```c
/*Geschwindigkeitsmessung einer Luftgewehrkugel an der Mündung.
 * Im Abstand von 10cm befinden sich zwei Lichtschranken, die durch die Luftgewehkugel unterbrochen wird.
 * Die erste Lichtschranke löst einen externen Interrupt aus, der dann den Zähler initialisiert.
 * Die zweite Lichtschranke hält den Zähler wieder an.
 *
 * Die Masse der Luftgewehrkugel kann im Bereich zwischen 0,45 und 0,6g eingestellt werden.
 * Dazu wird vor dem Einschalten die Taste Reset gedrückt und etwa 6s gehalten.
 * Das Gewicht kann nun mit der Taste Reset eingestellt werden(etwa 1s halten).
 * Das Menü wird durch halten der Reset Taste für etwa 4s wieder verlassen.
 *
 * Prozessor: ATMEGA 328  h-Fuse DFh; l-Fuse CCh
 * Version 0.0 8/19 pkr*/


#define F_CPU 4000000UL

#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/eeprom.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "bibliothek/e-paper-1,54.c"

#define LS1 2   //Lichtschranke Interrupt Auslöser PIN D2
#define LS2 0   //Lichtschranke nach 10cm an PIN B0
#define Taster 2    //PIN B2 für Taster
#define LED 1   //PIN B1



#define Timer1_start (TCCR1B |= (1<<CS10))
#define Timer1_halt (TCCR1B &= ~(1<<CS10))
#define Int_eingang_ein  (PCICR |= (1<<PCIE0))
#define Int_eingang_aus (PCICR &= ~(1<<PCIE0))
#define Int_ausgang_ein (TIMSK1 |= (1<<ICIE1))
#define Int_ausgang_aus (TIMSK1 &= ~(1<<ICIE1))
#define Taster_ein !(PINB & (1<<Taster))
#define Spannung_Kanal0    0


//Position für die Werte auf der Anzeige
#define y_Ausgabeanfang 199
#define x_v0 12
#define y_v0 y_Ausgabeanfang-60
#define x_Energie 16
#define y_Energie y_Ausgabeanfang-60
#define x_Masse 22
#define y_Masse y_Ausgabeanfang-60
#define LED_an (PORTB |= (1<<LED))
#define LED_aus (PORTB &= ~(1<<LED))

//RAM
char Ueberschrift[] = {"V0 Messung"};
char text1[] = {"V=     m/s"};
char text2[] = {"E=     J"};
char text3[] = {"m=     g"};
char Leerzeichen[] = {"     "};    //4 Leerzeichen

//Globale Variable
char Temporaer[] = {"0123456"};
uint16_t Zaehlerstand = 0;
uint16_t v0=10;    //m/s
uint8_t Energie;    //in J*10
uint8_t Masse = 45; //Masse des Geschosses in g*100 (Field Target Trophy=0,56g; DDR=0,48g)
uint8_t Merker_Ausgang = 0;
uint8_t Merker_Eingang = 0;
uint8_t Masse_Menue = 0;
uint8_t n;

//EEPROM
uint8_t ee_Masse EEMEM = 45;    //Masse des Geschosses
char ee_Version[] EEMEM = {"V0 Messung pkr 8/19 Version 0.0"};
```

```
 75
 76 //UP
 77 uint8_t Masse_einstellen();
 78 void Zaehler_auswerten(uint8_t Masse_Geschoss);
 79 void Grundinitialisierung_Anzeige();
 80 void Grundtexte();
 81 void Masse_Grundtexte();
 82 void Anzeige();
 83 void Batterieanzeige(uint16_t ADC_ausgabe);
 84 uint16_t ADC_Wert();
 85 void Timer1_init();
 86 void Interrupt_init();
 87 void Ports_init();
 88 void EEPROM_lesen();
 89
 90 uint8_t Masse_einstellen()
 91 {
 92     uint8_t n;
 93
 94         n=0;
 95             if(Taster_ein != 0)
 96             {
 97                 _delay_ms(5);
 98
 99                 while (Taster_ein != 0) //Warte bis Taste losgelassen
100                 {
101                     _delay_ms(100);
102                     n++;
103                 }
104
105                 if(n > 20)
106                 {
107                     n = 0;
108                     Masse_Menue = 0;
109                     eeprom_write_byte(&ee_Masse, Masse);
110                     Reset();
111                     Display_Init(lut_full_update);
112                     Grundinitialisierung_Anzeige();
113                     return(Masse);
114                 }
115
116                 Masse++;
117                 if (Masse > 60) Masse = 45; //Bereich eingrenzen
118             }
119             eeprom_write_byte(&ee_Masse, Masse);
120
121     return (Masse);
122 }
123
124
125
126 void Zaehler_auswerten(uint8_t Masse_Geschoss)
127 {
128     //Berechnung der Geschwindigkeit und Energie
129
130     if((Zaehlerstand < 40000) && (Zaehlerstand > 1600)) //10m/s <v0> 250m/s
131     {
132         v0= 400000/(Zaehlerstand);   //v0
133     }
134     else
135     {
136         v0 = 1;
137     }
138
139     Energie = (uint8_t)(((uint32_t)Masse_Geschoss * (uint32_t)v0 *(uint32_t)v0) / (2 * 10000));
140 }
141
142 void Grundinitialisierung_Anzeige()
143 {
144     //Wenn der ganze Bildschirm beschrieben wird, eine Pause von 2s
145     //nach dem Schreiben einhalten.
