```
/*
Prozessor: ATMEGA 168 e-Fuse 01h; h-Fuse DFh; l-Fuse E7h; lock 3Fh. externer Quarz
Version 0.0 1/20 pkr
Version mit RFM63
Version 0.2:
Alle 48s werden die Daten vom Temperatursensor (usw.) übertragen. Jedes zweite Datenpaket wird doppelt übertragen im Abstand von 31ms. Diese beiden Datenpakete werden ausgelesen und auf Gleichheit überprüft. Wenn sie gleich sind, dann werden die Daten übernommen.
WDT 411, 643, 756
391, 414, 655, 701 Interrupt sperren 6.4.*/

#define F_CPU 16000000UL

#include <stdlib.h>
#include <avr/wdt.h>
#include <avr/io.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <util/delay.h>
#include "Bibliothek/SPI1.c"
#include "Bibliothek/i2c.h"
#include "Bibliothek/uhr.h"
#include "Bibliothek/ws3080_sensor_0.2.h"
#include "Bibliothek/BME_Berechnung.c"
#include "Bibliothek/uart1.c"




//LEDs
#define LED_Empfang 0   //PC0
#define LED_Fehler 1    //PB1
#define LED_Batterie 2  //PD2

#define Daten_Eingang 0 //Dateneingang Empfang (PB0)
#define Datum_ein (PINB & (1<<Daten_Eingang))

#define LED_Empfang_an PORTC |= (1<<LED_Empfang)
#define LED_Empfang_aus PORTC &= ~(1<<LED_Empfang)
#define LED_Fehler_an PORTB |= (1<<LED_Fehler)
#define LED_Fehler_aus PORTB &= ~(1<<LED_Fehler)
#define LED_Batterie_an PORTD |= (1<<LED_Batterie)
#define LED_Batterie_aus PORTD &= ~(1<<LED_Batterie)

#define Anzahl_Byte_DS 27   //Anzahl der Byts pro Datensatz im EEPROM
#define Adresse_EEPROM 0xA2 //Schreiben 1. Block; 2.Block: 0xA2+8; zum Lesen 1 addieren
#define DS_Anzahl_Block (uint16_t) ((65536/Anzahl_Byte_DS)-1)   //Anzahl der Datensätze -1 pro Block im EEPROM

#define Timer1_aus TCCR1B &= ~(1<<CS12)
#define Timer2_an TCCR2B |= ((1<<CS22) | (1<<CS21) | (1<<CS20))    //Vorteiler = 1024
#define Timer2_aus TCCR2B &= ~((1<<CS22) | (1<<CS21) | (1<<CS20))


//void Lesen_letzten_DS_EEPROM(uint8_t Datensatz[Anzahl_Byte_DS]);

//Globale Variable im SRAM
volatile uint8_t Daten_roh[15];  //Empfangsdaten vom Sensor
uint8_t Fehler = 0;    //Fehlerausgabe im Teil ws3080_sensor
char Platz[] = {"1234567"};
uint8_t fehl = 1;   //Fehlerausgabe beim Dateneinlesen der Rohdaten
uint8_t k, n;
volatile uint8_t Anzahl_Flanken;    //wird zum Empfang der Kopfdaten benutzt (8*0xFF)
volatile uint8_t Paket_Zaehler = 0;
volatile uint8_t wechsel=0; //Variable für den Zellenwechsel beim EEPROM
uint8_t Datum_Zeit_k[] = {29, 2, 20, 9, 10, 5}; //T,M,J,h,m,s als Zahl
volatile uint16_t DS_Nr = 0;
volatile uint8_t Speicherintervall = 10;    //in Minuten
volatile uint8_t Stop = 0;
uint8_t Datensatz_eingelesen; //Merker, um Doppelspeicherung zu vermeiden
uint8_t Anforderung; //Anforderung von der seriellen Schnittstelle
uint8_t test[3];
uint8_t Fehl[3];
```

```
 73
 74  struct Datenausgabe{
 75   int16_t Temp1;    //aussen WS3080
 76   int16_t Temp2;    //aussen extra
 77   int16_t Temp3;    //innen
 78   uint8_t Feuchte1; //Aussen
 79   uint8_t Feuchte2; //Aussen extra
 80   uint8_t Feuchte3; //Innen
 81   uint16_t Helligkeit;
 82   uint8_t UV_I;
 83   uint16_t Druck;
 84   uint8_t Regen;
 85   uint16_t Wind_mittel;
 86   uint8_t Wind_Richtung;
 87   uint16_t Wind_Boen;
 88   uint8_t Batterie; // Vermutung obere Nibble Byte 7 WS3080
 89   uint8_t Fragezeichen; //Byte 10 vom WS3080 immer 0x55?
