

   //Geschwindigkeit errechnen
   switch( cb_events ) {
     case 0: //Kein Capture-Event in letzter Sekunde
       if( ( zeit - cb[liccb(cbz_shadow)]) > (unsigned long)(4UL*FCY) ||
sekunden_ohne_event > 4) {
         pulse = 0;  //Windmesser steht schon laenger
       } else {
         if( cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-1)] < zeit -
cb[liccb(cbz_shadow)]) {
           pulse = zeit - cb[liccb(cbz_shadow)];  //Geschwindigkeit wird gerade geringer
         } else {
           pulse = cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-1)];
//Geschwindigkeitsaenderung unbestimmt
         }
         if( pulse < (unsigned long)(FCY) ) {
           pulse = 0;
           error = 2;
         }
       }
       break;
     case 1: //Ein Event in letztter Sekunde
       gelbe_led_ein();
       if( ( cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-1)]) > (unsigned long)(4UL*FCY)
|| sekunden_ohne_event > 4) {
         pulse = (unsigned long)(3UL * FCY / 2UL);  //Ein Puls nach langer Zeit ...
Geschwindigkeit unbekannt (Anlaufwert)
       } else { //Letzter Puls nicht laenger als 4 Sekunden
         if( cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-1)] < zeit -
cb[liccb(cbz_shadow)]) {
           pulse = ( zeit - cb[liccb(cbz_shadow-1)]) / 2UL;  //Geschwindigkeit wird
gerade geringer
         } else {
           pulse = cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-1)];
//Geschwindigkeitsaenderung unbestimmt
         }
         if( pulse < (unsigned long)(FCY/4UL) || pulse > (unsigned long)(5UL * FCY)) {
           pulse = (unsigned long)(3UL * FCY / 2UL);
           error = 3;
         }
       }
       break;
     default: //Zwei events oder mehr in letzter Sekunde
       gelbe_led_ein();
       if( sekunden_ohne_event < 4) {
         pulse = ( cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-cb_events)]) / cb_events;
       } else { //Zu alten ersten Wert weglassen
         pulse = ( cb[liccb(cbz_shadow)] - cb[liccb(cbz_shadow-cb_events+1)]) /
(cb_events-1);
       }
       if( pulse < FCY / (cb_events+2) ) { //Da stimmt was mit Anzahl
Events/Geschwindigkeit nicht
         pulse = FCY / cb_events;
         error = 4;
       }
       break;
```

```
   }

   //Windrichtung errechnen (falls moeglich)
   switch( cb_events) {
     case 0: //Kein Capture-Events seit 1 Sekunde
       if( sekunden_ohne_event > 2) {
         richtung = 0; //Keine Windrichtung!
       } else { //Nur zwei Sekunden
         //Windrichtung beibehalten!
       }
       break;
     case 1:  //Ein oder mehrere Capture-Events in letzter Sekunde
     case 2:
     default:
       sin_richtung_summe = 0;
       cos_richtung_summe = 0;
       neue_richtung = 0;
       if( (cb_events > 1 || sekunden_ohne_event < 2) && sekunden_ohne_event_r < 3) {
         for( i = 0; i < cb_events; i++) {
           //Passenden Wert fuer Richtungsimpuls im Intervall suchen
           for( j = 0; j < cbr_events+1; j++) {
             if( ( cbr[ liccb(cbzr_shadow-j)] - cb[liccb(cbz_shadow-i-1)] ) < (unsigned
long)(5UL*FCY) &&
                 ( cb[liccb(cbz_shadow-i)] - cbr[ liccb(cbzr_shadow-j)]  ) < (unsigned
long)(5UL*FCY) ) {
               //Passenden Richtungsimpuls gefunden
               richtung_summe = ( cbr[ liccb(cbzr_shadow-j)] - cb[liccb(cbz_shadow-i-1)]
);
               richtung_pulse_summe = ( cb[liccb(cbz_shadow-i)] -
cb[liccb(cbz_shadow-i-1)] );
               if( richtung_pulse_summe) {
                 sin_richtung_summe += sin( 2.0 * PI * (float)(richtung_summe) /
(float)(richtung_pulse_summe));
                 cos_richtung_summe += cos( 2.0 * PI * (float)(richtung_summe) /
(float)(richtung_pulse_summe));
                 neue_richtung++;
               }
               break;
             }
           }
         }
         if( neue_richtung) { //Neue Richtung ermittelt
           richtung = x_y_to_winkel( -cos_richtung_summe, -sin_richtung_summe);
           richtung += 1;
           rote_led_ein();
         } else {
           richtung = 0; //Keine Richtung ermittelbar!
         }
     } else {
       richtung = 0; //Richtung
     }
   }

