Akkulader

Mikroprozessor gesteuertes Ladegerät für Klein-Akkus

Gerhard Schmidt, Mai 2005

Letzte Änderung: 21.8.2005

info@avr-asm-tutorial.net

Überblick

Der Akkulader lädt maximal vier kleine Akkus einzeln mit wählbaren Strömen zwischen 5 mA und 220 mA (max. bis ca. 350 mA in einem einzelnen Kanal). Das Laden und alle Einstellungen dazu erfolgt individuell für jeden einzelnen Akku, gemischte Bestückung ist daher ohne Einschränkung möglich. Der Ladevorgang jedes einzelnen der maximal vier Akkus wird vorgewählt und von einem Mikroprozessor überwacht, der Ströme und Ladekapazitäten misst und korrigiert. Die Bedienung des Geräts erfolgt entweder über die serielle Schnittstelle eines Rechners oder über drei Tasten. Die Bedienung und die Ausgabe der aktuellen Werte erfolgt über eine vierzeilige LCD-Anzeige.

Funktionsweise

Das Gerät wird durch einen ATMEL-AVR ATmega16 mit einem internen Takt von 8 MHz gesteuert. An den Prozessor sind angeschlossen:

- vier pulsweiten-getaktete Ausgänge für die Steuerung des Ladestroms,
- vier Ausgänge zur Ansteuerung des Entladevorgangs,
- vier AD-Wandler-Kanäle zur Überwachung der Spannung der vier Akkus,
- vier AD-Wandler-Kanäle zur Überwachung des Stroms beim Entladen und Laden der vier Akkus,
- drei Eingänge für die Bedientasten, teilweise kombiniert mit vier LED-Ausgängen zur Anzeige des Ladezustands,
- zwei serielle Ports für die Kommunikation über die RS232-Schnittstelle,
- 10 Port-Ein- und Ausgänge für die Ansteuerung der LCD-Anzeige, kombiniert mit
- vier Ein-/Ausgängen für die Programmierschnittstelle zur In-System-Programmierung des Prozessors.

Ladevorgang

Das Laden der Akkus erfolgt über jeweils einen Port-Ausgang des Mikroprozessors, der pulsweitengesteuert eine programmierbare Spannung zwischen 0 und 5 V erzeugt. Die Spannung treibt über einen Kleinsignal-Darlington in Emitterschaltung und über einen Strombegrenzer-Widerstand (3,3 Ohm) den Ladestrom. Die Spannung am Akku und am Emitter werden mittels AD-Wandler gemessen, aus der Spannungsdifferenz wird der Strom errechnet, mit dem eingestellten Sollwert verglichen und die Ausgangsspannung der pulsweitengesteuerten Ansteuerung entsprechend verändert.

Beim Laden wird aus dem gemessenen Strom laufend die erreichte Ladekapazität in mAh errechnet. Bei Erreichen des eingestellten Sollwertes der Kapazität wird der Ladestrom auf einen voreingestellten Wert für die Erhaltungsladung verringert.

Entladevorgang

Dem Ladevorgang kann ein kontrolliertes Entladen des Akkus vorgeschaltet werden. Dazu wird über einen Portausgang der entsprechende Akku über den 3,3 Ohm Strombegrenzungswiderstand und die Collector-Emitterstrecke eines Darlington-Transistors entladen. Der Entladestrom wird dabei gemessen, die Entladekapazität laufend errechnet. Bei Unterschreiten einer vorgewählten Akkuspannung wird der Entladevorgang beendet und das Laden eingeleitet.

Akkugeschichte

Der Prozessor verwaltet die Ladecharakteristika und die Anzahl Ladevorgänge für maximal 32 verschiedene individuelle Akkus in seinem EEPROM-Speicher. Nach Vorwahl eines gespeicherten Akkus wird automatisch die Ladekapazität, der Ladestrom und der Erhaltungsstrom eingestellt, das Laden kann sofort gestartet werden. Die aktuellen Werte aller gespeicherten Akkus können als Liste über die serielle Schnittstelle ausgegeben werden. Die folgende Liste wird dabei erzeugt und im Terminalprogramm ausgegeben (Ax> ist der von Ladegerät ausgegebenen Cursor, Kanal 4 ist gerade eingestellt):

```
A4>list
Id Sz Capac Loads _Rest Text____
 1 A3   500     4     0 ConradMH
 2 A3   500     4     0 ConradMH
 3 A3   500     4     0 ConradMH
 4 A3   500     4     0 ConradMH
 5 A2  1800    10     0 PixcelMH
 6 A2  1800    10     0 PixcelMH
 7 A2  1800    10     0 PixcelMH
 8 A2  1800    10     0 PixcelMH
 9 A2  1700     6     0 VoltcrMH
10 A2  1700     6     0 VoltcrMH
11 A2  1700     6     0 VoltcrMH
12 A2  1700     6     0 VoltcrMH
A4>
```

Mit der seriellen Schnittstelle können auch neue Akkus mit wählbaren Parametern erzeugt werden.

Einstellung über serielle Schnittstelle

Der Akkulader kann vollständig über die serielle Schnittstelle gesteuert werden. Die folgenden Befehle stehen zur Verfügung:

<a=x> wählt den Kanal x (1..4) für die folgenden Operationen aus

<n=x> wählt den Akku mit der Id x (1..32) aus und stellt Defaultwerte für diesen Akku ein

<u=x> stellt die Abschaltrestspannung x in mV beim Entladen des Akkus ein

<i=x> stellt den Ladestrom x in mA beim Laden des Akkus ein

<c=x> stellt die Ladekapazität in mAh für die Beendigung des Ladens ein

<m=x> stellt den Erhaltungsstrom x in mA nach Beenden des Ladens ein

<clear> beendet Entladen, Laden und Erhaltungsladen und trennt den Akku

<unload> startet den Entladevorgang (Übergang zum Laden erfolgt automatisch)

<load> startet den Ladevorgang

<maintain> startet das Erhaltungsladen des Akkus

<monitor> gibt jede Minute die Daten in allen aktiven Kanälen aus

<pwm=x> setzt den Pulsweitenmodulator auf den Wert x (0..255)

<new=capacity,size,loads,rest,"text"> erzeugt eine neue Akku-Id, capacity ist die Nennkapazität des Akkus in mAh, size ist das Format des Akkus (0..3), loads ist die Anzahl der Ladevorgänge mit der Nennkapazität (0..65535), rest ist die verbleibende Teilkapazität aus dem letzten Ladevorgang (0..65535), "text" ist ein Erkennungstext (max.8 Buchstaben)

<list> gibt alle gespeicherten Akku-Id's aus,

<restart> startet den Prozessor neu (Software-Reset)

Einstellung über Tasten

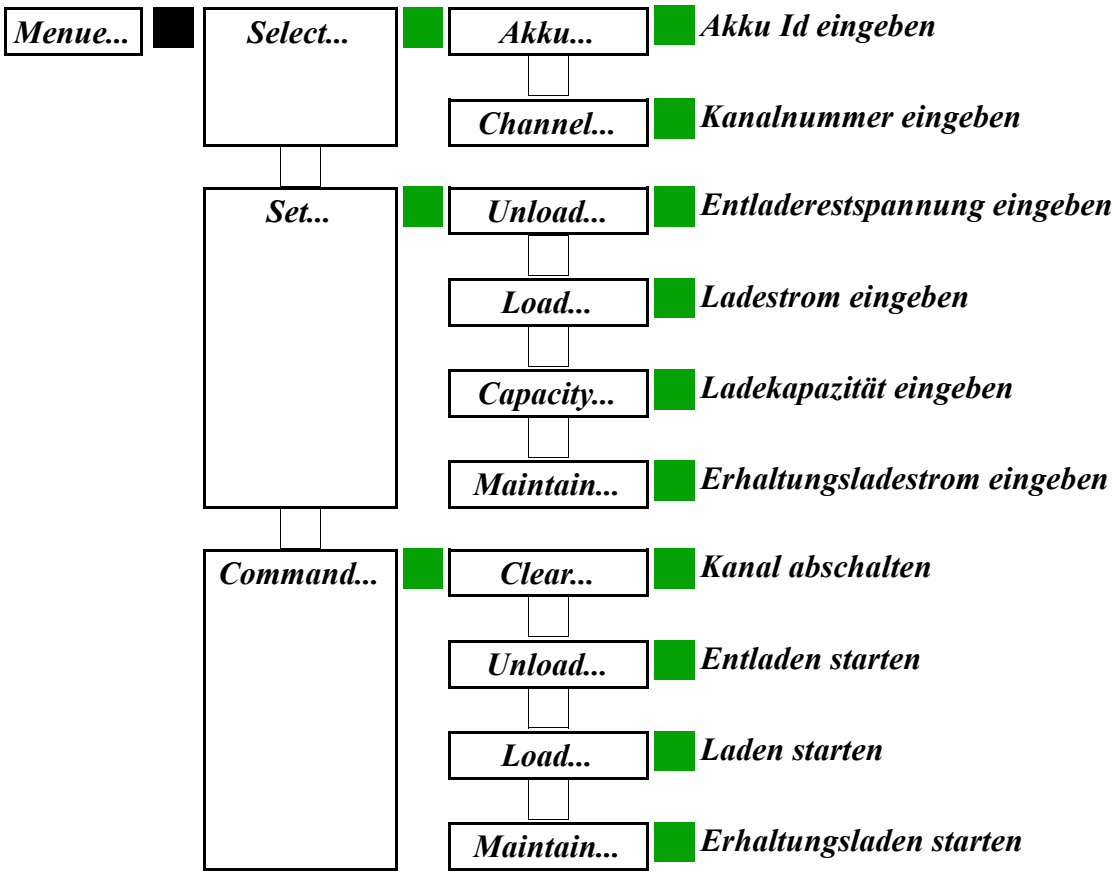
Der Akkulader lässt sich auch ohne Rechnerunterstützung über drei Tasten bedienen (schwarz/links, weiß/Mitte, grün/rechts). Die Tasten bedeuten:

schwarz: Menue einschalten, in Menues: zurück zur letzten Ebene, bei Dezimalzahleneingaben: eine Stelle links bzw. zurück ohne Änderung

weiß: eine Ebene höher, in Menues: nächster Menuepunkt, bei Dezimaleingaben: Ziffer erhöhen

grün: akzeptieren, in Menues: dargestellten Menuepunkt auswählen, bei Dezimaleingaben: nächste Ziffer bzw. Wert übernehmen

Das Menue hat folgende Struktur:



Die Eingabe von Zahlen verwendet die drei Tasten folgendermaßen:

- Die weiße Taste erhöht die Ziffer, bei der der Cursor der LCD blinkt, um Eins.
- Die schwarze Taste setzt den Cursor um eine Ziffer nach links. Beim Verlassen des Eingabebereichs wird die Zahleneingabe abgebrochen, der Wert wird unverändert beibehalten.
- Die grüne Taste setzt den Cursor um eine Ziffer nach rechts. Beim Verlassen des Eingabebereichs wird die Zahl gespeichert und der Zahleneingabe-Modus verlassen.

Über die Tastatur kann keine neue Akku-Id erzeugt werden, dies geht nur über die serielle Schnittstelle oder durch Neuprogrammierung des Mikrocontrollers.

Ausgaben auf der LCD-Anzeige

Die LCD-Anzeige hat vier Zeilen zu je 20 Zeichen.

Zeile 1: Akkuspannung der vier Kanäle

Zeile 2: Lade- bzw. Entladeströme der vier Kanäle in mA (Entladestrom mit negativer Anzeige)

Zeile 3: Kapazitäten der vier Kanäle in mAh

Zeile 4: Zeit seit Einschalten (Tage:Stunden:Minuten), Menuepunkte

LED-Anzeigen

Vier grüne LED zeigen den derzeitigen Zustand an:

LED aus: Kanal ist abgeschaltet

LED blinkt kurz im Sekundenrythmus: Akku wird entladen

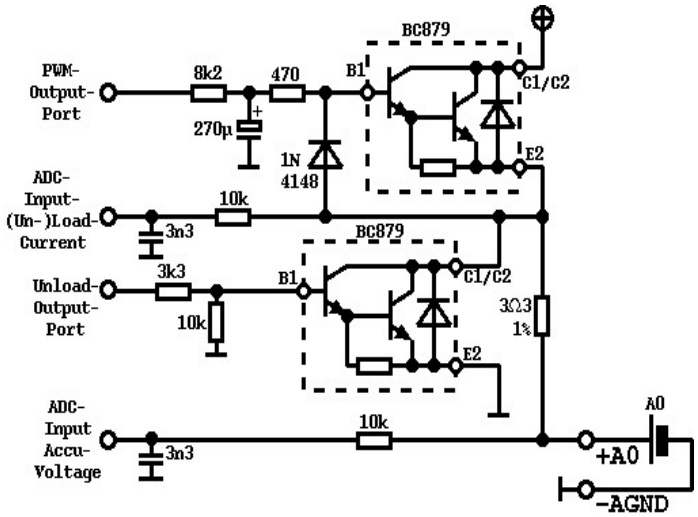
LED blinkt halb im Sekundenrrythmus: Akku wird geladen

LED ist dauerhaft an: Entladen beendet, Erhaltungsladen

Elektrische Funktionsweise

Ladesteuerung

Die Ladesteuerung besteht pro Kanal aus der folgenden Schaltung:



Ein PWM-Output-Port erzeugt eine Rechteckspannung von 31,25 kHz (8 MHz/256), die mit ihrer Pulsweite die Spannung am Elko 270µF bestimmt. Dieser treibt über einen Widerstand von 470 Ohm den oberen Darlington BC879. Dieser lädt über einen Emitterwiderstand von 3,3 Ohm den Akku mit dem voreingestellten Strom. Der Strom wird überwacht. Ist er zu klein, wird die Pulsweite verlängert, ist er zu hoch, wird die Pulsweite verkürzt.

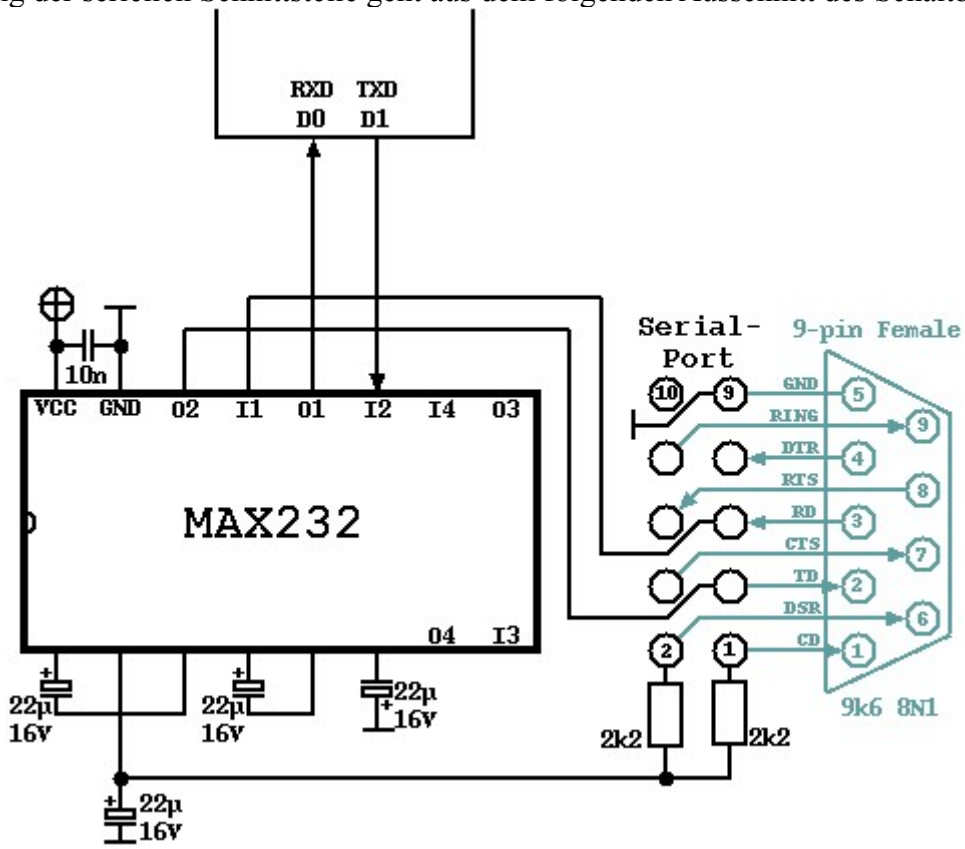
Beim Entladen ist der PWM-Output-Port abgeschaltet und der Unload-Output-Port eingeschaltet. Dieser steuert den unteren Darlington BC879 durch. Der Akku entlädt sich über den Widerstand von 3,3 Ohm und die Kollektor-Emitterstrecke des unteren Darlingtons.