 90 };
 91
 92
 93 struct Datenausgabe Daten_modifiziert;
 94
 95
 96 union Ausgabe
 97 {
 98   struct Datenausgabe Daten_mod;
 99   uint8_t Daten_ausgabe[sizeof (struct Datenausgabe)];
100 } DS_Ausgabe;
101
102
103 //Speicher im EEPROM
104 uint8_t ee_Name_Version[] EEMEM = {"WS3080 Auswertung Version 0.2 pkr 1/20"};
105 uint16_t ee_Regenmenge_alt EEMEM = 0;   //Speicher für alte Regenmenge in 0,3mm/Digit
106 uint8_t ee_Speicherintervall EEMEM = 10;    //Speicherintervall in Minuten
107 uint16_t ee_Adresse[15] EEMEM = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};  //Datensatzspeicher
108 uint8_t ee_wechsel = 0; //0 ... 14 für Zelle des Feldes ee_Adresse[]; damit die Zelle jedes Jahr
    gewechselt wird.
109
110 const uint16_t Frequenz[] = {0x067D, 0x0764, 0x0816};   //868,4MHz
111
112 const uint16_t RFM63_Config_Tabelle[] =
113     {
114        0x0148,   //OOK Modulation, Spitzenwert erfassen
115        0x043F,  //Schwelle *0,5dB, 0x60 ... 0x38 muss durch probieren festgelegt werden (Störungen,
        Rauschen)
116        0x0D40,
117        0x0E04,
118        0x0F20,
119        0x1004,  //Bandbreite 157kHz Versatz: 100kHz
120        0x1258  //Synch Ausgang0x1B3C,  //kein CLK aus
121     };
122
123 const uint16_t RFM63_Rx = 0x0070;
124
125
126 void RFM63_Config()
127 {
128     uint8_t n;
129
130     for(n=0; n<3; n++)
131     {
132         Kommando_senden(Frequenz[n]);   //Frequenz einstellen
133     }
134
135     for(n=0; n<7; n++)
136     {
137         Kommando_senden(RFM63_Config_Tabelle[n]);   //Konfiguration RFM66
138     }
139
140     Kommando_senden(RFM63_Rx);   //Empfang konfigurieren
141 }
142
143
144
145 uint8_t Mittelwertbildung(volatile uint8_t Daten_roh[], uint8_t Ende)
```

```
146 {
147 //formt Rohdaten vom Sensor in lesbare Werte und bildet Maximum Wind_Boen bzw.
148 //Mittelwerte bei den Temperaturen und der Feuchte zwischen den Abrufen.
149 //Der Rückgabewert zeigt den fehlerhaften Sensor an. Bei 0 kein Fehler.
150 //Eine 1 in Ende stoppt die Datensammlung, und das Programm wird verlassen
151 //mit der Fehlermeldung.