146
147     Schreibe_Bildschirm_voll(weiss);  //0xFF
148     _delay_ms(2000);
149
```

```
150     Schreibe_Bildschirm_voll(weiss);  //weiss
151     _delay_ms(2000);
152 }
153
154 void Grundtexte()
155 {
156     char Masse_String[] = {"0,56"};
157     uint8_t Masse_ganz;
158     uint8_t Masse_nachk;
159
160     //Hier stehen die Texte, die nicht verändert werden
161     Schreibe_Text(Ueberschrift, sans_mono_24, 5, y_Ausgabeanfang);
162     Schreibe_Text(text1, sans_mono_24, x_v0, y_Ausgabeanfang);
163     Schreibe_Text(text2, sans_mono_24, x_Energie, y_Ausgabeanfang);
164     Schreibe_Text(text3, sans_mono_24, x_Masse, y_Ausgabeanfang);
165     Zeichne_Strich(6,200,200,4);
166     Batterieanzeige(ADC_Wert());
167     //Masse anzeigen
168     Masse_ganz = Masse/100;  //Ganzahliger Massewert
169     Masse_nachk = (uint8_t)(Masse - (Masse_ganz * 100)); //Nachkommawert
170     Schreibe_Text(Leerzeichen, sans_mono_24,  x_Masse, y_Masse);   //Anzeige löschen
171     utoa(Masse_ganz, Temporaer, 10);
172     strcpy(Masse_String, Temporaer);
173     strcat(Masse_String, ",");
174     itoa(Masse_nachk, Temporaer, 10);
175     strcat(Masse_String, Temporaer);
176     Schreibe_Text(Masse_String, sans_mono_24, x_Masse, y_Masse);
177 }
178
179 void Masse_Grundtexte()
180 {
181     Schreibe_Text("  Masse", sans_mono_24, 5, y_Ausgabeanfang);
182     Zeichne_Strich(6,200,200,4);
183     Schreibe_Text("m = 0,  g", sans_mono_24, 12, y_Ausgabeanfang);
184     Schreibe_Text(" Die Masse wird nur ", sans_mono_12, 15, y_Ausgabeanfang);
185     Schreibe_Text(" zur Berechnung der ", sans_mono_12, 17, y_Ausgabeanfang);
186     Schreibe_Text(" Energie gebraucht. ", sans_mono_12, 19, y_Ausgabeanfang);
187     Schreibe_Text(" Die Messung wird   ", sans_mono_12, 22, y_Ausgabeanfang);
188     Schreibe_Text(" nicht beeinflusst. ", sans_mono_12, 24, y_Ausgabeanfang);
189 }
190
191
192 void Anzeige()
193 {
194 //UP für die Anzeige von Geschwindigkeit und Energie
195
196     char Energie_String[] = {"10,0"};
197     uint8_t Energie_ganz;
198     uint8_t Energie_nachk;
199
200     //v0 anzeigen
201     Schreibe_Text(Leerzeichen, sans_mono_24,  x_v0, y_v0);   //Anzeige löschen
202     utoa(v0, Temporaer, 10);
203     Schreibe_Text(Temporaer, sans_mono_24, x_v0, y_v0);
204
205    //Energie anzeigen. Eine Nachkommastelle
206     Energie_ganz = Energie/10;  //Ganzahliger Energiewert
207     Energie_nachk = (uint8_t)(Energie - (Energie_ganz * 10)); //Nachkommawert
208
209     Schreibe_Text(Leerzeichen, sans_mono_24,  x_Energie, y_Energie);   //Anzeige löschen
210     utoa(Energie_ganz, Temporaer, 10);
211     strcpy(Energie_String, Temporaer);
212     strcat(Energie_String, ",");
213     itoa(Energie_nachk, Temporaer, 10);
214     strcat(Energie_String, Temporaer );
215     Schreibe_Text(Energie_String, sans_mono_24, x_Energie, y_Energie);
216
217     Display_Fram();
218 }
219
220 void Batterieanzeige(uint16_t ADC_ausgabe)
221 {
222     //Bei 4,05V beträgt der ADC_Wert 970
223 #define x_Aus 21
224 #define y_Aus 20
```

```
225     uint16_t Spannung;  //Batteriespannung * 100
226     Schreibe_Text("          ", sans_mono_12, x_Aus + 3, 180);
227     /*Zum Eichen der Spannungsmessung
228     utoa(ADC_ausgabe, Temporaer, 10);
229     Schreibe_Text(Temporaer, sans_mono_12, x_Aus + 3, 180);