Die Spannung am Akku und die Spannung am 3,3 Ohm-Widerstand wird beim Laden und beim Entladen von je einem ADC-Kanal gemessen. Die Spannungen werden über einen Widerstand von 10 kOhm dem ADC zugeführt und mit einem Kondensator von 3,3 nF gesiebt. Die Siebung ist erforderlich, da der Darlington in Emitterschaltung hochfrequentes Rauschen verursacht, was die Messungen verfälschen würde.

Bei angeschlossenem Akku und abgeschalteter Betriebsspannung dienen die Diode 1N4148 in Kombination mit dem Widerstand von 470 Ohm sowie die beiden 10 kOhm-Widerstände dem Schutz der Mikroprozesser-Ports.

Serielle Schnittstelle

Die Schaltung der seriellen Schnittstelle geht aus dem folgenden Ausschnitt des Schaltbilds hervor.



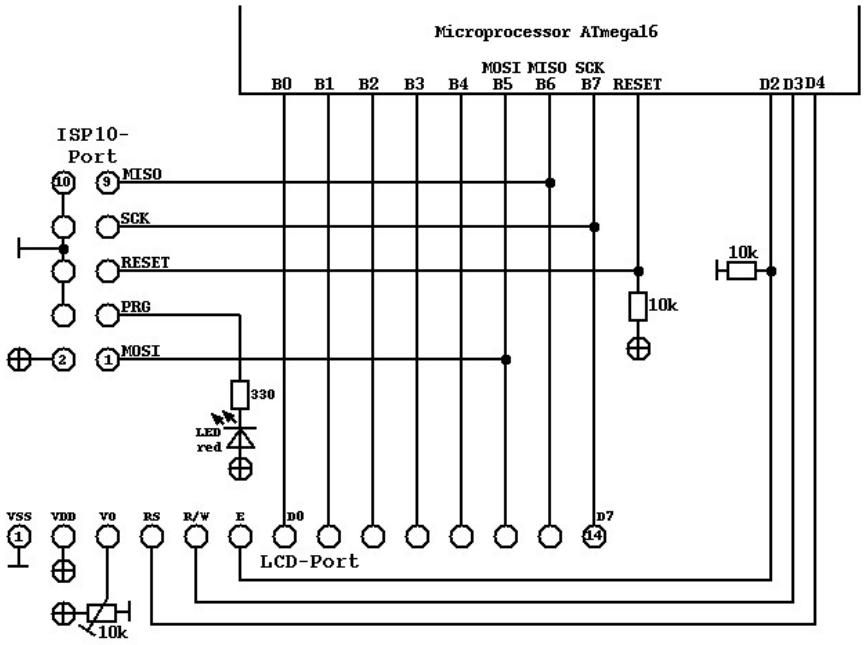
Der MAX232 dient der Spannungserzeugung und zur Umsetzung der seriellen Signale von TTL-Norm auf RS232-Pegel und umgekehrt. Die 9-polige Buchse ist bei den Signalen TD/RD und RTS/CTS (nicht verwendet) gekreuzt geschaltet, so dass ein Parallelkabel verwendet werden kann. Die Pin-Bezeichnungen an der Buchse sind entsprechend vertauscht.[1]

ISP- und LCD-Schnittstelle

Der Bildausschnitt stellt die In-System-Programmierschnittstelle und den LCD-Anschluss dar. Beide Schnittstellen verwenden teilweise diesselben Portanschlüsse des Mikroprozessors.

Der ISP-Port ist an die Ports B5 bis B7 angeschlossen und steuert den Reset-Eingang des Prozessors. Er ist an einer 10-poligen Standard-Pfostenleiste zugänglich.

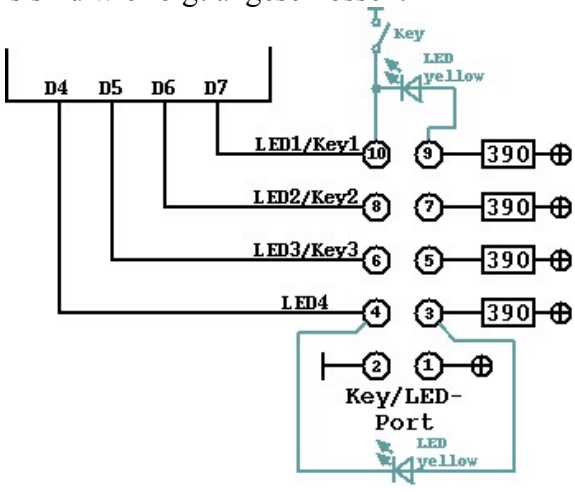
1 In der ersten Version waren RD und TD nicht gekreuzt an den MAX232 geführt, so dass ein gekreuztes Kabel verwendet werden musste.



Der LCD-Port arbeitet mit dem Port B (Daten) und den Portbits D2..D4 für die Kontrolleingänge der LCD-Ausgabe. Der Eingang E an D2 ist zusätzlich mit 10k inaktiviert, wenn der Mikroprozessor die Schnittstelle nicht ansteuert (beim Programmieren, beim Initialisieren der Hardware), damit die LCD keine Fehlsignale auswertet. Das Poti 10k dient der Einstellung des Kontrasts der LCD-Anzeige.

Schalter und LED-Anschlüsse

Die Tastenschalter und LEDs sind wie folgt angeschlossen:



Verwendet werden die Portbits D4 bis D7. Die vier LEDs werden über Widerstände von 390 Ohm angesteuert und von den Portbits aktiv-low gezogen. Die drei Tasten sind an die Portbits D7 (schwarz), D6 (weiß) und D5 (grün) angeschlossen. Sie ziehen im geschlossenen Zustand die Portbits auf Null.

Bitte beachten: Der LED-Ausgang am Portbit D4 wird parallel für die Steuerung der LCD-Anzeige verwendet. Daher kann an diesen Port keine Taste angeschlossen werden. D4 gibt ausschließlich

den Zustand des Kanals 4 wieder.

Netzteil

Das Netzteil ist nicht dargestellt. Es besteht aus einem Trafo (2*9V/2*0,6A), zwei Dioden 1N4001, einem Elko von 4700µF/16V und einem Spannungsregler 5V/1A. Der Spannungsregler ist auf dem Gehäuseboden aus Alu montiert, um eine bessere Kühlung zu erreichen.
Ebenfalls hier nicht dargestellt ist eine Konstantstromquelle, die aus der ungeregelten Gleichspannung einen Konstantstrom von 200 mA für die Hintergrundbeleuchtung der LCD-Anzeige liefert.
Bei der gleichzeitigen Aufladung von vier Akkus mit je mehr als 200 mA Ladestrom und eingeschalteter Hintergrundbeleuchtung der LCD reicht der angegebene Trafo nicht aus und es sollte eine Nummer größer gewählt werden.

Prozessorteil

Die gesamte Steuerung erfolgt durch einen ATmega16 in einem 40-poligen PDIP-Gehäuse. Der Prozessortakt wird im internen Taktgenerator erzeugt und durch Programmieren der entsprechenden Fuses auf 8 MHz eingestellt.
Der AD-Wandlerteil des Prozessors wird über ein Drossel-Kondensator-Sieb mit 5V versorgt, das die Betriebsspannung zusätzlich siebt.

Software

Abläufe

Alle wesentlichen Abläufe des Programmes sind um den Timer 1 herum aufgebaut und mittels Interrupts gesteuert. Dieser Timer ist durch interne Vorteilung (8) und durch den CTC-Teilermodus (Teiler durch 15625) auf eine Pulsfrequenz von 64 Hz eingestellt. Mit diesem Takt wird das AD-Wandler-Ergebnis ausgelesen und mittels MUX-Programmierung der nächste der insgesamt 8 AD-Wandlerkanäle angewählt. Der Start der nächsten Umwandlung wird rechtzeitig vor dem Auslesen durch einen weiteren Compare-Interrupt des Timers 1 ausgelöst, so dass das Wandler-Ergebnis rechtzeitig vor dem Auslesen verfügbar ist. Pro Sekunde wird daher jeder Kanal 8 mal angesteuert und ausgelesen. Die Messergebnisse der einzelnen Kanäle werden für 4 Sekunden lang aufsummiert (jeweils 32 Messungen pro Kanal) und danach ausgewertet. Das Erreichen der 4-s-Schwelle steuert über ein Flag alle weiteren Auswertungsvorgänge.
Die vier PWM-Kanäle zur Einstellung des Ladestroms werden von Timer 0 gesteuert. Timer 0 wird ohne Vorteilung direkt durch den Prozessortakt (8 MHz) getaktet. Dessen Überlauf-Interrupt nach 256 Takten schaltet die vier PWM-Ausgänge entsprechend ihrem jeweiligen Sollwert. Der Sollwert wird durch Auswertung der Messungen eingestellt und entsprechend verändert.
Der Empfang von Zeichen über das UART erfolgt ebenfalls interrupt-gesteuert. Die eingehenden Zeichen werden über den UART-Transmitter zurückgesendet (Echo) und in einem Zeilenpuffer im SRAM zwischengespeichert. Ist eine ganze Zeile empfangen, wird ein Flagbit gesetzt, das die Auswertung dieser Zeile auslöst. Das Senden von Zeichen erfolgt allerdings polling-gesteuert, nicht über Interrupts.

Messablauf und -auswertung

Wie beschrieben werden 4 Sekunden lang die eingehenden Messwerte gesammelt und addiert. Die Addition findet sofort mit Vorliegen des Messwertes statt, eine Zwischenspeicherung der einzelnen Messwerte erfolgt nicht. Gemessen wird in den geraden Kanälen (0, 2, 4, 6) die Spannung am Akku, in den ungeraden Kanälen (1, 3, 5, 7) die Spannung am 3,3-Ohm-Widerstand der Lade-/Entladeschaltung. Aus der Spannungsdifferenz über dem Widerstand lässt sich der Lade- bzw. Entladestrom errechnen.
Nach Ablauf von 4 Sekunden bzw. von 32 Messungen pro Kanal werden die eingegangenen Messwertsummen in Spannungen umgerechnet. Der AD-Wandler-Wert von 1024 entspricht dabei einer Spannung von 5.000 Volt, die Summe über 32 Messungen entsprechend dem 32-fachen der Spannung (m32s). Die Umrechnung vom Messwert in eine Spannung (in mV) erfolgt nach der folgenden Formel:

$$U(mV)*256=(m32s*256*5000)/(1024*32)=39*m32s$$

Die beiden obersten Bytes des Ergebnisses werden nach Rundung als Spannung in Form einer binären 16-Bit-Zahl gespeichert.
Der Strom wird aus der Differenz der Spannungsmessungen errechnet. Ausgegangen wird von den Messwertsummen über 32 Messungen, die subtrahiert werden (m32d). Nach der folgenden Formel ergibt sich daraus der Strom in mA:

$$I(mA)*256=(5000*m32d*256)/(1024*32*3{,}3)=11{,}84*m32d$$

Die beiden obersten Bytes des Ergebnisses werden gerundet und als Strom in mA in Form einer binären 16-Bit-Zahl gespeichert[1]. Im Fall negativen Stroms (Entladen des Akkus) wird das Komplement der Zahl gespeichert.
Aus dem errechneten Strom wird jeweils die Lade- bzw. Entladekapazität in mAh errechnet, die in den vergangenen 4 Sekunden in den Akku geladen wurde bzw. aus dem Akku entnommen wurde. Sie beträgt:

$$C(mAh)=I(mA)*t(h)=I(mA)*73/65536(h)$$

Zur Berechnung wird für eine höhere Genauigkeit auch der Rest aus der Stromberechnung verwendet (24-Bit-Zahl, MSB:LSB in mA, LLSB in mA/256).
Die errechnete Kapazität wird aufsummiert. Dabei wird zur Vermeidung von Rundungsfehlern eine 24-Bit-Zahl (MSB, LSB und LLSB) verwendet, deren oberste zwei Byte der Kapazität in mAh entsprechen, das niedrigste Byte den Rest als mAh/256 enthält.
Die errechnete Spannung am Akku dient der Erkennung, wann der Entladevorgang beendet ist (Spannung am Akku kleiner als eingestellter Vergleichswert). Die errechneten Ströme dienen durch Vergleich mit dem Sollwert der Steuerung der PWM (beim Laden und bei der Erhaltungsladung). Die errechneten Kapazitäten werden angezeigt (beim Entladen) bzw. dienen der Erkennung, wann der Akku vollgeladen ist (beim Ladevorgang). Die Kapazität wird, umgerechnet als Anzahl der Volladungen mit Nennkapazität, außerdem für jeden Akku dauerhaft im EEPROM gespeichert, so dass die Nutzungsgeschichte des jeweiligen Akkus erkennbar ist.

Ausgaben

Die Ausgabe der Ladedaten der vier Akkukanäle erfolgt alle vier Sekunden auf der LCD-Anzeige. Angezeigt werden

- die Akkuspannung U in mV,
- der Lade-(+) bzw. Entladestrom (-) I in mA,
- die erreichte Kapazität C in mAh.

Außerdem wird minütlich die vergangene Systemzeit seit Einschalten des Geräts in dd:hh:mm

1 In der ersten Version der Software verursachte ein Bug, dass der gemessene Strom um etwa den Faktor 2 überschätzt wurde. Entsprechend betrugen die Ladeströme nur etwa die Hälfte des eingestellten und angezeigten Stroms.

angezeigt. Das folgende Bild zeigt die LCD-Ausgabe beim gleichzeitigen Entladen von vier Akkus.

Die Werte aktivierter Akkukanäle werden auch über das UART an die serielle Schnittstelle ausgegeben, wenn die Ausgabe mit “monitor” auf der Kommandozeile des Terminalprogramms aktiviert wurde[1]. Die Ausgabe der Messwerte im Terminalprogramm erfolgt minütlich und sieht folgendermaßen aus (Kanal=1, Akku-Id N=1, Status S=2 entspricht Laden):

```
T=00:00:05
1: N=1, S=2, 1179mV, 64mA, 0mAh
1: N=1, S=2, 1189mV, 57mA, 1mAh
1: N=1, S=2, 1198mV, 65mA, 2mAh
1: N=1, S=2, 1202mV, 62mA, 3mAh
1: N=1, S=2, 1209mV, 69mA, 4mAh
T=00:00:10
1: N=1, S=2, 1212mV, 61mA, 5mAh
1: N=1, S=2, 1215mV, 61mA, 6mAh
1: N=1, S=2, 1219mV, 61mA, 7mAh
1: N=1, S=2, 1224mV, 66mA, 8mAh
1: N=1, S=2, 1228mV, 65mA, 9mAh
```

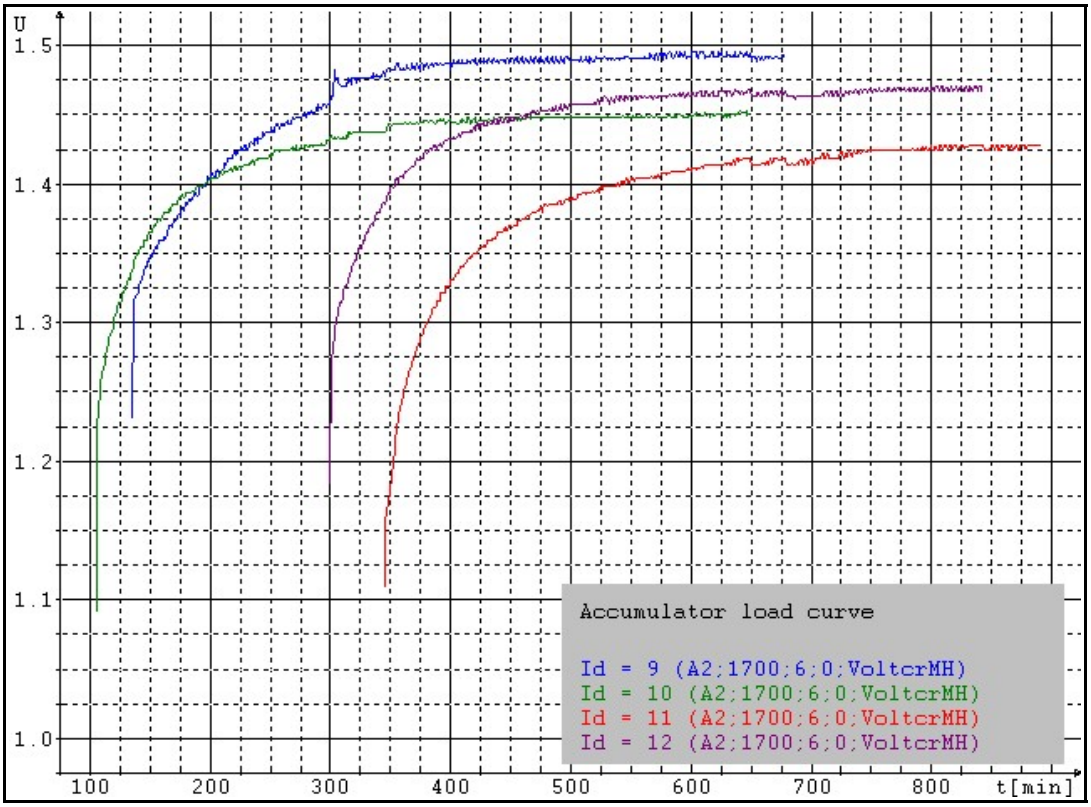
Die Ausgabe der Systemzeit erfolgt beim Monitoring alle 5 Minuten.