   //Anzahl Sekunden ohne Event dient zur ueberpruefung der Capturewerte auf
Zeitueberlauf
   if( cb_events == 0) {
     if(sekunden_ohne_event < 30) sekunden_ohne_event++;
   } else {
     sekunden_ohne_event = 0;
   }

   if( cbr_events == 0) {
     if(sekunden_ohne_event_r < 30) sekunden_ohne_event_r++;
   } else {
     sekunden_ohne_event_r = 0;
   }

   //Der Timer ist nicht vergleichbar da moeglicherweise mehrfach uebergelaufen
   if( sekunden_ohne_event >= 20) {
     ic1_unvergleichbar = 1;
   }
   if( sekunden_ohne_event_r >= 20) {
     ic7_unvergleichbar = 1;
   }

   //Geschwindigkeitsmittel (ueber 1 Sekunde) seriell ausgeben
   if( pulse > 0) double2string( 3.6 * WINDWEG * (double)FCY / (double)pulse, buffer);
   else double2string( 0.0, buffer);
```

```
buffer[5] = ' ';
for( j = 0; j < 6; j++) {
  while( !U1STAbits.TRMT);
  U1TXREG = buffer[j];
  Nop();
}

//Wegen Historie einfach 0.0 in Spalte ausgeben
double2string( 0.0, buffer);
buffer[5] = ' ';
for( j = 0; j < 6; j++) {
  while( !U1STAbits.TRMT);
  U1TXREG = buffer[j];
  Nop();
}

//Windrichtung ausgeben
int2string( richtung, buffer);
buffer[3] = ' ';
for( j = 0; j < 4; j++) {
  while( !U1STAbits.TRMT);
  U1TXREG = buffer[j];
  Nop();
}

//Debugwerte
while( !U1STAbits.TRMT);
U1TXREG = '0' + error;
Nop();
while( !U1STAbits.TRMT);
U1TXREG = ' ';
Nop();

while( !U1STAbits.TRMT);
U1TXREG = '0' + (cb_events / 10);
Nop();
while( !U1STAbits.TRMT);
U1TXREG = '0' + (cb_events % 10);
Nop();
while( !U1STAbits.TRMT);
U1TXREG = ' ';
Nop();

while( !U1STAbits.TRMT);
U1TXREG = '0' + (cbr_events / 10);
Nop();
while( !U1STAbits.TRMT);
U1TXREG = '0' + (cbr_events % 10);
Nop();
while( !U1STAbits.TRMT);
U1TXREG = ' ';
Nop();

if( error) { //Zusaetzliche Zaehlerausgabe falls Fehler aufgetreten
  ulong2string( zeit, buffer);
  buffer[8] = ' ';
  for( j = 0; j < 9; j++) {
    while( !U1STAbits.TRMT);
    U1TXREG = buffer[j];
    Nop();
  }
  for( i = 0; i < cb_events; i++) {
     ulong2string( cb[liccb(cbz_shadow-i)], buffer);
    buffer[8] = ' ';
    for( j = 0; j < 9; j++) {
      while( !U1STAbits.TRMT);
      U1TXREG = buffer[j];
      Nop();
    }
  }
  for( i = 0; i < cbr_events; i++) {
     ulong2string( cbr[liccb(cbzr_shadow-i)], buffer);
    buffer[8] = ' ';
    for( j = 0; j < 9; j++) {
      while( !U1STAbits.TRMT);
      U1TXREG = buffer[j];
      Nop();
    }
```

```
      }
    } //If Error

    while( !U1STAbits.TRMT);
    U1TXREG = '\r';
    Nop();

    while( !U1STAbits.TRMT);
    U1TXREG = '\n';
    Nop();
  } //While(1)
}

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