230     */
231     Spannung = (uint16_t) ((405 * (uint32_t)ADC_ausgabe) / 970);
232 
233     if (Spannung > 400)
234     {
235           //Batteriestand Ausgabe 100%
236         Schreibe_Buchstabe(0, Batterie, x_Aus, y_Aus);
237     }
238     else if ((Spannung > 375) && (Spannung <= 400))
239     {
240         //Batteriestand Ausgabe 75%
241         Schreibe_Buchstabe(1, Batterie, x_Aus, y_Aus);
242     }
243     else if ((Spannung > 356) && (Spannung <= 375))
244     {
245         //Batteriestand Ausgabe 50%
246         Schreibe_Buchstabe(2, Batterie, x_Aus, y_Aus);
247     }
248     else if ((Spannung > 325) && (Spannung <= 356))
249     {
250         //Batteriestand Ausgabe 25%
251         Schreibe_Buchstabe(3, Batterie, x_Aus, y_Aus);
252         Schreibe_Text("Akku laden", sans_mono_12, x_Aus + 3, 180);
253     }
254     else
255     {
256         //Batteriestand Ausgabe 0%
257         Schreibe_Buchstabe(4, Batterie, x_Aus, y_Aus);
258         Schreibe_Text("Akku leer", sans_mono_12, x_Aus + 3, 180);
259     }
260 
261     Schreibe_Text("Batt", sans_mono_12, x_Aus + 3, y_Aus + 20);
262 }
263 
264 uint16_t ADC_Wert()
265 {
266 //UP gibt den Digitalwert der Spannung am Kanal zurück
267 
268     ADMUX &= ~((1<<MUX3) | (1<<MUX2) | (1<<MUX1) | (1<<MUX0)); //Kanal 0
269 
270     ADCSRA |= (1<<ADSC);    //Starte Wandlung
271     while (!(ADCSRA & (1<<ADIF)));  //Schleife bis Wandlung fertig
272     ADCSRA &= ~(1<<ADIF);   //Lösche ADIF
273 
274     return ADC; //Rückgabe des Digitalwertes
275 }
276 
277 ISR (TIMER1_CAPT_vect)
278 {
279     Zaehlerstand = ICR1;    //Zählerstand in Variable
280     Timer1_halt;
281     Merker_Ausgang = 1;
282     TCNT1 = 0;  //Zähler zu Null setzen
283 }
284 
285 ISR (INT0_vect)
286 {
287     Timer1_start;   // Timer starten
288     Merker_Eingang = 1;
289 }
290 
291 void Timer1_init()
292 {
293     //Timer1 im Normalmodus
294     TCCR1B &= ~((1<<CS10) | (1<<ICES1));    // Zähler halt; Vorteiler = 1 bei 4MHz 0,25us/Takt;
    Negative Flanke an B1 auswerten
295 
296     TCNT1 = 0; //Zähler auf 0 setzen
297     TIMSK1 |= (1<<ICIE1);   //Interrupt für den Timereingang B0 setzen
298 }
```

```
299
300
301 void Interrupt_init()
302 {
303     //Initialisiert den externen Interrupt 0 (PIN D2)
304     EIMSK |= (1<<INT0);
305     EICRA |= (1<<ISC11);
306 }
307
308
309 void Ports_init()
310 {
311     //Port D für die Anzeige
312     DDRD &= ~((1<<E_Busy) | (1<<LS1));  //Busy und Lichtschranke 1 Eingang
313     PORTD |= (1<<E_Busy);// | (1<<LS1);   //Pullup Widerstand einschalten
314
315     DDRD |= (1<<E_Reset) | (1<<E_DC) | (1<<E_CS) | (1<<E_Clk) | (1<<E_DIN); //als Ausgang
316     // Port B für Lichtschranke und Taster
317     DDRB &= ~(1<<Taster);
318     PORTB |= (1<<Taster);   //Pullup Widerstände einschalten
319     DDRB |= (1<<LED);
320     DDRB &= ~(1<<LS2);  //Lichtschranke2 als Eingang
321     DDRC &= ~(1<<Spannung_Kanal0);  //Analogeingang0 als Eingang
322     ADMUX |= (1<<REFS1);    //Referenzspannung 1,1V
323     ADCSRA |= (1<<ADEN);    //AD-Wandler freigeben
324     return;
325 }
326
327 void EEPROM_lesen()
328 {
329     Masse = eeprom_read_byte(&ee_Masse);
330 }
331
332 int main()
333 {
334
335     wdt_disable();
336     Ports_init();
337     Interrupt_init();
338     Timer1_init();