Externe Auswertung

Die mitgeschriebenen Werte können extern ausgewertet werden. Das folgende Bild zeigt die Ladekurven von vier gleichzeitig geladenen NiMH-Akkus mit einer Nennkapazität von je 1700 mAh.

Der unterschiedliche Beginn des Ladevorganges ist durch die unterschiedlichen Restkapazitäten der vier Akkuzellen bedingt: alle Akkus wurden zunächst auf 1,0 Volt Zellenspannung entladen, dann erst wird automatisch der Ladevorgang eingeleitet. Die Akkus wurden mit ihrer Nennkapazität +12% geladen, die Endspannungen der Akkus lagen zwischen 1,4 und 1,5 Volt.

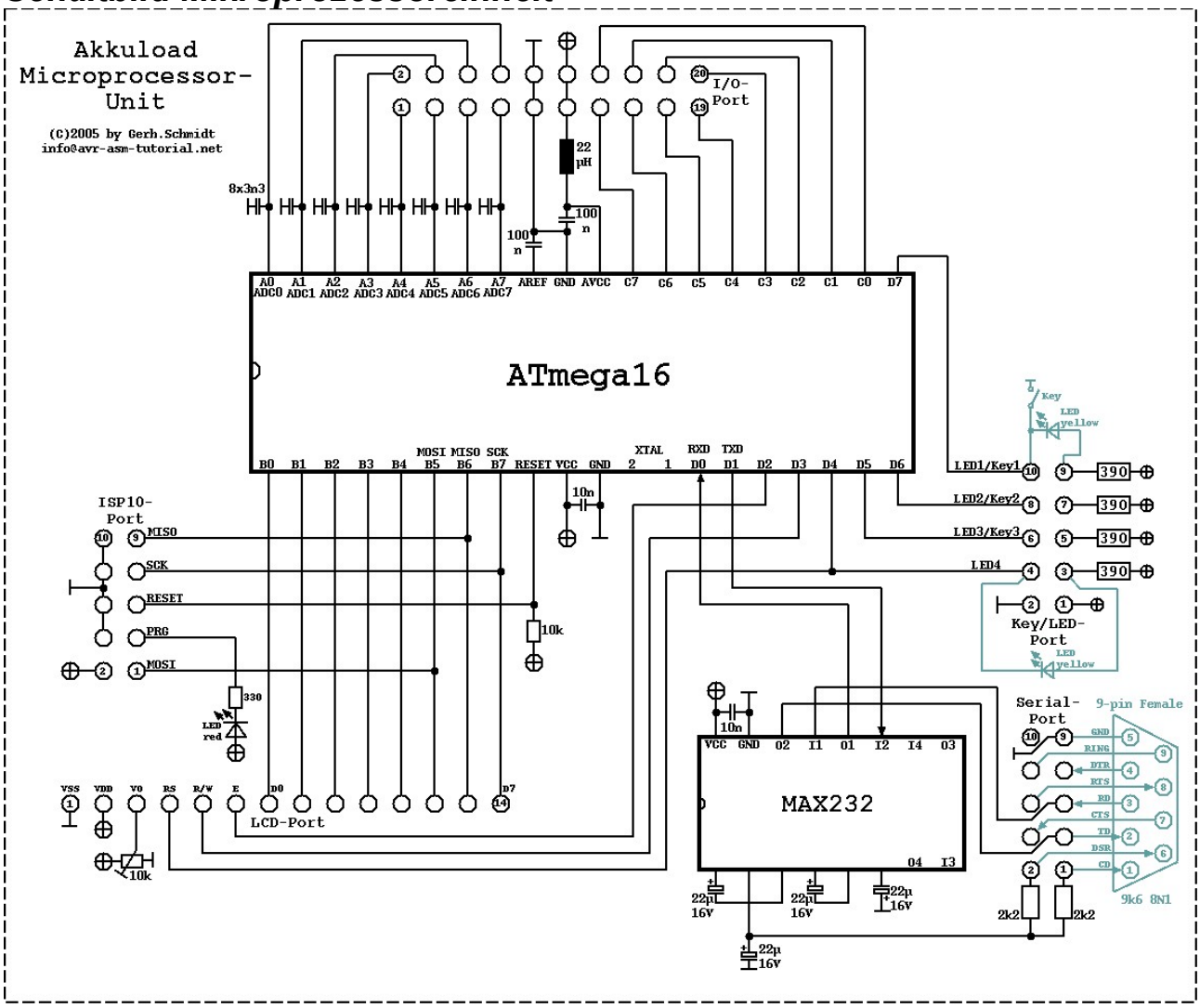
1 In der ersten Version der Software wurde der Zustand “Monitor aus” und “Monitor an” falsch ausgegeben.



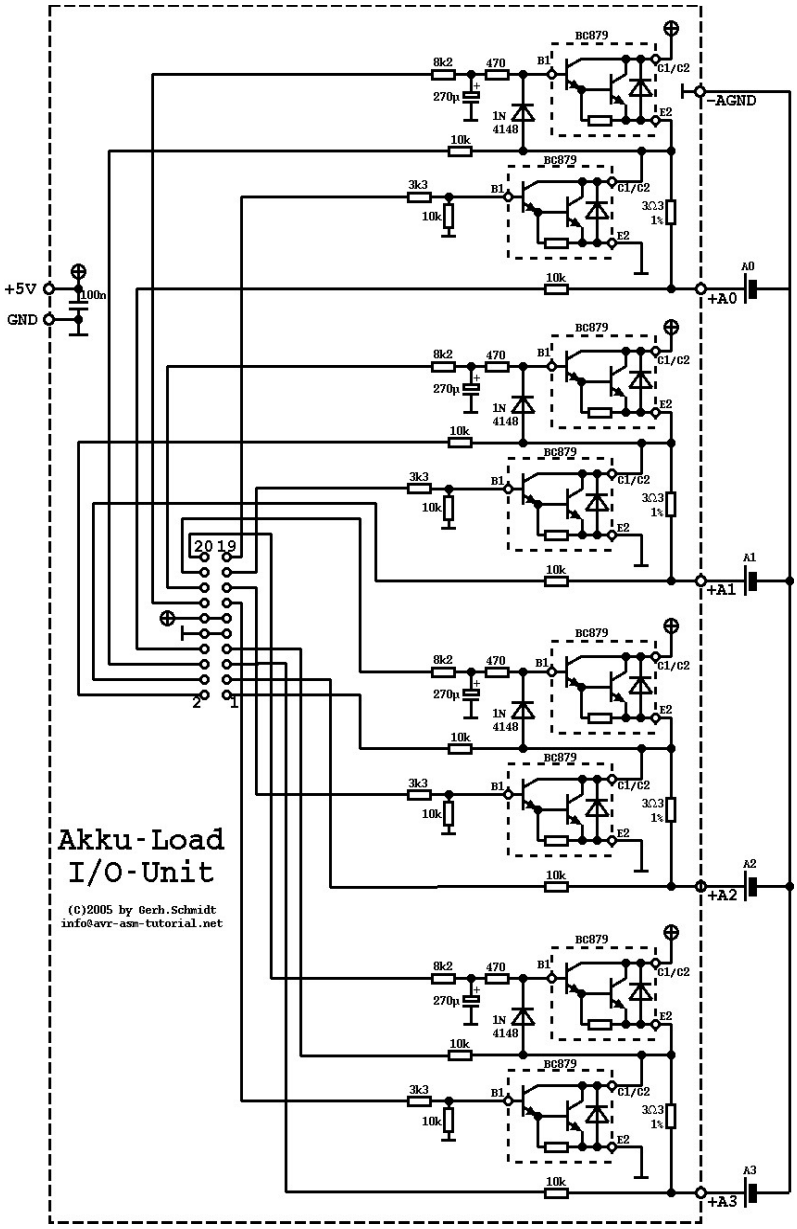
Die Software zur Auswertung ist in Delphi geschrieben und auf Anfrage bei mir erhältlich. Sie erzeugt aus Mitschreibetexten der seriellen Schnittstelle auf Wunsch GIF- oder BMP-Dateien der Ladekurven. Die Skalierung und Beschriftung der Grafiken erfolgt automatisch.

Anhang

Schaltbild Mikroprozessoreinheit



Schaltbild Ladeeinheit



Software (AVR-Assembler)

