```
1: program Datenlogger;
2: {
3:  18/16-Bit Datenlogger für die Erfassung von 4 analogen Differential-Eingängen.
4:  Der Messbereich beträgt -2,048V bis 2,048V.
5:  Für Single-Ended Messungen kann der Minus-Eingang per Jumper auf Masse gelegt
6:  werden.
7:
8:  Die Werte werden als ASCII-String über die serielle Schnittstelle
9:  kontinuierlich ausgegeben. Die Ausgabe erfolgt immer nach der Erfassung
10:  des 4. Messkanals. Bei 18-Bit Erfassung ist die Ausgaberate etwas mehr als
11:  eine Sekunde. Bei 16-Bit Erfassung erfolgt die Ausgabe etwa alle 260 ms.
12:  Die Ausgabe der Messwerte erfolgt in mV.
13:
14:  Einstellung der Schnittstelle:
15:  9600 Baud
16:  8 Bit Daten
17:  1 Startbit
18:  1 Stoppbit
19:  no Parity
20:  no Handshake
21:
22:  Datenformat:
23:  Wert1\Wert2\Wert3\Wert4 CRLF
24:  Beispiel:
25:  1238.03241\-1000.68321\245.30581\-1675.45902 CRLF
26:
27:  Es werden immer alle 4 Kanäle ausgegeben, auch wenn z.B. nur ein Kanal
28:  verwendet wird. Die Ausgabe erfolgt im ASCII Format. Die Integer-Messwerte
29:  werden in Strings umgewandelt.
30:
31:  Der Datenlogger kann via Terminal-Programm ausgelesen und die Messwerte
32:  dann in einer CSV-Datei gespeichert werden. Eine Weiterverarbeitung
33:  - z.B. in EXCEL - ist damit möglich. Als Trennzeichen zwischen den Werten
34:  wird ein Backslash "\" ausgegeben, nach dem vierten Wert ein CRLF.
35:
36:  Das Datenformat ist kompatibel zur ABACOM-Software "RealView". Mit dieser
37:  Software kann der Datenlogger komfortabel ausgewertet werden. Das Gerät muss
38:  hier dann als "User Interface" definiert werden.
39:
40:
41:           Author:       Hans-Peter Prast, DL2KHP
42:           Date:         30.09.2016
43:           Controller:   ATMega32
44:           Clock:        7,3728 MHz (externer Quarz)
45:           Version:      1.0
46:
47:  Letzte Änderungen:
48:           30.09.2016 Programmcode erstellt
49:           03.10.2016 Programmcode erstellt
50:           11.10.2016 Programmcode fertiggestellt
51:           20.10.2016 Debuging
52:           22.10.2016 Negative Zahlendarstellung eingebaut
53:           23.10.2016 Ausgabe Dezimalpunkt gegen Komma ausgetauscht
54:           27.10.2016 Kommentare ergänzt
55: }
56:
57:
58: // Compiler-Switch ------------------------------------------
59: // aktivieren, wenn 18-Bit-ADC MCP3424 eingesetzt wird
60: {$define 18Bit}
61: // Auskommentieren, wenn der 16-Bit-ADC MCP3428 eingesetzt wird
62: //-------------------------------------------------------------
```

```
63: Const
64: // Ports initialisieren ------------------------------------
65:    k_ddrb = %00000000;          // Datenrichtung Port B
66:    k_ddrc = %00000000;          // Datenrichtung Port C
67:    k_ddrd = %00000000;          // Datenrichtung Port D
68:    k_portb = %11111111;         // Pullup-Widerstände einschalten
69:    k_portc = %11111111;         // Pullup-Widerstände einschalten
70:    k_portd = %11111111;         // Pullup-Widerstände einschalten
71: // Timer 0 initialisieren -----------------------------------
72:    k_tccr0a = %00000010;        // CTC-Mode, 7,3728 MHz, 64
73:    k_tccr0b = %00000011;
74:    k_ocr0 = 57;                 // Compare Register für ca. 1 ms laden
75:    k_ociea = 1;                 // OCIEA-Bit im TIMSK0-Register
76: // TWI Interface, ADC Adresse %1101000x ---------------------
77:    k_adress_rd = %11010001;     // Bausteinadresse und Lesen (0xD1)
78:    k_adress_wr = %11010000;     // Bausteinadresse und Schreiben (0xD0)
79: // ADC-Parameter --------------------------------------------
80: {$ifdef 18Bit}
81: // ADC-Kanalauswahl MCP3424 ----------------------------------