152
153    uint16_t Wind_Boen;
154    static int32_t Temp_1_Summe = 0;
155    static uint8_t Anzahl_1 = 0;
156    static int32_t Temp_2_Summe = 0;
157    static uint8_t Anzahl_2 = 0;
158    static int32_t Temp_3_Summe = 0;
159    static uint8_t Anzahl_3 = 0;
160    static uint16_t Feuchte_1_Summe = 0;
161    static uint16_t Feuchte_2_Summe = 0;
162    static uint16_t Feuchte_3_Summe = 0;
163    static uint32_t Helligkeit_Summe = 0;
164    static uint8_t Anzahl_Helligkeit = 0;
165    static uint8_t UV_I_Summe = 0;
166    static uint16_t  Wind_Boen_alt = 0;
167    static uint16_t Regenmenge = 0;
168    uint16_t Regenmenge_neu = 0;
169    uint16_t Regenmenge_alt;
170   // static uint8_t letzter_DS[Anzahl_Byte_DS];
171
172    uint8_t Fehler = 0;
173
174
175 //Druck ermitteln
176     Daten_modifiziert.Druck = (uint16_t)(Druck_kompensiert_32Bit(Druck_roh())/10); //vom BME Sensor
177
178 //Temperatur 1
179     Temp_1_Summe += (int16_t)(((((int16_t)Daten_roh[0] & 0x0F) << 8) + (int16_t)Daten_roh[1])) - 400;
180 //Feuchte 1 ermitteln
181     Feuchte_1_Summe += Daten_roh[2];
182     Anzahl_1++;
183
184
185 //Regen ermitteln. Es wird nur die Regenmenge vom Mittelwertzeitraum bestimmt in 0,3mm Schritten
186     Regenmenge_neu = (((uint16_t)Daten_roh[5] << 8) + (uint16_t)Daten_roh[6]);
187     Regenmenge_alt = eeprom_read_word(&ee_Regenmenge_alt);
188     if ( Regenmenge_neu > Regenmenge_alt)
189     {
190         if ((Regenmenge_alt < 1) || (Regenmenge_neu > Regenmenge_alt + 10)) Regenmenge = 0; //EEPROM
        leer oder Fehler in 1min > 3mm; ev. Regenmenge_neu<Regenmenge_alt
191         else Regenmenge += Regenmenge_neu - Regenmenge_alt;
192         //Regenmenge_alt = Regenmenge_neu;
193         eeprom_write_word(&ee_Regenmenge_alt, Regenmenge_neu);
194     }
195
196     if ( Regenmenge_neu < Regenmenge_alt) eeprom_write_word(&ee_Regenmenge_alt, Regenmenge_neu);
    //Fehler im EEPROM
197
198     //Windwerte ermitteln
199     Daten_modifiziert.Wind_mittel = ((uint16_t)Daten_roh[3]*10 / 3); //in 0,1 m/s
200
201     Wind_Boen = ((uint16_t)Daten_roh[4]*10 / 3);   //in 0,1m/s
202     if ((Wind_Boen == 0) & (Daten_modifiziert.Wind_mittel == 0))    //bei Windstille wird sonst der
    letzte max_wert genommen.
203     {
204         Daten_modifiziert.Wind_Boen = 0;
205                 Wind_Boen_alt = 0;
206     }
207
208     if (Wind_Boen > Wind_Boen_alt)  //Maximalwert ermitteln: Böen
209     {
210         if (!(Wind_Boen > 4*Daten_modifiziert.Wind_mittel)) // bei Böen > 4*mittlere
        Windgeschwindigkeit Wert verwerfen
211         {
212             Daten_modifiziert.Wind_Boen = Wind_Boen;
213             Wind_Boen_alt = Wind_Boen;
214         }
215     }
216
```

```
217     Daten_modifiziert.Wind_Richtung = (Daten_roh[7] & 0x0F);   //Richtung
218
219 //Temperatur 2 ermitteln (frei für den zweiten Aussensensor)
220     Temp_2_Summe += 0;
221 //Feuchte 2 ermitteln
222     Feuchte_2_Summe += 0;
223     Anzahl_2++;
224
225 //Temperatur 3 ermitteln Innentemperatur
226     Temp_3_Summe +=  (int16_t)(Temperatur_kompensiert(Temperatur_roh())/10); //vom BME280 Sensor
227 //Feuchte 3 ermitteln
228     Feuchte_3_Summe += (uint8_t)(Feuchte_kompensiert(Feuchtigkeit_roh())>>10);
229     Anzahl_3++;
230
231     if(Daten_roh[10] == 0x55)   //ist wahrscheinlich immer 0x55
232     {
233         //Helligkeit ermitteln
234         Helligkeit_Summe += (((uint32_t)Daten_roh[11] << 16) + ((uint32_t)Daten_roh[12] << 8) +
        (uint32_t)Daten_roh[13]) / 1000; //Helligkeit in 0,1kLux
235         //UV Index ermitteln
236         UV_I_Summe += Daten_roh[9] & 0x0F;
237         Anzahl_Helligkeit ++;
238     }
239
240
241 //Batterie ?