Akkuload.asm

```
; ************************************************************
; * Akkulader Version 0.1 for ATmega16 @ 8 MHz Takt          *
; * (C)2005 by Gerh.Schmidt info@avr-asm-tutorial.net        *
; ************************************************************
;
.INCLUDE "m16def.inc"
;
; Konstanten
;
.EQU fclock = 8000000 ; Processortaktfrequenz
.EQU baud = 9600 ; Serielle Schnittstelle Baudrate
.EQU cUartTime = 5 ; Alle 5 Minuten die Systemzeit an das UART
.EQU cKeyRep = 5 ; Anzahl Durchläufe bis Tastendruck akzeptiert wird
.EQU cUnlVol = 1000 ; Standard-Entladespannung in mV
;
; Register Definitionen
;
; R0 wird für diverse Operationen verwendet
; R1:R0 Als Doppelregister bei Rechenoperationen verwendet
; R3:R2 dto.
; R6:R5:R4 Dreifachregister bei Rechenoperationen
.DEF rKeyLst = R7 ; Letzte gedrückte Taste
.DEF rKeyRep = R8 ; Zähler für Tastenwiederholung
;.DEF rAdcV  = R9  ; Betriebsspannungsmaske fr ADC
.DEF rPwmC  = R10 ; PWM Zähler
.DEF rPwm1S = R11 ; PWM Schaltwert fr Kanal 1
.EQU cPwm1S = 11  ; (Pointerwert fr Kanal 1)
.DEF rPwm2S = R12 ; PWM Schaltwert fr Kanal 2
.DEF rPwm3S = R13 ; PWM Schaltwert fr Kanal 3
.DEF rPwm4S = R14 ; PWM Schaltwert fr Kanal 4
.DEF rSreg  = R15 ; Sicherung von SREG bei Interrupts
.DEF rmp    = R16 ; Multipurpose Register
.DEF rimp   = R17 ; Multipurpose Register bei Interrupts
.DEF rimp1  = R18 ; Multipurpose Register bei Interrupts
.DEF rFlg   = R19 ; Multipurpose Flag Register (siehe unten)
.DEF rFlgD  = R20 ; Anzeige Flag Register (siehe unten)
.DEF rPwmOut= R21 ; Wert für die PWM-Ausgabe
; Frei R22..R23
.DEF rChCL = R24 ; Kanalzähler für ADC-Messungen, LSB
.DEF rChCH = R25 ; dto., MSB
; X (R27:R26) für verschiedene Zwecke außerhalb von Interrupts
; Y (R29:R28) bei Interrupts für Messauswertung verwendet
; Z (R31:R30) für verschiedene Zwecke außerhalb von Interrupts
;
; Bit-Definitionen für rFlg
.EQU bUartRxLine = 7 ; Eine vollständige Zeile über UART empfangen
.EQU bLcdOk = 6 ; LCD-Anzeige angeschlossen und ok
; nicht benutzt: Bits 3, 4 und 5
.EQU bMin = 2 ; Timer hat 1 Minute erreicht
.EQU b32m = 1 ; Timer Interrupt 4 Secunden, 32 ADC-Messungen komplett
.EQU b64Hz = 0 ; Timer Interrupt 64 Hz Tick
;
; Bit-Definitionen für das Anzeigeflag rFlgD
;
.EQU bUHex = 7 ; Zeige Ergebnisse in Hex ber UART an (Debug)
```

```
.EQU bUMoni = 6 ; Zeige aktive Kanäle über UART an
.EQU bURdy = 5 ; Ergebnisse komplett für die UART-Ausgabe
.EQU bLRdy = 4 ; Ergebnisse komplett für die LCD-Ausgabe
.EQU bLcd3 = 3 ; Zeile 4 der LCD ist frei verwendbar
.EQU bLcd2 = 2 ; Zeile 3 der LCD ist frei verwendbar
.EQU bLcd1 = 1 ; Zeile 2 der LCD ist frei verwendbar
.EQU bLcd0 = 0 ; Zeile 1 der LCD ist frei verwendbar
;
;
; Berechnete Konstanten
;
.EQU cTC1Prsc = 8 ; TC1 Vorteiler-Wert
.EQU cTC1Div = fClock / cTC1Prsc / 64 ; 64 Interrupts pro Sekunde
.EQU cTC1CompB = 50 ; Timer-Wert für den Start der ADC-Umwandlung
.EQU cStartAdc = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);ADC-Enable,Clock=128
.EQU cRestartAdc = cStartAdc | (1<<ADSC) ; Starte ADC-Umwandlung
.EQU cAdcV = 1<<REFS0 ; ADC-Referenzspannung auf 5V
.EQU cEepMax = 512/16 ; Maximale Anzahl gespeicherter Akku-Id's im EEPROM
.EQU cCr = $0D ; Wagenrücklauf-Zeichen für UART
.EQU cLf = $0A ; Zeilenvorschub-Zeichen für UART
;
; SRAM Positionen
;
; UART SRAM Variablen
.EQU sUartRxBp = $0060 ; UART Rx Pufferzeiger
.EQU sUartRxBs = $0061 ; Uart Rx Pufferanfang
.EQU sUartRxBe = $007E ; Uart Rx Pufferende
.EQU sUartCurCh = $007F ; Aktueller Kanal für UART-Ein-/Ausgabe
; Sollwerte der Ladekanäle
.EQU sCh1UV = $0080 ; Kanal 1, Entladespannung Abschaltwert
.EQU sCh1LC = $0082 ; Kanal 1, Ladestrom in mA
.EQU sCh1LW = $0084 ; Kanal 1, Ladekapazität in mAh
.EQU sCh1MC = $0086 ; Kanal 1, Erhaltungsladestrom in mA
.EQU sCh2UV = $0088 ; Kanal 2, dto.
.EQU sCh2LC = $008A ; Kanal 2, dto.
.EQU sCh2LW = $008C ; Kanal 2, dto.
.EQU sCh2MC = $008E ; Kanal 2, dto.
.EQU sCh3UV = $0090 ; Kanal 3, dto.
.EQU sCh3LC = $0092 ; Kanal 3, dto.
.EQU sCh3LW = $0094 ; Kanal 3, dto.
.EQU sCh3MC = $0096 ; Kanal 3, dto.
.EQU sCh4UV = $0098 ; Kanal 4, dto.
.EQU sCh4LC = $009A ; Kanal 4, dto.
.EQU sCh4LW = $009C ; Kanal 4, dto.
.EQU sCh4MC = $009E ; Kanal 4, dto.
; Messwerte vom ADC
.EQU sAdcE1 = $00A0 ; ADC Messergebnisse, geradzahlige Kanäle (Akkus)
.EQU sAdcO1 = $00A8 ; dto., ungeradzahlige Kanäle (Ladewiderstand)
.EQU sAdcE4 = $00B0 ; ADC Summe für 4 Sekunden, geradzahlige Kanäle
.EQU sAdcO4 = $00B8 ; dto., ungeradzahlige Kanäle
.EQU sAdcU  = $00C0 ; berechnete Spannungen, geradzahlige und ungeradzahlige
.EQU sAdcI  = $00D0 ; berechnete Ladeströme in mA
.EQU sAdcC  = $00D8 ; berechnete Kapazitäten in mAh (LSB,MSB)
.EQU sAdcLL1= $00E0 ; berechnete Kapazitäten in mAh (LLSB), Kanal 1
.EQU sAdcAI1= $00E1 ; Akku Id, Kanal 1
; Kanalsteuerung
.EQU sState = $00E8 ; Aktueller Status der Kanäle, Bits 44332211
            ; 00=abgeschaltet, 01:entladen, 10:laden, 11:erhalten
.EQU sTimeM = $00E9 ; Systemzeit, Minuten (0, 15, 30, 45)
.EQU sTimeH = $00EA ; dto., Stunden
```

```
.EQU sTimeD = $00EB ; dto., Tage
.EQU sLcdCs = $00EC ; LCD Cursor Position
; Tasteneingabe
.EQU sKeyNC = $00ED ; Dezimalzahl Eingabeposition
.EQU sKeyNW = $00EE ; Dezimalzahl Ein- und Ausgabewort
.EQU sKeyNM = $00F0 ; Dezimalzahl Maximalwert
.EQU sKeyJR = $00F2 ; Sprungadresse für Dezimalzahleingabeende
.EQU sKeyJS = $00F4 ; Sprungadresse bei Abbruch der Dezimalzahleingabe
.EQU sKeyJ1 = $00F6 ; Sprungadresse bei Taste 1 gedrückt
.EQU sKeyJ2 = $00F8 ; dto., Taste 2
.EQU sKeyJ3 = $00FA ; dto., Taste 3
.EQU sKeyCh = $00FC ; Aktueller Kanal für Tasteneingabe (0, 1..4)
; LED-Ausgabe
.EQU sLedC  = $00FD ; LED-Zähler
.EQU sLed1  = $00FE ; LED Kanal 1 Vergleichswert
.EQU sLed2  = $00FF ; dto., Kanal 2
.EQU sLed3  = $0100 ; dto., Kanal 3
.EQU sLed4  = $0101 ; dto., Kanal 4
; EEPROM-Lese- und Schreibpuffer
.EQU sEepB  = $0102 ; EEPROM Puffer für einen Akku-ID-Record, 16 Bytes
.EQU sEAId  = $0102 ; EEPROM Akku Id
.EQU sESize = $0103 ; EEPROM Akkugröße = A0..A3 (Mignon, Baby, AA, AAA)
.EQU sENCL  = $0104 ; EEPROM Nennkapazität LSB
.EQU sENCM  = $0105 ; dto., MSB
.EQU sENLL  = $0106 ; EEPROM Anzahl der Volladungen, LSB
.EQU sENLM  = $0107 ; dto., MSB
.EQU sENRL  = $0108 ; EEPROM Restkapazität in mAh, LSB
.EQU sENRM  = $0109 ; dto., MSB
.EQU sEATxt = $010A ; Textteil der Akku-Definition
.EQU sENul  = $0112 ; Null-Zeichen als Stopwert
.EQU sEANm  = $0113 ; Anzahl der im EEPROM gespeicherten korrekten IDs
;
; Nächste freie SRAM-Position: $0114
;
;
; Reset- und Interrupt-Vektoren
; (Vektoren beim ATmega16 sind Doppelworte!)
;
.CSEG
.ORG $0000
      rjmp main ; Reset Vektor
      nop
      reti ; INT0 Vektor
      nop
      reti ; INT1 Vektor
      nop
      reti ; TC2Comp Vektor
      nop
      reti ; TC2Ovf Vektor
      nop
      reti ; TC1Capt Vektor
      nop
      rjmp TC1CmpAIsr ; TC1CompA Vektor
      nop
      rjmp TC1CmpBIsr ; TC1CompB Vektor
      nop
      reti ; TC1Ovf Vektor
      nop
      rjmp TC0OvfIsr ; TC0Ovf Vektor
      nop
```

```
      reti ; SPI, STC Vektor
      nop
      rjmp SioRxCIsr; USARTRxC Vektor
      nop
      reti ; USARTUdre Vektor
      nop
      reti ; USARTTxc Vektor
      nop
      reti ; ADC Vektor
      nop
      reti ; EE_RDY Vektor
      nop
      reti ; ANA_COMP Vektor
      nop
      reti ; TWI Vektor
      nop
      reti ; INT2 Vektor
      nop
      reti ; TC0Comp Vektor
      nop
      reti ; SPM_RDY Vektor
      nop
;
; TC1 Compare A Int Serviceroutine
; (Liest das Resultat einer ADC-Messung und speichert es)
;
TC1CmpAIsr:
      in rSreg,SREG ; Sichere SREG
      sbr rFlg,1<<b64Hz ; Setze 64 Tick Flagge
      mov rimp,rChCL ; Prüfe Zähler
      andi rimp,1 ; ungerader oder gerader Kanal?
      brne TC1CmpAIsrO ; ungerader Kanal
      ; Gerader Kanal
      mov rimp,rChCL ; Prfe Zähler
      andi rimp,0xF8 ; Isoliere Zähler ohne die Kanalnummer
      brne TC1CmpAIsrE1 ; Kein Neustart
      ; TC1 hat 32 Messungen beendet, startet neu
      in rimp,ADCL ; Lese ADC LSB
      st Y,rimp ; Speichere im SRAM
      in rimp,ADCH ; Lese ADC MSB
      std Y+1,rimp ; Speichere im SRAM
      rjmp TC1CmpAIsrE3
TC1CmpAIsrE1: ; kein Neustart
      cpi rimp,0xF8 ; letzte Messung im aktuellen Kanal?
      breq TC1CmpAIsrE2 ; Ja, addiere die letzte Messung
      ; Normale Messung, addiere zur Summe
      in rimp1,ADCL ; Lese Ergebnis vom ADC, LSB
      ld rimp,Y ; Lese bisherige Summe aus SRAM
      add rimp,rimp1 ; Addiere Ergebnis
      st Y,rimp ; Speichere im SRAM
      in rimp1,ADCH ; Lese Ergebnis vom ADC, MSB
      ldd rimp,Y+1 ; lese gespeicherte Summe im SRAM, MSB
      adc rimp,rimp1 ; Addiere Messergebnis zur Summe
      std Y+1,rimp ; Speichere MSB im SRAM
      rjmp TC1CmpAIsrE3
TC1CmpAIsrE2: ; die letzte Messung der Messreihe
      in rimp1,ADCL ; Lese Ergebnis vom ADC, LSB
      ld rimp,Y ; Lese gespeicherte Summe, LSB
      add rimp,rimp1 ; Addiere zum Ergebnis
      std Y+16,rimp ; Speichere den Summenwert im SRAM, LSB
```

```
      in rimp1,ADCH ; Lese Ergebnis vom ADC, MSB
      ldd rimp,Y+1 ; Lese gespeicherte Summe, MSB
      adc rimp,rimp1 ; Addiere zum Ergebnis MSB
      std Y+17,rimp ; Speichere den Summenwert im SRAM, MSB
TC1CmpAIsrE3:
      adiw rChCL,1 ; Erhöhe den Kanalzähler
      mov rimp,rChCL ; Kopiere LSB des Kanalzählers
      andi rimp,0x07 ; isoliere Kanalnummer des nächsten Kanals
      ori rimp,cAdcV ; Setze die Referenzspannungsbits
      out ADMUX,rimp ; Wähle den nächsten MUX-Kanal des ADC an
      out SREG,rSreg ; Stelle SREG wieder her
      reti
;
TC1CmpAIsrO: ; ungerader Kanal
      mov rimp,rChCL ; Kopiere LSB Kanalzähler
      andi rimp,0xF8 ; isoliere alles bis auf die Kanalnummer
      brne TC1CmpAIsrO1 ; kein Neustart
      ; TC1 hat 32 Messungen beendet, startet neu
      in rimp,ADCL ; Lese ADC, LSB
      std Y+8,rimp ; Speichere im SRAM (überschreibt alte Summe)
      in rimp,ADCH ; Lese ADC, MSB
      std Y+9,rimp ; Speichere im SRAM (überschreibt alte Summe)
      rjmp TC1CmpAIsrO3
TC1CmpAIsrO1: ; ungerader Kanal, kein Neustart
      cpi rimp,0xF8 ; Letzte Messung im Kanal?
      breq TC1CmpAIsrO2 ; Ja, addiere den letzten Wert
      ; Normale Messung, addiere zur Summe und speichere
      in rimp1,ADCL ; Lese ADC, LSB
      ldd rimp,Y+8 ; Lese gespeicherte Summe aus SRAM, LSB
      add rimp,rimp1 ; addiere zum Ergebnis, LSB
      std Y+8,rimp ; Speichere Summe im SRAM, LSB
      in rimp1,ADCH ; Lese ADC, MSB
      ldd rimp,Y+9 ; Lese gespeicherte Summe aus SRAM, MSB
      adc rimp,rimp1 ; addiere zum Ergebnis, MSB
      std Y+9,rimp ; Speichere Summe im SRAM, MSB
      rjmp TC1CmpAIsrO3
TC1CmpAIsrO2: ; Ungerader Kanal, letzte Messung im Kanal