82:    k_kanal_1 = %00011100;       // Config-Byte Kanal 1, 18 Bit
83:    k_kanal_2 = %00111100;       // Config-Byte Kanal 2, 18 Bit
84:    k_kanal_3 = %01011100;       // Config-Byte Kanal 3, 18 Bit
85:    k_kanal_4 = %01111100;       // Config-Byte Kanal 4, 18 Bit
86: {$else}
87: // ADC-Kanalauswahl MCP3428 ----------------------------------
88:    k_kanal_1 = %00011000;       // Config-Byte Kanal 1, 16 Bit
89:    k_kanal_2 = %00111000;       // Config-Byte Kanal 2, 16 Bit
90:    k_kanal_3 = %01011000;       // Config-Byte Kanal 3, 16 Bit
91:    k_kanal_4 = %01111000;       // Config-Byte Kanal 4, 16 Bit
92: {$endif}
93: // User Flag Register ---------------------------------------
94:    b_neu = 1;                   // 1 = Neuer Messwert vorhanden
95:
96: Var
97:    v_kanal,                              // Aktiver Messkanal
98:    v_timer1 :  byte;                     // User Timer 1, 1ms
99:
100:   v_config : array [1..4] of byte;      // Config. Byte für Kanalwahl
101:
102: {$ifdef 18Bit}
103:   v_messwert : array [1..4] of longint; // 18-BitMesswerte der Messeingänge
104: {$else}
105:   v_messwert : array [1..4] of integer; // 16-Messwerte der Messeingänge
106: {$endif}
107:
108:   v_ufr : array [1..10] of boolean;     // User Flag Register
109:   v_text : array [1..4] of string[30];  // ASCII-Zeichen Ausgabe
110: // Ende Deklarationen ---------------------------------------
111:
112: // Interrupt Routinen ----------------------------------------
113: // Messwerte aus externem ADC lesen -------------------------
114: procedure p_getdata;          // Wird im Timer 1 Interrupt aufgerufen (jede ms)
115: var
116:   v_controll : byte;          // Variable beinhaltet das Controllbyte des ADC
117: {$ifdef 18Bit}
118:   v_messw_lokal : longint;    // Hilfsvariable zum Messwert einlesen, 18-Bit
119: {$else}
120:   v_messw_lokal : integer;    // Hilfsvariable zum Messwert einlesen, 16-Bit
121: {$endif}
122: begin
123:   TWI_Start();                // send TWI start signal
124:   TWI_Write(k_adress_rd);     // send device address + Read, (Adr ADC(1101000x)
```

```
125: {$ifdef 18Bit}
126:   highest(v_messw_lokal) := 0;            // Highest Byte immer löschen
127:   higher(v_messw_lokal) := TWI_Read(1);   // Lesen mit ACK senden
128: {$endif}
129:   hi(v_messw_lokal) := TWI_Read(1);       // Lesen mit ACK senden
130:   lo(v_messw_lokal) := TWI_Read(1);       // Lesen mit ACK senden
131:   v_controll := TWI_Read(0);              // Lesen mit NAK senden
132:   TWI_Stop();                             // send stop signal
133:   v_controll := (v_controll and 0x80); // Testen ob neuer Messwert
134:   if v_controll = 0 then      // Conversion abgeschlossen? (Bit 7 = 0)
135:     begin
136:       v_messwert[v_kanal] := v_messw_lokal;  // Messwert übernehmen
137:       inc(v_kanal);           // bei gültigem Messwert Kanal incrementieren
138:       if v_kanal > 4 then
139:         begin
140:           v_kanal := 1;             // Max. 4 Kanäle vorhanden
141:           v_ufr[b_neu] := 1;        // Flag für neue Messwerte setzen
142:         end;
143:       TWI_Start();                  // send TWI start signal
144:       TWI_Write(k_adress_wr);       // send device address + write
145:       TWI_Write(v_config[v_kanal]); // neuen Messkanal an ADC senden
146:       TWI_Stop();                   // send stop signal
147:     end;
148: end;  // p_getdata
149:
150: // User-Timer v_timer1 ----------------------------------------
151: procedure oc0int(); org 0x001c;         // Timer 0, Compare Match A Interrupt
152: begin