339     EEPROM_lesen();
340     LED_aus;
341
342     Reset();
343     Display_Init(lut_full_update);
344     Grundinitialisierung_Anzeige();
345
346     if(Taster_ein != 0)
347     {
348         _delay_ms(5);    //Eingangsprellung
349         while (Taster_ein != 0){};    //warten bis Taste losgelassen
350         Masse_Menue = 1;
351         Masse_Grundtexte();
352         Display_Init(lut_partial_update); //Ab jetzt nur noch teilweise Aktualisierung
353         Display_Fram();
354         _delay_ms(500);
355     }
356
357     if(Masse_Menue != 1)    //Normalmodus
358     {
359         Display_Init(lut_partial_update); //Ab jetzt nur noch teilweise Aktualisierung
360         Grundtexte();
361         Display_Fram();
362         _delay_ms(500);
363         sei();
364     }
365
366     while(1)
367       {
368         if(Masse_Menue == 1)  //Masse einstellen
369           {
370               while(Masse_Menue != 0)
371               {
372                 Masse_Grundtexte();
373                 utoa(Masse, Temporaer, 10);
```

```
374                 Schreibe_Text(Temporaer, sans_mono_24, 12, 80);
375                 Display_Fram();
376                 Masse = Masse_einstellen();
377                _delay_ms(200);
378               }
379               //Einstellmenü für die Masse verlassen
380                 Display_Init(lut_partial_update); //Ab jetzt nur noch teilweise Aktualisierung
381                 Grundtexte();
382                 Display_Fram();
383                 _delay_ms(500);
384                 sei();
385           }
386         else  //Normalmodus Geschwindigkeit messen
387           {
388             if(Taster_ein != 0)
389             {
390                 _delay_ms(50);    //Eingangsprellung
391                 while (Taster_ein != 0){};    //warten bis Taste losgelassen
392 
393                 Grundtexte();
394                 Schreibe_Text(Leerzeichen, sans_mono_12,  x_v0, y_v0);   //Anzeige löschen
395                 Schreibe_Text(Leerzeichen, sans_mono_12,  x_Energie, y_Energie);   //Anzeige löschen
396                 Display_Fram();
397 
398                 Merker_Ausgang = 0; //Merker löschen
399                 Merker_Eingang = 0;
400                 EIFR |= (1<<INTF0); //Innerrupt Flag löschen
401                 TIFR1 |= (1<<ICF1); //Timer Interrupt Flag
402 
403                 TIMSK1 |= (1<<ICIE1);   //Interrupt freigeben
404                 EIMSK |= (1<<INT0); //Interrupt freigeben
405                 LED_aus;
406             }
407 
408             if(Merker_Eingang == 1)
409             {
410                 _delay_ms(10);   //Warten auf Interrupt Ausgang
411                 LED_an;
412                 Merker_Eingang = 0;
413             }
414 
415             if(Merker_Ausgang == 1 )
416             {
417                 EIMSK &= ~(1<<INT0);    //Eingangsinterrupt halt
418                 TIMSK1 &= ~(1<<ICIE1); //Timer Interrupt halt
419                 Zaehler_auswerten(Masse);
420                 Grundtexte();
421                 Anzeige();
422                 _delay_ms(500); //für LED Anzeige
423                 LED_aus;
424                 Merker_Ausgang = 0;
425                 _delay_ms(2000);
426 
427             }
428           }
429       }
430 
431 return (0);
432 }
433 
434 
435 
```