      in rimp1,ADCL ; Lese ADC, LSB
      ldd rimp,Y+8 ; Lese bisherige Summe, LSB
      add rimp,rimp1 ; Addiere zum Ergebnis, LSB
      std Y+24,rimp ; Speichere das letzte Summenergebnis, LSB
      in rimp1,ADCH ; Lese ADC, MSB
      ldd rimp,Y+9 ; Lese bisherige Summe, MSB
      adc rimp,rimp1 ; addiere zum Ergebnis, MSB
      std Y+25,rimp ; Speichere das letzte Summenergebnis, MSB
TC1CmpAIsrO3:
      adiw YL,2 ; zeige auf nächste Position im SRAM
      adiw rChCL,1 ; nächster Kanal
      mov rimp,rChCL ; Kopiere Z ler, LSB
      andi rimp,0x07 ; Isoliere nächsten Kanal
      brne TC1CmpAIsrO4 ; Nicht Kanal 1
      ldi YH,HIGH(sAdcE1) ; Setze Pointer zurück auf Kanal 1
      ldi YL,LOW(sAdcE1)
TC1CmpAIsrO4:
      ori rimp,cAdcV ; Setze die Referenzspannungs-Bits
      out ADMUX,rimp ; Wähle nächsten ADC-MUX-Kanal
      tst rChCL ; Alles am Anfang?
      brne TC1CmpAIsrO5 ; nein
      sbr rFlg,1<<b32m ; Setze die Flagge für 32 Zyklen komplett
TC1CmpAIsrO5:
```

```
      out SREG,rSreg ; Stelle SREG wieder her
      reti
;
; TC1 Compare B Interrupt Serviceroutine
;
TC1CmpBIsr:
      sbi ADCSR,ADSC ; Starte ADC-Messzyklus
      reti
;
; TC0 Overflow Interrupt Serviceroutine
;
Tc0OvfIsr:
      in rsreg,SREG ; Sichere SREG
      inc rPwmC ; Erhöhe PWM-Zähler
      brne Tc0OvfIsr1 ; nicht Null, weiter
      mov rimp,rPwmOut ; Kopiere die bisherige Kanalausgabe
      andi rimp,0xF0 ; Erhalte die vier obersten Bits (Entladebits)
      mov rPwmOut,rimp ; Setze die Kanalausgabe auf Null
      ldi rimp,0xF0 ; Maskiere die vier obersten Bits
      eor rimp,rPwmOut ; Invertiere die vier obersten Bits
      swap rimp ; vertausche die oberen und unteren Bits
      or rPwmOut,rimp ; Setze die Kanalausgabe (1 für aktive Kanäle)
TC0OvfIsr1:
      cp rPwmC,rPwm1S ; Vergleiche den Kanalzähler mit Sollwert Kanal 1
      brne TC0OvfIsr2 ; nicht gleich, springe
      cbr rPwmOut,1 ; PWM-Bit für Kanal 1 abschalten
TC0OvfIsr2:
      cp rPwmC,rPwm2S ; Vergleiche den Kanalzähler mit Sollwert Kanal 2
      brne TC0OvfIsr3 ; nicht gleich, springe
      cbr rPwmOut,2 ; PWM-Bit für Kanal 2 abschalten
TC0OvfIsr3:
      cp rPwmC,rPwm3S ; Vergleiche den Kanalzähler mit Sollwert Kanal 3
      brne TC0OvfIsr4 ; nicht gleich, springe
      cbr rPwmOut,4 ; PWM-Bit für Kanal 3 abschalten
TC0OvfIsr4:
      cp rPwmC,rPwm4s ; Vergleiche den Kanalzähler mit Sollwert Kanal 4
      brne TC0OvfIsr5 ; nicht gleich, springe
      cbr rPwmOut,8 ; PWM-Bit für Kanal 4 abschalten
TC0OvfIsr5:
      out PORTC,rPwmOut ; Auf Port C ausgeben
      out SREG,rsreg ; Stelle SREG wieder her
      reti
;
; Uart RxC Interrupt Serviceroutine
;
SioRxCIsr:
      in rSreg,SREG ; Sichere SReg
      in rimp,UCSRA ; Lese UART Fehlerbits
      andi rimp,(1<<FE)|(1<<DOR)|(1<<PE) ; Isoliere Fehler-Bits
      in rimp,UDR ; Lese empfangenes Zeichen vom UART
      breq SioRxCIsr1 ; kein Fehler, springe
      ldi rimp,'*' ; Zeige einen Empfangsfehler an
      sbrs rFlg,bUartRxLine ; Ist der Zeilenpuffer blockiert?
      out UDR,rimp ; Echo des Zeichens an UART
      rjmp SioRxCIsr4 ; Ignoriere das Zeichen
SioRxCIsr1:
      sbrc rFlg,bUartRxLine ; Zeilenpuffer blockiert?
      rjmp SioRxCIsr4 ; Ja, ignoriere das Zeichen
      out UDR,rimp ; Echo des Zeichens an UART
      push ZH ; Sichere Z-Register
```

```
      push ZL
      ldi ZH,HIGH(sUartRxBs) ; Position für nächstes Zeichen, MSB
      lds ZL,sUartRxBp ; dto., LSB
      st Z+,rimp ; speichere Zeichen im Puffer
      cpi ZL,LOW(sUartRxBe+1) ; Ende des Puffers erreicht?
      brcc SioRxCIsr2 ; Pufferberlauf
      sts sUartRxBp,ZL ; Speichere nächste Pufferposition
SioRxCIsr2:
      cpi rimp,cCr ; Wagenrücklauf?
      brne SioRxCIsr3 ; Nein, mach weiter
      ldi rimp,cLf ; Echo Zeilenvorschub an UART
      out UDR,rimp ; in Ausgaberegister des UART
      sbr rFlg,(1<<bUartRxLine) ; Setze Zeile-Komplett-Flag
SioRxCIsr3:
      pop ZL ; Stelle Z-Register wieder her
      pop ZH
SioRxCIsr4:
      out SREG,rSreg ; Stelle SREG wieder her
      reti ; Return vom Interrupt
;
;************************** Hauptprogram ******************
;
; Beginn des Hauptprogramms, Init der Hardware und der SRAM-Variablen
;
main:
      ldi rmp,HIGH(RAMEND) ; Stelle Stapelzeiger ein
      out SPH,rmp
      ldi rmp,LOW(RAMEND)
      out SPL,rmp
      ; Initiiere Register und SRAM
      clr rFlg ; Flaggen löschen
      clr rFlgD ; Anzeigeflaggen löschen
      ldi rFlgD,0x0F ; LCD-Zeilenzugriff ermöglichen
      ; Initiiere die Ports
      clr rmp ; Port A alle Bits Eingang
      out DDRA,rmp
      ldi rmp,0xFF ; Port C alle Bits Ausgang
      out DDRC,rmp
      ldi rmp,0xF2 ; Port D Bits 7..4 und 1 als Ausgänge
      out DDRD,rmp
      ; Initiiere die Tasten
      rcall KeyInit ; Init-Routine für Tasten aufrufen
      ; Initiiere die LEDs
      rcall KeyLedInit ; Init-Routine für die LEDs aufrufen
      ; Statusflaggen und Systemzeit initiieren
      clr rmp
      ldi ZH,HIGH(sState) ; Statusflagge
      ldi ZL,LOW(sState)
      st Z+,rmp ; sState
      st Z+,rmp ; sTimeM
      st Z+,rmp ; sTimeH
      st Z+,rmp ; sTimeD
      st Z+,rmp ; LCD-Cursorposition
      ldi ZH,HIGH(sAdcC) ; Initiiere mAh-Werte
      ldi ZL,LOW(sAdcC)
      ldi rmp,16 ; 16 Bytes löschen
      clr R0
main1:
      st Z+,R0 ; Position Null setzen
      dec rmp ; nächste Position
```

```
      brne main1 ; noch welche
      ldi ZH,HIGH(sCh1UV) ; Initiiere Akku Sollwerte
      ldi ZL,LOW(sCh1UV)
      ldi rmp,32
main2:
      st Z+,R0 ; auf Null setzen
      dec rmp ; nächste Position
      brne main2 ; noch welche
      sts sUartCurCh,R0 ; Setze UART Kanal auf Null
      ; Initiiere die EEPROM Liste der Akku-Id's
      rcall EpInit ; rufe EEPROM-Init-Routine auf
      ; Initiiere den UART
      call UartInit ; Initiiere UART-Empfang und -Senden
      call UartSendOpenText ; Sende den Eröffnungstext
      call UartSendCursor ; Sende den Cursor ber den UART
      ; Initiiere die LCD-Anzeige
      rcall LcdInit ; Rufe die LCD-Init-Routine auf
      ; Initiiere die PWM-Kanäle
      clr rPwm1S ; Setze den Anfangswert für Kanal 1
      clr rPwm2S ; Setze den Anfangswert für Kanal 2
      clr rPwm3S ; Setze den Anfangswert für Kanal 3
      clr rPwm4S ; Setze den Anfangswert für Kanal 4
      clr rPwmOut ; PWM-Ausgabe deaktivieren
      ; Initiiere die Timer 0 und 1
      ldi rmp,HIGH(cTC1Div) ; TC1: Teiler für das 64 Hz-Signal
      out OCR1AH,rmp ; MSB zuerst!
      ldi rmp,LOW(cTC1Div)
      out OCR1AL,rmp ; LSB zuletzt!
      ldi rmp,HIGH(cTC1CompB) ; MSB zuerst!
      out OCR1BH,rmp
      ldi rmp,LOW(cTC1CompB) ; LSB zuletzt!
      out OCR1BL,rmp
      clr rmp ; TC1: TC1-Mode: keine Ausgabe, kein PWM
      out TCCR1A,rmp
      ldi rmp,(1<<WGM12)|(1<<CS11) ; TC1: CTC und Vorteiler = 8
      out TCCR1B,rmp
      ldi rmp,(1<<TOIE0)|(1<<OCIE1A)|(1<<OCIE1B) ;TC0Überlauf und TC1CompA/B Int
      out TIMSK,rmp
      ldi rmp,1<<CS00 ; Setze TC0 Vorteiler auf 1
      out TCCR0,rmp
      ; Initiiere den ADC
      ldi YH,HIGH(sAdcE1) ; Y ist der ADC-Ergebnis-Zeiger
      ldi YL,LOW(sAdcE1)
      clr rChCH ; Setze Kanalzähler für ADC auf Null
      clr rChCL
      ldi rmp,cAdcV ; Setze ADC-Referenzspannung und MUX auf Kanal 0
      out ADMUX,rmp
      ldi rmp,cStartAdc ; Starte ADC, Taktvorteiler=128, keine Interrupts
      out ADCSR,rmp
      sei ; Ermögliche Interrupts
loop: ; Hauptprogramm-Schleife
      ; Keinen SLEEP-Mode hier benutzen, da das den ADC-Ablauf stört
      tst rFlg ; Irgendeine Flagge gesetzt?
      breq loop ; nein, starte Schleife neu, bis was passiert
      ldi rmp,LOW(loop) ; Schleifenanfang als Rcksprungadresse auf den Stapel
      push rmp
      ldi rmp,HIGH(loop)
      push rmp
      ; Prüfe die 64 Hz Tick Flagge von Timer 0
      sbrc rFlg,b64Hz ; ein Tick?
```

```
      rjmp Tick64Hz ; lese Tasten und werte sie aus
      ; Prüfe ob 32 Messzyklen komplett sind und 4 Sekunden vorüber sind
      sbrc rFlg,b32m ; Prüfe 32 Messungen und 4 Sekunden vorüber
      rjmp Adc32m ; Springe zur Messauswertung
      ; Prüfe ob Hexadezimalwerte über das UART ausgegeben werden sollen (Debug)
      sbrc rFlgD,bURdy ; Werte Flagge aus
      rjmp DisplHex ; Gib Hexwerte über das UART aus (Debug)
      ; Prüfe ob Ergebnisse über die LCD-Anzeige ausgegeben werden können
      sbrc rFlgD,bLRdy ; Prüfe LCD-Werte fertig
      rjmp DisplLcd ; rufe LCD-Anzeige-Routine auf
      ; Prüfe ob eine Minute vorber ist
      sbrc rFlg,bMin ; Werte Minuten-Flagge aus
      rjmp Minute ; Springe zur Minutenausgabe
      ; Prfe ob eine über das UART empfangene Zeile komplett ist
      sbrs rFlg,bUartRxLine ; Werte Rx-Zeile-Komplett-Flagge aus
      ret ; nein
      jmp UartRxLine ; ja, beantworte empfangene UART-Zeile
;
; Ein 64 Hz Tick
;
Tick64Hz:
      cbr rFlg,1<<b64Hz ; Setze Flagge zurck
      rcall KeyLeds ; rufe die LED-Auswertungsroutine auf
      rjmp Keys ; rufe die Tastenauswertung auf
;
; Binde die Berechnungsroutinen ein
;
.INCLUDE "akkucalc.asm"
;
; Binde die Tastenauswertungsroutinen ein
;
.INCLUDE "akkukey.asm"
;
; Binde die LCD-Anzeigenroutinen ein
;
.INCLUDE "akkulcd.asm"
;
; Binde die UART-Routinen ein
;
.INCLUDE "akkuuart.asm"
;
; Ende des Code-Segments
;
; EEPROM-Segment, Initiiere meine Akkus
;
.ESEG
;
; Akku-Id-Definition:
;   Byte 1: Id-Nummer, muss von 1 an in fortgesetzter Reihenfolge sein
;   Byte 2: Bauart der Zelle, A0(Mono), A1(Baby), A2(AA), A3(AAA)
;   Byte 2 und 3: Nennkapazität in mAh, LSB and MSB
;   Byte 4 and 5: Anzahl der Volladungen mit Nennkapazität, LSB and MSB
;   Byte 6 and 7: Ladekapazität, verbleibende mAh der letzten Ladung
;   Byte 8 to 16: Textbeschreibung der Zelle, exakt 8 Buchstaben lang
;
; Akku 1: AAA, NiMH, Conrad 250185
.DB 0x01,0xA3
.DW 500,4,0
.DB "ConradMH"
;
```

```
; Akku 2: AAA, NiMH, Conrad 250185
.DB 0x02,0xA3
.DW 500,4,0
.DB "ConradMH"
;
; Akku 3: AAA, NiMH, Conrad 250185
.DB 0x03,0xA3
.DW 500,4,0
.DB "ConradMH"
;
; Akku 4: AAA, NiMH, Conrad 250185
.DB 0x04,0xA3
.DW 500,4,0
.DB "ConradMH"
;
; Akku 5: AA, NiMH, Pixcell Camera
.DB 0x05,0xA2
.DW 1800,10,0
.DB "PixcelMH"
;
; Akku 6: AA, NiMH, Pixcell Camera
.DB 0x06,0xA2
.DW 1800,10,0
.DB "PixcelMH"
;
; Akku 7: AA, NiMH, Pixcell Camera
.DB 0x07,0xA2
.DW 1800,10,0
.DB "PixcelMH"
;
; Akku 8: AA, NiMH, Pixcell Camera
.DB 0x08,0xA2
.DW 1800,10,0
.DB "PixcelMH"
;
; Akku 9: AA, NiMH, Conrad Voltcraft
.DB 0x09,0xA2
.DW 1700,6,0
.DB "VoltcrMH"
;
; Akku 10: AA, NiMH, Conrad Voltcraft
.DB 0x0A,0xA2
.DW 1700,6,0
.DB "VoltcrMH"
;
; Akku 11: AA, NiMH, Conrad Voltcraft
.DB 0x0B,0xA2
.DW 1700,6,0
.DB "VoltcrMH"
;
; Akku 12: AA, NiMH, Conrad Voltcraft
.DB 0x0C,0xA2
.DW 1700,6,0
.DB "VoltcrMH"
;
; Ende der Akkumulator Id-Definitionen
;
```