153:   p_getdata;                            // Messwerte aus externem DAC lesen
154: end;
155: // Ende Interrupt Routinen -----------------------------------
156:
157: // Unterprogrammteile ----------------------------------------
158: procedure p_init;               // Datenlogger initialisieren
159: var
160:   i : byte;                     // lokale Laufvariable
161: begin
162: // Ports initialisieren --------------------------------------
163:   ddrb := k_ddrb;               // Kein Port in Benutzung
164:   ddrc := k_ddrc;               // Alles als Input deklarieren
165:   ddrd := k_ddrd;
166:   portb := k_portb;             // Pullup Widerstände Einschalten
167:   portc := k_portb;             // Pullup Widerstände Einschalten
168:   portd := k_portb;             // Pullup Widerstände Einschalten
169: // Timer 0 initialisieren -----------------------------------
170:   tccr0a := k_tccr0a;           // Timer Controll Register A laden
171:   tccr0b := k_tccr0b;           // Timer Controll Register B laden
172:   ocr0a := k_ocr0;              // Output Compare Register laden
173:   timsk0.k_ociea := 1;          // Timer Interrupt freigeben
174: // TWI Interface initialisieren -----------------------------
175:   twi_init(100000);             // TWI-Interface initialisieren, 100K
176: // UART Interface initialisieren ----------------------------
177:   UART1_Init(9600);             // UART initialisieren, 9600 Baud
178: // Variablen initialisieren ---------------------------------
179:   v_config[1] := k_kanal_1;     // Array Controllbytes laden
180:   v_config[2] := k_kanal_2;     // 18Bit, PGA = 1, Continue-Mode
181:   v_config[3] := k_kanal_3;
182:   v_config[4] := k_kanal_4;
183:   for i := 1 to 10 do
184:     v_ufr[i] := 0;              // User Flagregister löschen
185:   for i := 1 to 4 do
186:     v_messwert[i] := 0;         // Messwert löschen
```

```
187:   v_kanal := 1;                  // Messkanal auf 1 setzen
188:   SREG_I_Bit := 1;               // Globel Interrupt freigeben
189: end; // p_init
190:
191:
192: // Messwerte an Schnittstelle ausgeben ----------------------
193: procedure p_writedata;
194: var
195:   ex : boolean;                    // Flag für Stringende erreicht
196:   i, x : byte;                     // lokale Laufvariablen
197:   Sign : Char;                     // Variable für die Zeichenausgabe
198: begin
199:   if  v_ufr[b_neu] = 1 then        // Wenn neue Messwerte vorliegen: senden
200:     begin
201:       i := 1;                                 // Ausgabe auf 1. Messwert
202:       repeat                                  // Messwert 1 bis 4 ausgeben
203:         x := 0;                               // Zeichenzähler auf 0
204:         ex := 0;                              // Stringende-Flag löschen
205:           repeat
206:             if (UCSR0A.UDRE0 = 1) then      // warten auf Sendepuffer leer
207:               begin
208:                 sign := v_text[i][x];       // Zeichen in Char-Variable einlesen
209:                 if sign = '.' then          // Dezimalpunkt gegen Komma tauschen
210:                   sign := ',';
211:                 UDR0 :=(sign);              // Zeichen ausgeben
212:                 if (sign = '\') or (sign = char(10)) then // Test auf Ende
213:                   ex := 1;                  // Stringende erreicht, Abbruch
214:                 inc(x);                     // Zeichenzähler incrementieren
215:               end;
216:           until (ex = 1);                   // Stringende erreicht
217:           inc(i);                           // Ausgabekanal incrementieren
218:       until (i > 4);                        // Kanal 1 bis 4 ausgegeben