242     Daten_modifiziert.Batterie = (Daten_roh[7] & 0xF0);   //Batterie ?
243
244 //Fragezeichen Byte 10 vom Lichtsensor
245     Daten_modifiziert.Fragezeichen = Daten_roh[10];   //keine Ahnung wofür das ist (0x55)
246
247
248     if (Ende)
249   {
250     //Mittelwerte bilden und Daten in Ausgabe struct Datenausgabe schreiben
251     //Wind_mittel wird auch im Sensor 15min? gemittelt
252
253         Fehler = 0; //Fehlermarker rücksetzen
254         Daten_modifiziert.Temp1 = (int16_t) (Temp_1_Summe / Anzahl_1);
255         Daten_modifiziert.Feuchte1 = (uint8_t) (Feuchte_1_Summe / Anzahl_1);
256         Daten_modifiziert.Temp2 = (int16_t) (Temp_2_Summe / Anzahl_2);
257         Daten_modifiziert.Feuchte2 = (uint8_t) (Feuchte_2_Summe / Anzahl_2);
258         Daten_modifiziert.Temp3 = (int16_t) (Temp_3_Summe / Anzahl_3);
259         Daten_modifiziert.Feuchte3 = (uint8_t) (Feuchte_3_Summe / Anzahl_3);
260         Temp_1_Summe = 0;
261         Feuchte_1_Summe = 0;
262         Anzahl_1 = 0,
263         Temp_2_Summe = 0;
264         Feuchte_2_Summe = 0;
265         Anzahl_2 = 0;
266         Temp_3_Summe = 0;
267         Feuchte_3_Summe = 0;
268         Anzahl_3 = 0;
269
270
271
272         //Regensensor
273         Daten_modifiziert.Regen =(uint8_t) Regenmenge;
274         Regenmenge = 0;
275
276         //Helligkeitssensor
277         Daten_modifiziert.Helligkeit = (uint16_t) (Helligkeit_Summe / Anzahl_Helligkeit);
278         Daten_modifiziert.UV_I = UV_I_Summe / Anzahl_Helligkeit;
279         UV_I_Summe = 0;
280         Helligkeit_Summe = 0;
281         Anzahl_Helligkeit = 0;
282
283         //Windsensor
284         Wind_Boen_alt = 0;
285   }
286
287   return(Fehler) ;
288
289 }
290
```

```
291
292
293  uint16_t Daten_in_EEPROM_schreiben(uint16_t DS_Nr)
294 {
295   //Daten ins EEPROM schreiben. Der Rückgabewert ist der nächste Datensatz
296      //Aufbau des Datensatzes:
297      //
         T,M,J,H,Min,T1_L,T1_H,T2_L,T2_H,T3_H,T3_L,F1,F2,F3,H_L,H_H,UV_I,Druck_L,Druck_H,R,Wm_L,Wm_H,WR,W
         b_L,Wb_H,B ,?
298      //0,1,2,3,4  ,5   ,6   ,7   ,8   ,9   ,10  ,11,12,13,14 ,15 ,16  ,17     ,18    ,19,20  ,21  ,
         22,23  ,24  ,25,26
299     uint8_t  n;
300     uint8_t Datensatz[Anzahl_Byte_DS];
301     uint16_t Adr_intern;
302     uint32_t Adr;
303     uint8_t Adresse_EEPROM_intern;
304
305
306     //Signalisiert die Speicherung eines Datensatzes
307      LED_Empfang_an;
308      _delay_ms(1000);
309      LED_Empfang_aus;
310
311      Lesen_Datum_Uhrzeit_kurz(Datum_Zeit_k);
312
313      //Jährlich wird die Adresse im EEPROM gewechselt, um Ausfälle vorzubeugen.