Include-Datei AkkuCalc.asm

```
; ***************************************************************
; * Include Routine für Akkuload, Berechnungs- und Anzeigeroutinen *
; * (C)2005 by Gerh.Schmidt, info@avr-asm-tutorial.net            *
; ***************************************************************
;
; ************** EEPROM Init, Lesen und Schreiben *****************
;
; Initiiert den EEPROM-Speicher und stellt gespeicherte Id's fest
;
EpInit:
      clr ZH ; Z ist EEPROM-Lesezeiger
      clr ZL
      clr R0 ; R0 ist Zähler
EpInit1:
      sbic EECR,EEWE ; Warte auf Schreib-Ende
      rjmp EpInit1 ; Warte weil Schreib-Bit gesetzt
      inc R0 ; nächste Akku-Id
      ldi rmp,cEepMax+1 ; Maximalanzahl Id's
      cp R0,rmp ; vergleiche
      brcc EpInit4 ; Ende erreicht
      out EEARL,ZL ; EEPROM-Leseadresse setzen
      out EEARH,ZH
      sbi EECR,EERE ; EEPROM-Lesebit setzen
EpInit2:
      sbic EECR,EERE ; warte auf Lesen ok
      rjmp EpInit2 ; warte weiter
      in rmp,EEDR ; Lese Akku-Id-Nummer
      cp rmp,R0 ; vergleiche mit aktueller Nummer
      brne EpInit4 ; nicht richtig, beende
      adiw ZL,1 ; nächstes Byte im EEPROM
      out EEARL,ZL ; EEPROM-Lese-Adresse setzen
      out EEARH,ZH
      sbi EECR,EERE ; Read Enable setzen
EpInit3:
      sbic EECR,EERE ; warte auf Lesen ok
      rjmp EpInit3 ; warte weiter
      in rmp,EEDR ; Lese Typangabe
      cpi rmp,0xA0 ; muss zwischen hex A0 und A3 sein
      brcs EpInit4 ; kleiner als A0, beenden
      cpi rmp,0xA4 ; größer als A3?
      brcc EpInit4 ; größer als A3, beenden
      adiw ZL,15 ; nächste EEPROM-Position
      rjmp EpInit1 ; weiter
EpInit4:
      dec R0 ; speichere Anzahl korrekter Akku-Id's
      sts sEaNm,R0
      ret
;
; Lese den EEPROM-Inhalt der Akku-Id in rmp in den SRAM-Puffer
;
EpRead:
      dec rmp ; Startet mit Id=1
      mov ZL,rmp ; Kopiere nach Z
      clr ZH ; MSB auf Null, Adresse mit 16 multiplizieren
      lsl ZL ; *2
      rol ZH
      lsl ZL ; *4
      rol ZH
```

```
      lsl ZL ; *8
      rol ZH
      lsl ZL ; *16
      rol ZH
      ldi XH,HIGH(sEepB) ; X ist SRAM-Puffer-Zeiger
      ldi XL,LOW(sEepB)
EpRead1:
      sbic EECR,EEWE ; warte auf Schreibende
      rjmp EpRead1
      out EEARH,ZH ; gib EEPROM-Adresse aus
      out EEARL,ZL
      sbi EECR,EERE ; Setze Read Enable
EpRead2:
      sbic EECR,EERE ; Warte auf Lese-Ende
      rjmp EpRead2
      in rmp,EEDR ; Lese Byte
      st X+,rmp ; Speichere im SRAM-Puffer
      adiw ZL,1 ; nächstes Byte im EEPROM
      mov rmp,ZL ; Teste auf Ende
      andi rmp,0x0F ; letzte vier Bits = Null?
      brne EpRead1 ; Lese weiteres Byte
      ret
;
; Schreibe den Pufferinhalt im SRAM in das EEPROM
;
EpWrite:
      ldi XH,HIGH(sEepB) ; X ist der SRAM-Puffer-Zeiger
      ldi XL,LOW(sEepB)
      ld rmp,X ; Lese die ID-Nummer
      dec rmp ; Id's starten mit Id=1
      mov ZL,rmp ; Kopiere nach Z
      clr ZH ; MSB leeren, EEPROM-Adresse mit 16 multiplizieren
      lsl ZL ; *2
      rol ZH
      lsl ZL ; *4
      rol ZH
      lsl ZL ; *8
      rol ZH
      lsl ZL ; *16
      rol ZH
EpWrite1:
      sbic EECR,EEWE ; warte auf Schreibende
      rjmp EpWrite1
      out EEARH,ZH ; gib EEPROM-Schreibadresse aus
      out EEARL,ZL
      ld rmp,X+ ; Lese Byte aus SRAM-Puffer
      out EEDR,rmp ; Schreibe ins EEPROM-Datenregister
      cli ; Stop Interrupts
      sbi EECR,EEMWE ; Setze Schreibmode Enable
      sbi EECR,EEWE ; Setze Write Enable
      sei ; ermögliche Interrupts wieder
      adiw ZL,1 ; nächste Adresse im EEPROM
      mov rmp,ZL ; Test auf Ende
      andi rmp,0x0F ; Letzte vier Bits = Null?
      brne EpWrite1 ; Schreibe noch weiter Bytes
      ret
;
; Prüfe Akku-Id und addiere die Ladekapazität zur gespeicherten Id im EEPROM
; R0 ist der Kanal, erhält den Inhalt von R0 und R1!
;
```

```
EpStore:
      rcall GetAkkuId ; Lese die Id des aktuellen Kanals
      tst rmp ; Id=0?
      breq EpStore4 ; ja, beende
      lds ZL,sEANm ; Lese die Anzahl Id's im EEPROM
      sec ; setze Carry-Flagge
      cpc rmp,ZL ; vergleiche mit (Anzahl der Id's+1)
      brcc EpStore4 ; größer als die Anzahl der gespeicherten Id's
      rcall EpRead ; lese die gewählte Id in das SRAM
      ldi ZH,HIGH(sAdcC) ; Zeiger auf die gemessene Ladekapazität in mAh
      ldi ZL,LOW(sAdcC)
      add ZL,R0 ; addiere Kanal zur Adresse
      add ZL,R0
      ld XL,Z+ ; gemessene Ladekapazität in X lesen
      ld XH,Z
      lds ZL,sENRL ; Restkapazität vom Puffer nach Z lesen
      lds ZH,sENRM
      add ZL,XL ; gemessene Ladekapazität addieren
      adc ZH,XH
      lds XL,sENCL ; Nennkapazität des Akkus in X
      lds XH,sENCM
      clr rmp ; Zähler für Anzahl Volladungen leeren
EpStore1:
      cp ZL,XL ; Restkapazität in Z mit Nennkapazität in X vergleichen
      cpc ZH,XH
      brcs EpStore2 ; Restkapazität kleiner als Nennkapazität
      sub ZL,XL ; subtrahiere Nennkapazität
      sbc ZH,XH
      inc rmp ; Erhöhe Anzahl der Volladungen
      rjmp EpStore1 ; weiter
EpStore2: ; Anzahl der Volladungen ist in rmp, Restkapazität in Z
      sts sENRL,ZL ; Restkapazität im SRAM-Puffer speichern
      sts sENRM,ZH
      tst rmp ; eine volle Ladung oder mehr?
      breq EpStore3 ; nein
      lds ZL,sENLL ; Lese Anzahl Volladungen aus SRAM-Puffer
      add ZL,rmp ; addiere zur Anzahl Volladungen
      sts sENLL,ZL ; Speichere LSB der Anzahl Volladungen
      brcc EpStore3 ; kein Übertrag
      lds ZL,sENLM ; erhöhe MSB der Anzahl Volladungen
      inc ZL
      sts sENLM,ZL ; speichere MSB der Anzahl Volladungen
EpStore3: ; SRAM-Puffer ist komplett, speichere
      rcall EpWrite ; Schreibe den Puffer ins EEPROM
EpStore4:
      ret
;
; *** Wandle die gemessenen Spannungen in dezimal und berechne Strom dezimal ***
;
; Wandle die Summe aus 32 ADC-Messungen in binäre Spannung um
; Die Summe sd32e ist in R3:R2, das Ergebnis in R3:R2, verwendet R6:R5:R4
; Referenzspannung ist 5V, Umrechnung: U[mV] = s32e * 39 / 256
;
Adc2mV:
      mov R4,R2 ; Kopiere s32e nach R6:R5:R4 und multipliziere mit 8
      mov R5,R3
      clr R6
      lsl R4 ; * 2
      rol R5
      rol R6
```

```
	lsl R4 ; * 4
	rol R5
	rol R6
	lsl R4 ; * 8
	rol R5
	rol R6
	mov R0,R4 ; Kopiere Zwischenergebnis in rmp:R1:R0, multipliziere mit 32
	mov R1,R5
	mov rmp,R6
	lsl R4 ; * 16
	rol R5
	rol R6
	lsl R4 ; * 32
	rol R5
	rol R6
	add R4,R0 ; * 40, addiere 8-fach und 32-fach = 40-fach
	adc R5,R1
	adc R6,rmp
	clr rmp
	sub R4,R2 ; * 39, ziehe Ausgangswert einmal ab = 39-fach
	sbc R5,R3
	sbc R6,rmp
	mov R2,R5 ; / 256, Kopiere die beiden obersten Bytes ins Ergebnis (=/256)
	mov R3,R6
	ldi rmp,0x80 ; runde das Ergebnis
	add rmp,R4
	clr rmp
	adc R2,rmp
	adc R3,rmp
	ret
;
; Negativen in positiven bzw. positiven in negativen Wert umwandeln (R3:R2)
;
Negative:
	neg R2 ; Zweierkomplement
	clr rmp
	sbc rmp,R3
	mov R3,rmp
	ret
;
; Umrechnung der Differenzspannung über dem Widerstand (in mV) in Strom (mA)
; mV Differenz in R3:R2, Ergebnis in R3:R2
; ADC Referenzspannung ist 5V
; Umrechnung: mA = 11.84 * (s32ungerade - s32gerade) / 256
;
mV2mA:
	ldi rmp,0x80 ; teste ob Spannungsdifferenz negativ
	and rmp,R3
	mov R0,rmp ; Speichere Vorzeichen
	breq mV2mA1 ; nicht negativ
	; wandle negativ in positiv um
	rcall negative
mV2mA1:
	mov R4,R2 ; kopiere nach R6:R5:R4
	mov R5,R3
	clr R6
	lsl R4 ; * 2, multipliziere mit zwei
	rol R5
	rol R6
	add R4,R2 ; * 3, addiere Ausgangswert = * 3
```

```
      adc R5,R3
      clr rmp
      adc R6,rmp
      lsl R4 ; * 6, multipliziere mit zwei = * 6
      rol R5
      rol R6 ; <== Fehler in der Version vom Maerz 2005
      lsl R4 ; * 12, multipliziere mit zwei = * 12
      rol R5
      rol R6
      lsr R3 ; R3:R2 / 2, dividiere Ausgangswert / 8
      ror R2
      lsr R3 ; R3:R2 / 4
      ror R2
      lsr R3 ; R3:R2 / 8
      ror R2
      sub R4,R2 ; * 11.88, subtrahiere ein Achtel
      sbc R5,R3
      sbc R6,rmp
mV2mA3:
      mov R2,R5 ; kopiere Ergebnis nach R3:R2, teilen durch 256
      mov R3,R6
      ldi rmp,0x80 ; runde Ergebnis
      add rmp,R4
      clr rmp
      adc R2,rmp
      adc R3,rmp
      tst R0 ; prüfe negativ
      breq mV2mA4 ; nein
      rjmp negative ; wandle in negativ um
mV2mA4:
      ret
;
; Rechne Strom in mA in Kapazität in mAh um
; Anfang: R6:R5:R4 enthält den Strom als 256*I(mA)
; Ende: R3:R2 ist die Ladekapazität über 4 Sekunden als 256*mAh
;
mA2mAh:
      mov R1,R4 ; Kopiere den Ausgangswert nach R3:R2:R1
      mov R2,R5
      mov R3,R6
      lsr R3 ; dividiere den Strom durch 2
      ror R2
      ror R1
      lsr R3 ; ... durch 4
      ror R2
      ror R1
      ; Der Strom in mA, multipliziert mit 64, ist jetzt in R3:R2:R1
      clr rmp
      add R1,R5 ; addiere den Original-Strom, um 65 * I zu erhalten
      adc R2,R6
      adc R3,rmp
      ; Der Strom in mA, multipliziert mit 65, ist jetzt in R3:R2:R1
      lsl R4 ; Multiplikation des Ausgangsstroms mit 8
      rol R5
      rol R6
      lsl R4 ; * 4
      rol R5
      rol R6
      lsl R4 ; * 8
      rol R5
```

```
      rol R6
      clr rmp ; Runden
      rol R4 ; oberstes Bit ins Carry
      adc R5,rmp
      adc R6,rmp
      add R1,R5 ; addiere 8*I zu 65*I = 73*I
      adc R2,R6
      adc R3,rmp
      ; Das Ergebnis von 73*I/256 ist jetzt in R3:R2
      ; Diese Zahl ist gleich der Ladekapazität in mAh, multipliziert mit 256,
      ; die in den vergangenen vier Sekunden in den Akku geladen wurden
      ret
;
; ********* 32 Messungen der Spannung sind komplett *******
;
; ADC hat 32 Messungen durchgeführt und aufsummiert
;
Adc32m:
      cbr rFlg,1<<b32m ; lösche die 4s Flagge
      mov rmp,rChCH ; prüfe das MSB des Messzählers
      andi rmp,0x0F ; prüfe ob das niedrige Nibble 15 erreicht hat
      cpi rmp,15
      brcs Adc32m1 ; nein, mach weiter
      sbr rFlg,1<<bMin ; setze die Minuten-Flagge
      ldi rmp,1 ; korrigiere das niedrige Nibble des MSB des Messzählers
      add rChCH,rmp ; addiere eine Eins
      mov rmp,rChCH ; Prüfe das obere Nibble des MSB des Messzählers
      andi rmp,0xF0 ; 15 Minuten erreicht?
      cpi rmp,0xF0 ; oberes Nibble = 15?
      brne Adc32m1 ; nein, mach weiter
      ldi rmp,0x10 ; korrigiere MSB
      add rChCH,rmp ; addiere 10
      ; 15 Minuten sind hier vorber, wird nicht weiter ausgewertet
Adc32m1: ; berechne alle Spannungen
      ldi ZH,HIGH(sAdcE4); Z zeigt auf die gemessenen Werte
      ldi ZL,LOW(sAdcE4)
Adc32m2:
      ld R2,Z ; Kopiere den Messwert nach R3:R2
      ldd R3,Z+1
      rcall Adc2mV ; rechne in mV um
      std Z+16,R2 ; speichere Ergebnis im SRAM
      std Z+17,R3
      adiw ZL,2 ; der nächste gemessene Wert
      cpi ZL,LOW(sAdcE4+16) ; Ende erreicht?
      brcs Adc32m2
      ; Berechne Ströme und Ladekapazitäten
      ldi ZH,HIGH(sAdcE4) ; Z zeigt auf die gemessenen Werte
      ldi ZL,LOW(sAdcE4)
Adc32m3:
      ld R0,Z ; Kopiere die Summe des geraden Messkanals LSB nach R0
      ldd R1,Z+1 ; dto., MSB nach R1
      ldd R2,Z+8 ; Kopiere die Summe des ungeraden Messkanals LSB nach R2
      ldd R3,Z+9 ; dto., MSB nach R3
      sub R2,R0 ; subtrahiere LSB
      sbc R3,R1 ; dto., MSB
      rcall mV2mA ; berechne den Strom in mA
      std Z+32,R2 ; Speichere den Strom im SRAM, LSB
      std Z+33,R3 ; dto., MSB
      rcall mA2mAh ; berechne die mAh aus dem Strom in mA
      ldd rmp,Z+48 ; lese das LLSB der mAh in rmp
```

```
      ldd R0,Z+40 ; lese LSB mAh nach R0
      ldd R1,Z+41 ; lese MSB mAh nach R1
      add rmp,R2 ; addiere das LLSB
      adc R0,R3 ; addiere das LSB
      clr R3 ; add das MSB
      adc R1,R3
      std Z+48,rmp ; speichere das LLSB im SRAM
      std Z+40,R0 ; speichere das LSB im SRAM
      std Z+41,R1 ; speichere MSB im SRAM
      adiw ZL,2 ; nächster Kanal
      cpi ZL,LOW(sAdcE4+8) ; am Ende?
      brcs Adc32m3 ; weiter insgesamt vier mal
      sbr rFlgD,(1<<bURdy)|(1<<bLRdy) ; Setze die Ergebnis-fertig-Flaggen
      ret
;
; ******* Vergleiche den aktuellen Kanal-Status mit dem Sollwert und ändere ****
;
; Addiere den Kanal in R0 zur Adresse in Z
;
ChAdrs:
      mov rmp,R0 ; Kanal in rmp kopieren
      lsl rmp ; * 2
      lsl rmp ; * 4
      lsl rmp ; * 8
      add ZL,rmp ; und zur Basisadresse in Z addieren
      ret
;
; Gibt die Akku-Id vom Kanal in R0 in rmp zurck
;
GetAkkuId:
      ldi ZH,HIGH(sAdcAI1) ; Z ist die Adresse des Kanals
      ldi ZL,LOW(sAdcAI1)
      add ZL,R0 ; addiere Kanal
      add ZL,R0
      ld rmp,Z ; Lese Id
      ret
;
; Setzt die Sollwerte für die gewählte Akku-Id
; Erwartet den Kanal in R0
;
IdSetPar:
      rcall GetAkkuId ; Lese die Akku-Id des Kanals
      tst rmp ; = Null?
      breq IdSetPar1 ; ja, tue nichts
      lds R1,sEANm ; Lese die Anzahl gespeicherter Id's
      sec
      cpc rmp,R1 ; vergleiche mit der maximalen Anzahl Id's
      brcc IdSetPar1 ; zu groß, tu nichts
      rcall EpRead ; Lese die Akku-Id in den SRAM-Puffer
      lds R2,sENCL ; Lese die Nennkapazität, LSB
      lds R3,sENCM ; dto., MSB
      ldi ZH,HIGH(sCh1UV) ; Z auf die Sollwerte des Kanals setzen
      ldi ZL,LOW(sCh1UV)
      rcall ChAdrs ; Adresse des Kanals
      ldi rmp,LOW(cUnlVol) ; Setze die Entladespannung auf Defaultwert
      st Z+,rmp
      ldi rmp,HIGH(cUnlVol)
      st Z+,rmp
      mov R4,R2 ; Berechne den Ladestrom aus der Nennkapazität
      mov R5,R3
```

```
      lsr R5
      ror R4
      lsr R5
      ror R4
      lsr R5
      ror R4
      st Z+,R4 ; setze den Sollwert Ladestrom
      st Z+,R5
      add R2,R4 ; Berechne die Ladekapazität aus der Nennkapazität
      adc R3,R5
      st Z+,R2 ; Speichere den Sollwert Ladekapazität
      st Z+,R3
      lsl R2 ; Berechne den Erhaltungsstrom aus der Nennkapazität
      rol R3
      st Z+,R3 ; Speichere den Sollwert Erhaltungsstrom
      clr rmp
      st Z+,rmp
IdSetPar1:
      ret
;
; Berechne die PWM-Adresse des aktuellen Kanals in R0 in Z
;
PwmAdrs:
      ldi ZH,HIGH(cPwm1S) ; Z auf Sollwert von Kanal 1
      ldi ZL,LOW(cPwm1S)
      add ZL,R0 ; Addiere aktuellen Kanal in R0 zur Adresse
      ret
;
; Setze den PWM-Wert in Kanal R0 auf den Wert in rmp und l che das Unload-Bit
;
PwmSet:
      ldi ZL,0xEF ; Unload-Bit für Kanal 0
      mov ZH,R0 ; Kopiere Kanal
PwmSet1:
      tst ZH ; ok?
      breq PwmSet2 ; ja
      sec ; Setze Carry-Bit vor dem Rollen
      rol ZL ; nächsthöheres Bit
      dec ZH ; nächster Kanal
      rjmp PwmSet1 ; und wieder von vorn
PwmSet2:
      and rPwmOut,ZL ; lösche das Unload-Bit des Kanals