219:       v_ufr[b_neu] := 0;                    // Messwert neu Flag wieder löschen
220:     end;
221: end; // p_writedata
222:
223: // Messwerte in ASCII-Zeichen umwandeln -----------------
224: procedure p_calc;
225: var
226:   i,                               // Lokale Laufvariable
227:   y  : byte;                       // Zwischenvariable für 16-Bit Berechnung
228:   zwi : array [1..4] of real;      // Zwischenvariable für Umwandlung in String
229: begin
230:   if v_ufr[b_neu] = 1 then         // Wenn neue Messwerte vorliegen: berechnen
231:     begin
232:       for i  := 1 to 4 do
233:         begin
234: {$ifdef 18Bit}
235: // Wenn Integer Variable negativ, Leerbits setzen + Zweierkomplement berechnen
236: // MSB-ADC und MSB-Variable nicht identisch, Leerbits mit "1" füllen
237:           if (higher(v_messwert[i]) >= 2) then  // MSB (Bit 18) Wandler testen
238:             begin
239:               higher(v_messwert[i]) := (higher(v_messwert[i]) or %11111110);
240:               highest(v_messwert[i]) := %11111111;
241:               v_messwert[i] := ((v_messwert[i] * -1)+1); // Zweierkomplement
242:               v_messwert[i] := (v_messwert[i] * -1); // - Vorzeichen einfügen
243:             end;
244: // Ende Zweierkomplement für Negativwerte --------------------------------
245: // 18-Bit Messwert in mV berechnen und in String ablegen
246:           zwi[i] := (v_messwert[i] * 0.015625); // 18Bit-Messwert in mV wandeln
247:           FloatToStr(zwi[i], v_text[i]);        // Messwert in Text wandeln
248: {$else}
```

```
249: // Wenn Integer Variable negativ, Zweierkomplement berechnen
250:           y := hi(v_messwert[i]);             // High-Byte in Zwischenvariable
251:           if (y >= 0x8000) then               // MSB (Bit 16) Wandler testen
252:             begin
253:               v_messwert[i] := ((v_messwert[i] * -1)+1); // Zweierkomplement
254:               v_messwert[i] := (v_messwert[i] * -1); // - Vorzeichen einfügen
255:             end;
256: // Ende Zweierkomplement für Negativwerte ---------------------------------
257: // 16-Bit Messwert in mV berechnen und in String ablegen
258:           zwi[i] := (v_messwert[i] * 0.0625);  // 16Bit Messwert in mV wandeln
259:           FloatToStr(zwi[i], v_text[i]);       // Messwert in Text wandeln
260: {$endif}
261:         end;
262: // Trennzeichen zwischen den Messwerten einfügen, Kanal 1-3 "\", Kanal 4 CRLF
263:       for i := 1 to 3 do
264:         v_text[i] := v_text[i]+'\';  // bei Kanal 1 - 3 Trennzeichen anfügen
265:       v_text[4] := v_text[4]+char(13)+char(10);  // bei Kanal 4 CR/LF anfügen
266:     end;
267: end; // p_calc
268:
269: // Externen ADC starten --------------------------------
270: procedure p_start;
271: begin                              // Externen ADC mit Kanal 1 starten
272:   TWI_Start();                     // send TWI start signal
273:   TWI_Write(k_adress_wr);          // send device address + write
274:   TWI_Write(v_config[v_kanal]); // mit Messkanal 1 ADC starten
275:   TWI_Stop();                      // send stop signal
276: end; // p_start
277:
278: // main programm ---------------------------------------
279: begin
280:   p_init;                          // Datenlogger initialisieren
281:   p_start;                         // Externen ADC starten
282:   while true do
283:     begin
284:       p_calc;                      // Messwerte in ASCII-String umwandeln
285:       p_writedata;                 // Messwerte ausgeben
286:       delay_ms(1);                 // 1 ms Pause
287:     end;
288: end. // main program
```