314      //reicht für 15 Jahre
315      if ((Datum_Zeit_k[2] - 20) != wechsel)
316      {
317        wechsel = Datum_Zeit_k[2] - 20;
318        eeprom_write_byte(&ee_wechsel, wechsel);
319      }
320
321      //Tag, Monat, Jahr, Std und Minute in die ersten 5 Byte schreiben
322      for (n=0;n<5;n++)
323      {
324         Datensatz[n] = Datum_Zeit_k[n];
325      }
326
327      DS_Ausgabe.Daten_mod = Daten_modifiziert;  //in union überführen
328      for (n=0; n<sizeof (struct Datenausgabe); n++)
329      {
330         Datensatz[n+5] = DS_Ausgabe.Daten_ausgabe[n];
331      }
332
333       if ( DS_Nr <= DS_Anzahl_Block)
334       {
335         Adresse_EEPROM_intern = Adresse_EEPROM;
336         Adr = DS_Nr * Anzahl_Byte_DS;
337       }
338       else
339       {
340         Adr = (DS_Nr + 2) * Anzahl_Byte_DS; //Adresse für den zweiten Block im EEPROM
341         Adresse_EEPROM_intern = Adresse_EEPROM | 0x08; //Wenn Adresse > 0x0000FFFF, dann EEPROM Bit 3
           setzen
342       }
343
344      for (n=0;n<Anzahl_Byte_DS;n++)
345      {
346         Adr_intern = (uint16_t) (Adr & 0x0000FFFF);
347        //  Schreibe_Byts_ab_Adresse(Adresse_EEPROM_intern, Adr_intern, Datensatz, Anzahl_Byte_DS, 2);
348         Schreibe_Byte_an_Adresse(Adresse_EEPROM_intern,Adr_intern,Datensatz[n],2);
349         _delay_ms(10);
350         Adr++;
351      }
352      //letzte DS Adresse ins EEPROM schreiben. Adresse im EEPROM wird jedes Jahr geändert
353      //Bei 10 min reicht es für ca. 2 Jahre (100000 mal schreiben)
354      eeprom_write_word(&ee_Adresse[wechsel], DS_Nr + 1);
355
356     return (DS_Nr + 1);
357 }
358 /*
359 void Lesen_letzten_DS_EEPROM(uint8_t Datensatz[Anzahl_Byte_DS])
360 {
361     uint8_t Adresse_EEPROM_intern;
```

```
362       uint16_t Adr_intern;
363       uint32_t z;
364
365       z = (DS_Nr-1) * Anzahl_Byte_DS;   //Berechnung der Speicheradresse
366
367         if ((DS_Nr-1) > DS_Anzahl_Block)    //2. Block wird beschrieben Adresse > 0xFFFF
368         {
369             z += 2 * Anzahl_Byte_DS;
370             Adresse_EEPROM_intern = Adresse_EEPROM | 0x08;
371         }
372         else
373         {
374             Adresse_EEPROM_intern = Adresse_EEPROM;
375         }
376
377         Adr_intern = (uint16_t) (z & 0x0000FFFF);
378         Lese_Byts_ab_Adresse(Adresse_EEPROM_intern, Adr_intern, Datensatz, Anzahl_Byte_DS, 2);
379 }
380   */
381 void Lesen_aus_EEPROM(uint16_t DS_Nr, uint16_t Anzahl)
382 {
383   //liest von der Datensatznummer ab, Anzahl Datensätze und sendet sie über die serielle
      Schnittstelle.
384
385       uint8_t Datensatz[Anzahl_Byte_DS] =
        {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,0};
386       uint8_t Adresse_EEPROM_intern;
387       uint16_t m, Adr_intern;
388       uint8_t n;
389       uint32_t z;
390
391       cli();
392       for (m=DS_Nr;m<DS_Nr+Anzahl;m++)
393       {
394         z = m * Anzahl_Byte_DS; //Berechnung der Speicheradresse
395
396         if (m > DS_Anzahl_Block)    //2. Block wird beschrieben Adresse > 0xFFFF
397         {
398             z += 2 * Anzahl_Byte_DS;
399             Adresse_EEPROM_intern = Adresse_EEPROM | 0x08;
400         }
401         else
402         {
403             Adresse_EEPROM_intern = Adresse_EEPROM;
404         }
405
406         Adr_intern = (uint16_t) (z & 0x0000FFFF);
407        Lese_Byts_ab_Adresse(Adresse_EEPROM_intern, Adr_intern, Datensatz, Anzahl_Byte_DS, 2);
408         //DS ausgeben
409         for (n=0;n<Anzahl_Byte_DS;n++)
410         {
411             UART_Sende_Zeichen(Datensatz[n]);
412         }
413         //    wdt_reset();  //Die Datenübertragung kann länger als 1s dauern.