      rcall PwmAdrs ; Hole die PWM-Adresse des aktuellen Kanals nach Z
      st Z,rmp ; Setze den PWM-Wert für den Kanal
      ret
;
; Setze den Kanal(in R0)status auf den Wert in rmp
;
ChSet:
      ldi ZL,0xFC ; Kanalmaske
      mov ZH,R0 ; kopiere Kanal
ChSet1:
      tst ZH ; Kanal erreicht?
      breq ChSet2 ; ja, raus
      sec ; Setze Carry-Bit für das Rollen
      rol ZL ; n hste Kanalmaske
      rol ZL
      lsl rmp ; schiebe rmp links
      lsl rmp
      dec ZH ; nächster Kanal
```

```
      rjmp ChSet1 ; und wiederholen
ChSet2:
      lds ZH,sState ; Lese Statusbits
      and ZH,ZL ; lösche Statusbits des Kanals
      or ZH,rmp ; Setze die Statusbits des Kanals
      sts sState,ZH ; neuen Status speichern
      ret
;
; Schalte den Kanal in R0 (von Entladen) auf Laden
;
ToLoad:
      ldi rmp,0x02 ; Statusbits für Laden
      rcall ChSet ; Setze Status des Kanals in R0
      ldi rmp,150 ; Setze die PWM auf eine mittlere Spannung
      rcall PwmSet ; Setze PWM des aktuellen Kanals
      rjmp ClearCap ; Lösche den Kapazitätswert des Kanals
;
; Lösche den Kapazitätswert des Kanals in R0
;
ClearCap:
      ldi ZH,HIGH(sAdcC) ; Zeiger auf die Kapazität des Kanals
      ldi ZL,LOW(sAdcC)
      add ZL,R0
      add ZL,R0
      clr rmp
      std Z+8,rmp ; l che LLSB
      st Z+,rmp ; l che LSB
      st Z,rmp ; l che MSB
      ret
;
; Vergleiche den Strom im Kanal mit dem Sollwert und korrigiere die PWM
; Erwartet den Kanal in R0 und den Zeiger auf den Sollwert in Z
;
ChCurrent: ; Prüfe Strom zu hoch/zu niedrig und korrigiere
      rcall ChAdrs ; Adresse in Z auf korrekten Kanal-Sollwert
      ld R2,Z+ ; Sollstrom nach R3:R2 laden
      ld R3,Z
      ldi ZH,HIGH(sAdcI) ; Z auf gemessenen Strom setzen
      ldi ZL,LOW(sAdcI)
      add ZL,R0 ; Adresse des Kanals
      add ZL,R0
      ld R4,Z+ ; Ladestrom nach R5:R4 kopieren
      ld R5,Z
      sub R4,R2 ; vom Sollwert subtrahieren
      sbc R5,R3
      breq ChCurrent3 ; Strom ist korrekt, raus
      brcs ChCurrent1 ; Strom ist zu niedrig, erhöhe PWM
      rcall PwmAdrs ; Strom zu hoch, erniedrige PWM
      ld rmp,Z ; Lese PWM-Wert
      dec rmp ; niedriger setzen
      brne ChCurrent2 ; ausgeben, wenn nicht Null
      inc rmp ; Null erreicht, bleibe bei 1
      rjmp ChCurrent2 ; ausgeben
ChCurrent1: ; Strom zu niedrig, erhöhen
      rcall PwmAdrs ; Adresse des PWM-Registers holen
      ld rmp,Z ; PWM-Wert lesen
      inc rmp ; erhöhen
      brne ChCurrent2 ; kein Überlauf
      dec rmp ; Überlauf, wieder auf FF setzen
ChCurrent2: ; neuen PWM-Wert speichern
```

```
      st Z,rmp ; neuen PWM-Wert setzen
ChCurrent3: ; fertig
      ret
;
; Prfe die Ist- und Sollwerte der Kanäle
;
ChckStates:
      clr R0 ; Kanal 0
      lds R1,sState ; Statusbits holen
ChckStates1:
      mov rmp,R1 ; Status kopieren
      andi rmp,0x03 ; Statusbits des Kanals isolieren
      cpi rmp,1 ; Status inaktiv = 0?
      brcc ChckStates2 ; Status = 1 oder höher
      clr rmp ; Kanal inaktiv
      rcall PwmSet ; Setze PWM auf Null
      rjmp ChckStates6 ; nächster Kanal
ChckStates2: ; Status = 1..3
      brne ChckStates3 ; Status=2 oder höher
      ldi ZH,HIGH(sCh1UV) ; Entladen, Z auf Schwellwertspannung
      ldi ZL,LOW(sCh1UV)
      rcall ChAdrs ; Adresse des Kanals
      ld R2,Z+ ; Entladespannung in R3:R2
      ld R3,Z
      ldi ZH,HIGH(sAdcU) ; Z auf letzte gemessene Spannung
      ldi ZL,LOW(sAdcU)
      add ZL,R0 ; Adresse des Kanals
      add ZL,R0
      ld R4,Z+ ; gemessene Spannung in R4:R5
      ld R5,Z
      sub R4,R2 ; gemessene Spannung < Sollwert
      sbc R5,R3
      brcc ChckStates6 ; nein, nächster Kanal
      rcall ToLoad ; starte Ladevorgang
      rjmp ChckStates6 ; nächster Kanal
ChckStates3: ; Status ist 2 oder höher
      cpi rmp,2 ; Laden?
      brne ChckStates5 ; nein, Erhaltungsladung
      ldi ZH,HIGH(sCh1LW) ; Z auf Ladekapazität-Sollwert
      ldi ZL,LOW(sCh1LW)
      rcall ChAdrs ; Kanaladresse
      ld R2,Z+ ; Sollwert Ladekapazität in R3:R2
      ld R3,Z
      ldi ZH,HIGH(sAdcC) ; Z auf Istwert Ladekapazität
      ldi ZL,LOW(sAdcC)
      add ZL,R0 ; Kanaladresse
      add ZL,R0
      ld R4,Z+ ; Istwert Ladekapazität in R4:R3
      ld R5,Z
      sub R4,R2 ; Istwert - Sollwert
      sbc R5,R3
      brcs ChckStates4 ; noch nicht erreicht, weiter laden
      ldi rmp,0x03 ; Ende erreicht, setze Status auf Erhaltungsladung
      rcall ChSet
      ldi rmp,100 ; setze PWM auf niedrigen Wert
      rcall PwmSet
      rcall EpStore ; Addiere die Ladekapazität zur Id im EEPROM
      rjmp ChckStates6 ; nächster Kanal
ChckStates4: ; Sollwert Kapazität nicht erreicht, prüfe Strom
      ldi ZH,HIGH(sCh1LC) ; Z auf Sollwert Ladestrom
```

```
      ldi ZL,LOW(sCh1LC)
      rcall ChCurrent ; Korrigiere Ladestrom, wenn nötig
      rjmp ChckStates6 ; nächster Kanal
ChckStates5: ; Erhaltungsladung
      ldi ZH,HIGH(sCh1MC) ; Z auf Stromsollwert für Erhaltungsladung
      ldi ZL,LOW(sCh1MC)
      rcall ChCurrent ; Korrigiere Ladestrom, wenn nötig
ChckStates6: ; nächster Kanal
      lsr R1 ; Maske für nächsten Kanal
      lsr R1
      inc R0 ; nächster Kanal
      mov rmp,R0 ; Kanal = 5?
      cpi rmp,5
      brcc ChckStates7 ; ja, fertig
      rjmp ChckStates1 ; weiter
ChckStates7: ; fertig
      ret
;
; Prüft den Kanalstatus, zeigt das Ergebnis in Hex über das UART an (Debug)
;
DisplHex:
      cbr rFlgD,1<<bURdy ; Display-Flagge löschen
      rcall ChckStates ; Kanalstatus checken
      sbrs rFlgD,bUHex ; Hexanzeige-Flagge gesetzt?
      ret ; nein, fertig
      ldi ZH,HIGH(sAdcE4) ; Ergebnis in Hex anzeigen
      ldi ZL,LOW(sAdcE4)
      rjmp UartHex16 ; Roh-Messwerte in Hex ausgeben
;
; Spannungen in Zeile 1 des LCD-Displays anzeigen
;
DispVtg:
      ldi rmp,0 ; LCD auf Zeile 1
      rcall LcdLine
      ldi XH,HIGH(sAdcU) ; X als Zeiger auf die Spannungen
      ldi XL,LOW(sAdcU)
      ldi rmp,'U' ; Zeichen U ausgeben
DispVtg1:
      rcall LcdChar ; Zeichen in rmp ausgeben
      ld R2,X+ ; Spannung lesen, LSB
      ld R3,X+ ; dto., MSB
      ldi rmp,LOW(10000) ; mit 10000 mV vergleichen
      cp R2,rmp ; zu hoch?
      ldi rmp,HIGH(10000)
      cpc R3,rmp
      brcs DispVtg2 ; Spannung in Ordnung
      ldi rmp,'E' ; zu groß
      rcall LcdChar ; EEEEE ausgeben
      rcall LcdR0
      rcall LcdR0
      rcall LcdR0
      rjmp DispVtg3
DispVtg2: ; Spannung ok
      clr R0 ; führende Nullen unterdrücken
      rcall LcdDec4 ; vier Dezimalstellen ausgeben
DispVtg3:
      ldi rmp,' ' ; Blank ausgeben
      cpi XL,LOW(sAdcU+8) ; Ende erreicht?
      brcs DispVtg1 ; wiederholen
      ret
```

```
;
; Ströme in Zeile 2 der LCD anzeigen
;
DispCurr:
      ldi rmp,1 ; LCD-Zeile 2
      rcall LcdLine ; Zeile ansteuern
      ldi XH,HIGH(sAdcI) ; X auf Ströme setzen
      ldi XL,LOW(sAdcI)
      ldi rmp,'I' ; Stromsymbol ausgeben
DispCurr1:
      rcall LcdChar ; Zeichen in rmp an LCD-Display
      ld R2,X+ ; lese Strom, LSB
      ld R3,X+ ; dto., MSB
      mov rmp,R2 ; Null?
      or rmp,R3
      ldi rmp,' ' ; Leerzeichen ausgeben
      breq DispCurr2 ; Null, leer
      mov rmp,R3 ; testen ob negativ
      rol rmp ; Bit 7 des MSB
      ldi rmp,'+' ; +Zeichen in rmp laden
      brcc DispCurr2 ; nicht negativ
      rcall negative ; Wert in positiven Wert wandeln
      ldi rmp,'-' ; -Zeichen in rmp
DispCurr2: ; Vorzeichen ausgeben
      rcall LcdChar ; Vorzeichen an LCD
      clr R0 ; führende Nullen unterdrücken
      rcall LcdDec3 ; dreistellige Dezimalzahl ausgeben
      ldi rmp,' ' ; Leerzeichen
      cpi XL,LOW(sAdcI+8) ; Ende?
      brcs DispCurr1 ; nein, nächster Kanal
      ret
;
; Ladekapazität in mAh in Zeile 3 der LCD ausgeben
;
DispmAh:
      ldi rmp,2 ; Zeile 3
      rcall LcdLine ; LCD auf Zeile setzen
      ldi XH,HIGH(sAdcC) ; X auf mAh setzen
      ldi XL,LOW(sAdcC)
      ldi rmp,'C' ; C ausgeben
DispmAh1:
      rcall LcdChar ; Zeichen in rmp an LCD ausgeben
      ld R2,X+ ; lesen mAh LSB
      ld R3,X+ ; lesen mAh MSB
      mov rmp,R3 ; negativ?
      rol rmp ; Vorzeichen in Carry
      brcc DispmAh2 ; positiv
      rcall negative ; negativ, umwandeln
DispmAh2:
      clr R0 ; führende Nullen unterdrücken
      rcall LcdDec4 ; vierstellig an LCD ausgeben
      ldi rmp,' ' ; Leerzeichen
      cpi XL,LOW(sAdcC+8) ; Ende?
      brcs DispmAh1 ; nein, nächster Kanal
      ret
;
; Ergebnisse auf LCD anzeigen
;
DisplLcd:
      cbr rFlgD,1<<bLRdy ; Flagge löschen
```

```
      sbrs rFlg,bLcdOk ; LCD vorhanden?
      ret ; nein, zurück
      sbrs rFlgD,bLcd3 ; Zeile 4 verfügbar?
      rjmp DisplLcd1 ; jump over line 4 output
      ; Ausgaben in Zeile 4, hier nicht benutzt
DisplLcd1:
      sbrc rFlgD,bLcd0 ; Zeile 1 der LCD ok für Ausgabe?
      rcall DispVtg ; Spannungen anzeigen
      sbrc rFlgD,bLcd1 ; Zeile 2 der LCD verfügbar?
      rcall DispCurr ; Ströme anzeigen
      sbrc rFlgD,bLcd2 ; Zeile 3 der LCD verfügbar?
      rcall DispmAh ; Ladekapazität anzeigen
      rjmp LcdCursor ; Cursor der LCD setzen
;
; Systemzeit eine Minute höher setzen
;
Time:
      ldi ZH,HIGH(sTimeM) ; Z auf Minutenwert
      ldi ZL,LOW(sTimeM)
      ld rmp,Z ; eine Minute addieren
      inc rmp
      st Z,rmp
      cpi rmp,60 ; Stunde erreicht?
      brcs Time1 ; nein, fertig
      clr rmp ; Minuten auf Null
      st Z+,rmp
      ld rmp,Z ; nächste Stunde
      inc rmp
      st Z,rmp
      cpi rmp,24 ; nächster Tag?
      brcs Time1 ; nein, fertig
      clr rmp ; Stunde auf Null
      st Z+,rmp
      ld rmp,Z ; nächster Tag
      inc rmp
      st Z,rmp
      cpi rmp,100 ; Überlauf?
      brcs Time1 ; nein, weiter
      clr rmp ; Tag auf Null
      st Z,rmp
Time1:
      ret
;
; Ergebnisse ber UART ausgeben
;
Monitor:
      lds R1,sState ; Statusbits in R1
      clr R5 ; R5 ist Kanalzähler
Monitor1:
      mov rmp,R1 ; Status nach rmp kopieren
      andi rmp,0x03 ; Statusbits isolieren
      brne Monitor3 ; Status>0
Monitor2: ; Status = Clear, nichts ausgeben
      lsr R1 ; nächster Kanal
      lsr R1
      inc R5 ; Kanalzahl
      ldi rmp,4
      cp R5,rmp ; Ende?
      brcs Monitor1 ; nein, weiter
      ret
```

```
Monitor3: ; Status > 0, ausgeben
      ldi rmp,'1' ; Sende Kanalnummer
      add rmp,R5
      rcall UartSendChar
      ldi rmp,':' ; Sende :
      rcall UartSendChar
      ldi rmp,' '
      rcall UartSendChar
      ldi rmp,'N' ; Sende Id
      rcall UartSendChar
      ldi rmp,'='
      rcall UartSendChar
      ldi ZH,HIGH(sAdcAI1) ; Z auf Id
      ldi ZL,LOW(sAdcAI1)
      add ZL,R5
      add ZL,R5
      ld R2,Z
      clr R3
      rcall UartSendNmbr
      ldi rmp,','
      rcall UartSendChar
      ldi rmp,' '
      rcall UartSendChar
      ldi rmp,'S' ; Sende Status
      rcall UartSendChar
      ldi rmp,'='
      rcall UartSendChar
      mov rmp,R1
      andi rmp,0x03
      subi rmp,-'0'
      rcall UartSendChar
      ldi rmp,','
      rcall UartSendChar
      ldi rmp,' '
      rcall UartSendChar
      ldi ZH,HIGH(sAdcU) ; Sende Spannung, Z auf Spannung
      ldi ZL,LOW(sAdcU)
      add ZL,R5
      add ZL,R5
      ld R2,Z+ ; lese Spannung
      ld R3,Z+
      rcall UartSendNmbr
      ldi rmp,'m'
      rcall UartSendChar
      ldi rmp,'V'
      rcall UartSendChar
      ldi rmp,','
      rcall UartSendChar
      ldi rmp,' '
      rcall UartSendChar
      ldi ZH,HIGH(sAdcI) ; Sende Strom, Z auf Strom
      ldi ZL,LOW(sAdcI)
      add ZL,R5
      add ZL,R5
      ld R2,Z+
      ld R3,Z
      mov rmp,R3 ; negativ?
      rol rmp
      brcc Monitor4
      rcall Negative
```

```
      ldi rmp,'-'
      rcall UartSendChar
Monitor4:
      rcall UartSendNmbr
      ldi rmp,'m'
      rcall UartSendChar
      ldi rmp,'A'
      rcall UartSendChar
      ldi rmp,','
      rcall UartSendChar
      ldi rmp,' '
      rcall UartSendChar
      ldi ZH,HIGH(sAdcC) ; Sende Kapazität, Z auf Kapazität
      ldi ZL,LOW(sAdcC)
      add ZL,R5
      add ZL,R5
      ld R2,Z+
      ld R3,Z
      rcall UartSendNmbr
      ldi rmp,'m'
      rcall UartSendChar
      ldi rmp,'A'
      rcall UartSendChar
      ldi rmp,'h'
      rcall UartSendChar
      ldi rmp,cCr
      rcall UartSendChar
      ldi rmp,cLf
      rcall UartSendChar
      rjmp Monitor2 ; nächster Kanal
;
; Zeit auf LCD ausgeben
;
DisplTimeL:
      sbrs rFlg,bLcdOk ; Lcd ok?
      ret
      sbrs rFlgD,bLcd3 ; Zeile 4 verfügbar?
      ret
      ldi rmp,3 ; Zeile 4 setzen
      mov R1,rmp ; drei Zahlen darzustellen
      rcall LcdLine ; Line 4
      ldi XH,HIGH(sTimeD+1) ; X auf Zeit in Tagen
      ldi XL,LOW(sTimeD+1)
DisplTimeL1:
      ld R2,-X ; Zeit lesen
      clr R3 ; MSB l chen
      mov R0,R1 ; fhrende Nullen anzeigen
      rcall LcdDec2 ; zwei Digits ausgeben
      mov rmp,R1
      cpi rmp,1 ; : ausgeben?
      breq DisplTimeL2 ; nein
      ldi rmp,':' ; : anzeigen
      rcall LcdChar
DisplTimeL2:
      dec R1 ; alle gesendet?
      brne DisplTimeL1 ; nein, nächste Zahl
      ldi rmp,' ' ; Leerzeichen anhängen
      rcall LcdChar
      rjmp LcdCursor ; Cursor setzen
;
```

```
; Zeit prüfen und über UART ausgeben
;
DispTimeU:
      ldi rmp,cUartTime ; Minutenintervall der Ausgabe
      mov R0,rmp
      lds rmp,sTimeM ; Minuten lesen
DispTimeU1:
      tst rmp ; Null?
      breq DispTimeUd
      sub rmp,R0 ; subtrahiere Intervall
      brcc DispTimeU1 ; nicht Null, weiter prüfen
      ret ; nicht ausgeben
;
; Zeitausgabe ber UART
;
DispTimeUd:
      ldi rmp,'T' ; Zeit ausgeben
      rcall UartSendChar
      ldi rmp,'='
      rcall UartSendChar
      ldi rmp,3 ; drei Werte
      mov R1,rmp
      ldi ZH,HIGH(sTimeD+1) ; Z auf Tage
      ldi ZL,LOW(sTimeD+1)
DispTimeUd1:
      ld R2,-Z ; Zahl lesen
      clr R3
      mov R0,R1 ; führende Nullen nicht unterdrücken
      rcall UartDec2
      mov rmp,R1 ; : ausgeben?
      cpi rmp,1
      breq DispTimeUd2 ; nein
      ldi rmp,':'
      rcall UartSendChar
DispTimeUd2:
      dec R1
      brne DispTimeUd1 ; nächste Zahl
      ldi rmp,cCr ; Wagenrücklauf
      rcall UartSendChar
      ldi rmp,cLf ; Zeilenvorschub
      rjmp UartSendChar
;
; Eine Minute ist vergangen, Ausgabe wenn enabled
;
Minute:
      cbr rFlg,1<<bMin ; Flagge löschen
      rcall Time ; Systemzeit um eine Minute erhöhen
      rcall DisplTimeL ; anzeigen
      sbrs rFlgD,bUMoni ; Monitor-Flagge auswerten
      ret ; nichts weiter zu tun
      rcall DispTimeU ; Zeit über UART ausgeben
      rjmp Monitor ; Ergebnisse über UART ausgeben
;
; Ende von akkucalc
;
```