414         sei();
415       }
416
417    return;
418 }
419
420 void Daten_lesen()
421 {
422   uint16_t Anzahl;
423     //Anzahl der vorhandenen Datensätze über serielle Schnittstelle senden
424     UART_Sende_Zeichen((uint8_t)(DS_Nr & 0x00FF));  //L_Byte zuerst
425     UART_Sende_Zeichen((uint8_t)((DS_Nr & 0xFF00)>>8));
426
427     //Anzahl der zu sendenden Datensätze empfangen
428     Anzahl = (UART_Empfange_Zeichen() & 0x00FF) <<8;    //H_Byte
429     Anzahl += UART_Empfange_Zeichen() & 0x00FF; //L_Byte
430     while (UART_Empfange_Zeichen() != UART_KEINE_DATEN);    //Puffer leeren
431
432     if (Anzahl > DS_Nr) Anzahl = DS_Nr; //Wenn Anforderung > vorhanden auf vorhandene DS reduzieren
433
434     Lesen_aus_EEPROM(0, Anzahl);
```

```
435     return;
436 }
437 
438 
439 void Uhr_stellen()
440   {
441         //Lese 12Byte von der seriellen Schnittstelle jjmmttHHMMSS
442         //zum Stellen der Uhr
443     char Datum_Uhrzeit_Ascci[12];
444     uint8_t n;
445 
446       UART_Sende_Zeichen(Anforderung);  //muß gemacht werden, damit der Rechner hier wartet bevor er
        sendet.
447       _delay_ms(10);    //bis 12 Zeichen geladen sind.
448 
449        for (n=0;n<12;n++)
450        {
451          Datum_Uhrzeit_Ascci[n] =(uint8_t) (UART_Empfange_Zeichen() & 0x00FF);
452        }
453         for (n=0;n<12;n++)
454        {
455          UART_Sende_Zeichen(Datum_Uhrzeit_Ascci[n]);
456        }
457 
458        Stellen_Datum_Uhrzeit_kurz(Datum_Uhrzeit_Ascci);
459        //Signalisierung, daß Uhr gestellt wurde
460        LED_Batterie_an;
461         _delay_ms(1000);
462        LED_Batterie_aus;
463 
464     return;
465   }
466 
467 void Letzten_DS_auf_0(uint16_t letzter_DS)
468 {
469     //UP liest den letzten Datensatz und speichert ihn auf DS_Nr 0
470     uint8_t Datensatz[Anzahl_Byte_DS];
471     uint8_t Adresse_EEPROM_intern;
472     uint16_t Adr;
473     uint32_t z;
474 
475     z = letzter_DS * Anzahl_Byte_DS;
476 
477     if (letzter_DS > DS_Anzahl_Block)
478     {
479         z += 2 * Anzahl_Byte_DS;
480         Adresse_EEPROM_intern = Adresse_EEPROM | 0x08;
481     }
482     else
483     {
484         Adresse_EEPROM_intern = Adresse_EEPROM;
485     }
486 
487     Adr = (uint16_t) (z & 0x0000FFFF);
488     Lese_Byts_ab_Adresse(Adresse_EEPROM_intern, Adr, Datensatz, Anzahl_Byte_DS, 2);
489 
490     Adr = 0;    //Schreibe Datensatz ab Adresse 0
491     Schreibe_Byts_ab_Adresse(Adresse_EEPROM_intern, Adr, Datensatz, Anzahl_Byte_DS, 2);
492 
493     DS_Nr = 1;  //Globale DS_Nr
494     eeprom_write_word(&ee_Adresse[wechsel], DS_Nr); //Datensatznummer = 1 speichern
495 
496     return;
497 
498 }
499 
500 void Timer0_ini()
501 {
502     //Timer für die Überschreitung der Wartezeit Eingangssignal (3ms)
503     TIMSK0 |= (1<<OCIE0A);   //Interrupt für Vergleich mit Register A
504     TCCR0A |= (1<<WGM01);    //CTC Modus
505     TCCR0B |=  (1<<CS02); //Teiler 256
506     OCR0A = 187; //Für 3ms bei Vorteiler 256 und 16MHz
507     TCNT0 = 0;  //Anfang Timerzähler
508 
```

```
509 }
510
511 void Timer1_ini()
512 {
513     //Timer für die Selektierung ob zwei Datensätze empfangen werden.
514     TIMSK1 |= (1<<OCIE1A);
515     TCCR1B |= (1<<CS12);    // | (1<<WGM12);    //Vorteiler = 256; verkürzter Zählumfang
516     OCR1A = 12500;    //200ms
517     TCNT1 = 0;
518 }
519
520
521 void Timer2_ini()
522 {
523     //Timer für die Überschreitung des Kopfsignals 8*1,5ms
524     TIMSK2 |= (1<<OCIE2A);  //Vergleich mit Register A
525     TCCR2A |= (1<<WGM21);   //CTC Funktion
526     OCR2A = 200;    //12,8ms
527     TCNT2 = 0;  //Zähler auf Null setzen
528    // TCCR2B |= (1<<CS22) | (1<<CS21) | (1<<CS20);    //Vorteiler = 1024
529
530 }
531
532 void Port_interrupt()
533 {
534     PCMSK0 |= (1<<PCINT0);  //PIN B0 als Interrupt benutzen
535     PCICR |= (1<<PCIE0);  //Interrupt freigeben
536 }
537
538
539 ISR (PCINT0_vect)
540 {
541    //Timer2 starten und die Dauer der Kennung prüfen zwischen 11 und 12,8ms
542
543     Timer2_an;
544     Anzahl_Flanken++;
545     if ((Anzahl_Flanken >= 17) )   //nach der Kennung Daten lesen
546     {
547
548         PCICR &= ~(1<<PCIE0); //Interrupt abschalten
549         Timer2_aus; //Timer2 anhalten
550
551         if(TCNT2 < 156) //kleiner als 11ms Kopfdaten: Fehler
552         {
553             //Timer2_aus;
554             TCNT2 = 0;
555             Anzahl_Flanken = 0;
556             _delay_ms(20); //warten bis Störung vorbei ist
557         }
558         else
559         {
560             fehl = Daten_einlesen(Daten_roh);
561
562             TCNT2 = 0;  //Zähler auf Null setzen
563             Anzahl_Flanken = 0;
564             cli();
565             PCICR |= (1<<PCIE0);
566             sei();
567         }
568     }
569
570 }
571
572 ISR (TIMER2_COMPA_vect)
573 {
574     //Timer für die Zeit der Kopfdaten 0xFF (11...13ms)
575     Timer2_aus;
576     TCNT2 = 0;
577     Anzahl_Flanken = 0;
578 }
579
580 ISR (TIMER1_COMPA_vect)
581 {
582     //Timer für rücksetzen des Paketzählers, wenn nur ein Paket (200ms)
583     Paket_Zaehler = 0;
```

```
584     Timer1_aus;
585     TCNT1 = 0;  //Zähler auf Null setzen
586 }
587
588 ISR (TIMER0_COMPA_vect)
589 {
590     //Timer für fehlerhafte Daten (3ms)
591     Fehler = 0x01;
592 }
593
594
595 void WD_Timer_init()
596 {
597     WDTCSR |= (1<<WDCE) | (1<<WDE); //Reset Betrieb
598     WDTCSR |= (1<<WDP1) | (1<<WDP2);    //1s
599
600   //  wdt_enable(WDTO_1S);
601
602 }
603
604 void Ports_init()
605 {
606     DDRB &= ~(1<<Daten_Eingang);    //Empfängereingang
607     DDRB |= (1<<LED_Fehler);
608   //DDRC &= ~(1<<MISO); //Eingang Daten
609     //I2C Bus in i2c.h definiert
610     DDRC |= (1<<MOSI) | (1<<SCK) | (1<<NSS);    //MOSI, SCK und NSS als Ausgang
611     DDRC |= (1<<LED_Empfang);   //LED als Ausgang
612     DDRD |= (1<<LED_Batterie);  //Led Batterie Ausgang
613     return;
614 }
615
616
617 void Startwerte_setzen()
618 //Setzt Startwerte
619 {
620   wechsel = eeprom_read_byte(&ee_wechsel);
621   DS_Nr = eeprom_read_word(&ee_Adresse[wechsel]);
622   Speicherintervall = eeprom_read_byte(&ee_Speicherintervall);
623   return;
624 }
625
626
627 int main(void)
628 {
629
630     Ports_init();
631     Port_interrupt();
632     _delay_ms(100);
633     RFM63_Config();
634     Timer0_ini();
635     Timer1_ini();
636     Timer2_ini();
637     Timer2_an;
638     Startwerte_setzen();
639
640     BME_280_init();
641     _delay_ms(20);
642     if (BME_280_test() == 0)  BME_280_Parameter_lesen();
643
644     uart_init(BR_Teiler(F_CPU,Baudrate));   //serielle Schnittstelle initialisieren Parameter in
    ▒▒▒▒uart.c
645
646     // WD_Timer_init();
647
648     sei();
649
650     while(1)
651     {
652
653        if(fehl == 0)
654        {
655            cli();
656             BME_280_init();  //BME Messung starten
657
```

```
658             Lesen_Datum_Uhrzeit_kurz(Datum_Zeit_k);
659             //Minuten = Datum_Zeit_k[4]; Sekunde = Datum_Zeit_k[5];
660
661             if ((Datum_Zeit_k[4] % Speicherintervall) == 0)    // || (((Datum_Zeit_k[4] %
                Speicherintervall) == 1) && (Datum_Zeit_k[5] < 3)))    //62 s Abfragebereitschaft
662             {
663                 Stop = 1;
664             }
665             else
666             {
667                 Stop = 0;
668                 Datensatz_eingelesen = 0;   //Nach einer Minute Rücksetzen
669             }
670
671             if(Datensatz_eingelesen > 0) Stop = 0; //innerhalb der Minute wurde der Datensatz schon
                eingelesen
672
673             Fehler = Mittelwertbildung(Daten_roh,Stop);
674
675             //Die letzten 300 DS des Speichers werden belegt. Batterie LED an
676             if (DS_Nr > ((DS_Anzahl_Block * 2) - 300))
677             {
678                 LED_Batterie_an;
679             }
680             else
681             {
682                 LED_Batterie_aus;
683             }
684
685             if (Stop)
686             {
687             //Daten ins EEPROM schreiben
688
689                 DS_Nr = Daten_in_EEPROM_schreiben(DS_Nr);
690
691                 Stop = 0;
692                 Datensatz_eingelesen = 1;
693                  LED_Batterie_an;
694                 _delay_ms(200);
695                 LED_Batterie_aus;
696             // Fehler_anzeigen(Fehler);
697             }
698
699             fehl = 4;   //Fehlerspeicher löschen
700
701             sei();
702
703             LED_Empfang_an;
704             _delay_ms(500);
705             LED_Empfang_aus;
706         }
707
708         else if(fehl == 1)   //Fehler beim Datenlesen
709         {
710             LED_Fehler_an;
711             _delay_ms(200);
712             LED_Fehler_aus;
713             fehl = 4;   //Fehlerspeicher löschen
714         }
715
716        else
717        {
718              //Auswahlzeichen empfangen
719              Anforderung = (uint8_t) (UART_Empfange_Zeichen() & 0x00FF);
720
721
722             if (Anforderung == 'l') //Daten lesen
723             {
724                 Daten_lesen();
725             }
726
727             else if(Anforderung == 's')
728             {
729                 Uhr_stellen();
730             }
```

```
731
732             else if (Anforderung == 'e')    //Speicherintervall einstellen
733             {
734                 _delay_us(160); //Wartezeit für das nächste Zeichen
735
736                 Speicherintervall = (uint8_t)(UART_Empfange_Zeichen() & 0x00FF);
737
738                 eeprom_write_byte(&ee_Speicherintervall, Speicherintervall);
739
740                 UART_Sende_Zeichen(Speicherintervall);
741                 Anforderung = 0;
742
743                 LED_Batterie_an;
744                 _delay_ms(500);
745                 LED_Batterie_aus;
746             }
747
748             else if (Anforderung == 'z')    //Speicherzähler rücksetzen
749             {
750                 Letzten_DS_auf_0(DS_Nr-1);
751
752                 LED_Batterie_an;
753                 _delay_ms(500);
754                 LED_Batterie_aus;
755             }
756
757             Anforderung = 0;
758
759             cli();
760             PCICR |= (1<<PCIE0);
761             sei();
762         //    wdt_reset();
763         }
764
765     }
766   return(0);
767 }
768
```