Include-Datei akkukey.asm

```
; ************************************************************
; * Datei mit Routinen fuer Tastenbedienung des Akkuladers     *
; * (C)2005 by Gerhard Schmidt, info@avr-asm-tutorial.net      *
; ************************************************************
;
; Routine  Register  Zweck
; ---------------------------------------------------------
; KeyInit  rmp,Z,X   Initiiert Tasteneingaben, setzt Startwerte
; Keys     rmp       Fragt gedrueckte Tasten ab
;                    fuer die Hauptprogrammschleife
; KeyMenueFirst nn   Ausgabe des ersten Tastenmenues
; ---------------------------------------------------------
;
; Konstanten
;
.EQU pLedO = PORTD ; LEDs sind an Port D angeschlossen
.EQU cLedM = 0xF0 ; Alle LED Port Bits gesetzt
.EQU pKeyO = PORTD ; Tasten an Port D angeschlossen
.EQU pKeyD = DDRD ; Tastenrichtungsregister
.EQU pKeyI = PIND ; Tastenlese-Port
.EQU cKeyB = 0xE0 ; Drei Tasten an Bits 7, 6 und 5
.EQU cKeyM = 0x1F ; Maske fuer Loeschen der Bits 7, 6, und 5
;
; ============== Routine zur Initialisierung der Tasten =====
;
; Initiieren der LEDs, LEDs fuer 4 Sekunden einschalten
;
KeyLedInit:
      ldi ZH,HIGH(sLedC) ; LED PW-Zaehler Null setzen
      ldi ZL,LOW(sLedC)
      ldi rmp,5 ; 5 bytes
      clr R0
KeyLedInit1:
      st Z+,R0
      dec rmp
      brne KeyLedInit1
      in rmp,pLedO
      cbr rmp,cLedM ; loeschen des LED-Bits, LED an
      out pLedO,rmp
      ret
;
; Initiieren der Tasten
;
KeyInit:
      clr rKeyLst ; Loesche letzte Taste
      clr rKeyRep ; Loesche Zaehler fuer Tastenwiederholung
;
; Loesche die Variablen fuer Tasteneingaben
;
KeyReset:
      ldi XH,HIGH(sKeyJ1) ; Ziel im SRAM
      ldi XL,LOW(sKeyJ1)
      ldi ZH,HIGH(2*KeyInitTable) ; Quellentabelle
      ldi ZL,LOW(2*KeyInitTable)
      ldi rmp,6 ; 3 Worte, 6 Bytes
KeyReset1:
      lpm ; Lese aus Tabelle
      adiw ZL,1
```

```
      st X+,R0 ; in SRAM
      dec rmp
      brne KeyReset1
      clr rmp ; Setze Kanal = 0
      sts sKeyCh,rmp
      ret
;
; Tabelle fuer Tasteneingabe
;
KeyInitTable:
.DW KeyM1,0,0
;
; ====================== Routinen fuer Hauptschleife ============
;
; bedient die LED-Ausgabe
;
KeyLeds:
      in R0,pLedO ; Speichere aktuellen Ausgabestatus
      ldi rmp,0x0F ; Loesche alle Ausgabebits, LEDs an
      and R0,rmp
      lds rmp,sLedC ; lese LED-Zaehler
      dec rmp ; zaehle Zaehler herunter
      brne KeyLeds4 ; nicht Null, keine Zustandspruefung
      lds R1,sState ; Lese Kanalstatus
      ldi ZH,HIGH(sLed1) ; Z zeigt auf Kanal
      ldi ZL,LOW(sLed1)
      ldi XL,4 ; vier Mal wiederholen
KeyLeds1:
      lsr R1 ; niedriges Statusbit
      brcs KeyLeds2 ; niedrigstes Statusbit ist Eins
      ldi XH,32 ; Kanalvergleichswert für 50%-Anzeige
      lsr R1 ; hoeheres Statusbit lesen
      brcs KeyLeds3 ; Status=10=load, setze Kanalwert
      ldi XH,64 ; 00=inactive, spaeter LEDs abschalten
      rjmp KeyLeds3 ; setze Kanalwert
KeyLeds2: ; niedriges Statusbit ist Eins
      clr XH ; 11=Erhaltungsladung, LED dauernd an
      lsr R1 ; hoeheres Bit in Carry
      brcs KeyLeds3 ; setze Kanalwert
      ldi XH,56 ; 01=Entladen, Led an fuer 8 Zyklen
KeyLeds3: ; speichere Kanalwert und wiederhole
      st Z+,XH ; setze Kanalwert
      dec XL ; naechster Kanal
      brne KeyLeds1 ; weiterer Kanal
      ldi rmp,64 ; 64 Zyklen in Zaehler
KeyLeds4: ; vergleiche Kanalwert mit Zaehler
      sts sLedC,rmp ; speichere Zaehler
      ldi XH,0x80 ; Maske fuer Setzen eines LED-Bit
      ldi XL,4 ; vier Kanaele
      ldi ZH,HIGH(sLed1) ; Z auf Kanalwert setzen
      ldi ZL,LOW(sLed1)
KeyLeds5: ; pruefe ob LED aus sein sollte
      ld R1,Z+ ; Lese Kanalwert
      cp R1,rmp ; vergleiche Zaehler und Kanalwert
      brcs KeyLeds6 ; lasse LED an
      or R0,XH ; schalte Kanalausgang an = LED aus
KeyLeds6:
      lsr XH ; Maske eine Position rechts
      dec XL ; naechster Kanal
      brne KeyLeds5 ; noch ein Kanal
```

```
      out pLedO,R0 ; Setze Ausgabebits
      ret ; fertig
;
; Keys fragt die Eingabepins auf gedrueckte Tasten ab
;
Keys:
      in R1,pKeyO ; Rette aktuelle Statusbits (LEDs)
      in R0,pKeyD ; Sichere Richtungsbits
      mov rmp,R0 ; kopiere Richtungsbits
      andi rmp,cKeyM ; loesche Richtungsbits
      out pKeyD,rmp ; Tastenbits als Eingaenge
      mov rmp,R1 ; Pullup-Widerstaende
      ori rmp,cKeyB ; Bits auf Eins setzen
      out pKeyO,rmp ; an Port
      nop ; warte eine Weile
      nop
      nop
      nop
      in rmp,pKeyI ; Lese Tastenport
      out pKeyD,R0 ; Stelle Richtungsbits wieder her
      out pKeyO,R1 ; Stelle Ausgabezustand wieder her
      andi rmp,cKeyB ; Isoliere Tastenbits
      cpi rmp,cKeyB ; alle Tasten inaktiv?
      brne Keys2 ; nein
Keys1:
      clr rKeyRep ; Wiederholung auf Null
      mov rKeyLst,rmp ; speichern
      ret ; fertig
Keys2:
      cp rmp,rKeyLst ; gleich letztem Tastenstatus?
      brne Keys1 ; nein, loesche Zaehler
      inc rKeyRep ; erhoehe Zaehler
      ldi rmp,cKeyRep ; vergleiche mit vorgewaehlter Zahl
      cp rmp,rKeyRep
      breq Keys3 ; gleich, mach weiter
      ret
Keys3: ; Taste ok, starte ausgewaehlte Routine
      mov R0,rKeyLst ; lese die Tasten erneut
      ldi ZH,HIGH(sKeyJ1) ; zeige auf erste Adresse
      ldi ZL,LOW(sKeyJ1)
Keys4:
      rol R0
      brcc Keys5
      adiw ZL,2 ; naechste Adresse
      rjmp Keys4
Keys5:
      ld rmp,Z+ ; Lese Sprungadresse und springe
      ld R0,Z
      mov R1,R0 ; pruefe ob Adresse Null ist
      or R1,rmp
      breq Keys6
      push rmp
      push R0
Keys6:
      ret ; jump
;
; ================== Allgemeine Routinen fuer Menues ============
;
; Setze Position auf Menue-Anfang
;
```

```
KeyMStrt:
      push ZH ; sichere Z
      push ZL
      ldi ZH,3 ; Zeile 4
      ldi ZL,9 ; Position 10
      rcall LcdPos ; setze Position
      pop ZL ; stelle Z wieder her
      pop ZH
      ret
;
; loescht den Menueteil und setzt die Position auf den Anfang
;
KeyMClr:
      rcall KeyMStrt
      ldi rmp,' '
      mov R0,rmp
      ldi rmp,11
      mov R1,rmp
KeyMClr1:
      rcall LcdR0
      dec R1
      brne KeyMClr1
      rjmp KeyMStrt
;
; Erstes Menue in Zeile 4 ausgeben
;
KeyMenueFirst:
      rcall KeyMStrt ; Setze Position
      ldi ZH,HIGH(2*KeyTxtMenue)
      ldi ZL,LOW(2*KeyTxtMenue)
      rcall LcdDisplZ
      ldi rmp,17 ; Setze Cursor Position
      sts sLcdCs,rmp
      rjmp KeyReset ; initiiere Sprungadresse
; Erstes Menue an LCD
KeyTxtMenue:
.DB " Menue...",0
;
; Fehlermeldung falsche Kanalnummer ausgeben
;
KeyGetChError:
      rcall KeyMClr ; Menueteil loeschen
      ldi ZH,HIGH(2*KeyTxtChErr) ; Fehlertext
      ldi ZL,LOW(2*KeyTxtChErr)
      rcall LcdDisplZ ; zeige Text an
      ldi rmp,18 ; setze Cursor Position
      sts sLcdCs,rmp
      ret
;
; Hole Kanalnummer in R0
; Wenn Kanalnummer nicht zwischen 1 und 4 liegt, gib Fehlermeldung
; aus und kehre mit gesetztem Carry zurueck
;
KeyGetChR0:
      lds R0,sKeyCh ; Lese aktuellen Kanal
      tst R0 ; = Null?
      breq KeyGetChR0e
      push rmp
      ldi rmp,5
      cp R0,rmp ; > 4 ?
```

```
      pop rmp
      brcc KeyGetChR0e
      clc
      ret
KeyGetChR0e:
      rcall KeyGetChError
      sec
      ret
;
KeyTxtChErr:
.DB "Ch error!",0
;
; Setze gewaehlten Kanal auf den Wert n rmp
;
KeyChSet:
      rcall KeyGetChR0 ; Kanalnmmer in R0
      brcs KeyChSet3
      ldi ZL,0xFC ; Kanalmaskenbits = 0
KeyChSet1:
      dec R0 ; Kanal erreicht?
      breq KeyChSet2 ; ja
      sec ; setze Carry fuer Maskenrollen
      rol ZL ; naechster Kanal
      rol ZL
      lsl rmp ; schiebe rmp links
      lsl rmp
      rjmp KeyChSet1 ; noch mal
KeyChSet2:
      lds ZH,sState ; lade Statusbits
      and ZH,ZL ; loesche Kanalbits
      or ZH,rmp ; setze Kanalbits auf Wert in rmp
      sts sState,ZH ; setze neuen Status
KeyChSet3:
      ret
;
; Setze PWM-Wert des Kanals auf den Wert in rmp
;
KeySetPwm:
      rcall KeyGetChR0
      brcs KeySetPwm2
      ldi ZH,HIGH(cPwm1S-1) ; loesche PWM-Wert des Kanals
      ldi ZL,LOW(cPwm1S-1)
KeySetPwm1:
      adiw ZL,1 ; naechster PWM-Kanal
      dec R0
      brne KeySetPwm1
      st Z,rmp ; setze PWM-Wert
KeySetPwm2:
      ret
;
; Entlade-Bit des Kanals loeschen
;
KeyClrUnl:
      rcall KeyGetChR0
      brcs KeyClrUnl2
      ldi rmp,0xF7 ; Maske
KeyClrUnl1:
      sec ; rolle eine 1 ein
      rol rmp ; rotiere Maske links
      dec R0 ; naechster  Kanal
```

```
      brne KeyClrUnl1
      and rPwmOut,rmp ; loesche Entladebit
KeyClrUnl2:
      ret ; fertig
;
; Loesche Kapazitaetszaehler
;
KeyClrCap:
      rcall KeyGetChR0
      brcs KeyClrCap1
      dec R0
      ldi ZH,HIGH(sAdcC)
      ldi ZL,LOW(sAdcC)
      add ZL,R0
      add ZL,R0
      st Z+,rmp
      st Z+,rmp
      std Z+6,rmp
KeyClrCap1:
      ret
;
; Hole die Parameter des aktuellen Kanals, setze Adresse und hole
; Dezimalzahl, erwartet die Parameter-Basisadresse in X und den
; Maximalwert in Z
;
KeyGetParam:
      rcall KeyGetChR0
      brcs KeyGetParam1
      dec R0
      lsl R0 ; * 8
      lsl R0
      lsl R0
      add XL,R0 ; korrigiere Adresse
      clr R0
      adc XH,R0
      ld R0,X+ ; lese LSB
      sts sKeyNW,R0 ; speichere Dezimalzahl
      ld R0,X ; lese MSB
      sts sKeyNW+1,R0
      sts sKeyNM,ZL
      sts sKeyNM+1,ZH
KeyGetParam1:
      ret
;
; Setze die Parameter im aktuellen Kanal
; Erwartet Parameter-Basisadresse in X und Text in Z
;
KeySetParam:
      rcall KeyGetChR0
      brcs KeySetParam1
      dec R0
      lsl R0 ; * 8
      lsl R0
      lsl R0
      add XL,R0 ; korrigiere Adresse
      clr R0
      adc XH,R0
      lds rmp,sKeyNW
      st X+,rmp
      lds rmp,sKeyNW+1
```

```
      st X,rmp
      rcall KeyMClr
      rcall LcdDisplZ
      ldi ZH,HIGH(KeyM12a)
      ldi ZL,LOW(KeyM12a)
      ijmp
KeySetParam1:
      ret
;
; Nach Beenden eines Menueeintrags zeige Text in X an
;
KeyMDone:
      rcall KeyMClr ; Loesche Menuebereich der LCD
      rcall LcdDisplZ
      mov ZH,XH
      mov ZL,XL
      ijmp
;
; ================= Behandlung von Menueeintraegen ===========
;
; Ausgabe des Menueeintrags in X, des Menueeintrags in sKeyNC
;
KeyMOut:
      rcall KeyMClr
KeyMOut1:
      mov ZH,XH ; kopiere Menuebeginn nach Z
      mov ZL,XL
      lds R1,sKeyNC ; lese Menueeintragnummer
KeyMOut2: ; pruefe ob der Menueeintrag erreicht ist
      tst R1 ; Menueeintraszaehler = 0?
      brne KeyMOut5 ; nicht erreicht, weiter
KeyMOut3: ; Menueeintrag erreicht, Ausgabe des Menuetexts
      lpm
      adiw ZL,1
      mov rmp,R0
      cpi rmp,'#' ; Endezeichen
      breq KeyMOut4
      rcall LcdChar
      rjmp KeyMout3
KeyMOut4: ; Ausgabe beendet, suche NULL
      lpm
      adiw ZL,1
      tst R0
      brne KeyMOut4
      ; Lese die LCD Cursor position und die Sprungadresse
      lpm
      adiw ZL,1
      sts sLcdCs,R0
      lpm
      adiw ZL,1
      sts sKeyJ3,R0 ; speichere in Adresse fuer Taste 3
      lpm
      sts sKeyJ3+1,R0
      ret
KeyMOut5: ; nicht der gewueschte Eintrag suche naechsten Text
      lpm
      adiw ZL,1
      tst R0
      brne KeyMOut5
      adiw ZL,3
```

```
      lpm
      ldi rmp,0xFF
      cp rmp,R0
      breq KeyMOut6
      dec R1
      rjmp KeyMOut2
KeyMOut6: ; starte neu
      clr R1
      sts sKeyNC,R1
      rjmp KeyMOut1
;
; ================= Hole Dezimalzahl Eingabe ============
;
;
; Zeigt die Dezimalzal in sKeyNW am Ende von Zeile 4 der LCD an
;
KeyDecDispl: ; zeige Dezimalzahl an
      ldi ZH,3 ; Zeile 4
      ldi ZL,15 ; Position 16
      rcall LcdPos ; Setze LCD Position
      lds R2,sKeyNW ; lese Dezimalzahl
      lds R3,sKeyNW+1
      rcall LcdDec5 ; schreibe Zahl auf LCD
      lds ZL,sKeyNC ; setze LCD Cursor Position
      subi ZL,-15
      sts sLcdCs,ZL
      ldi ZH,3
      rjmp LcdPos
;
; Setze die Sprungadresse fuer die Dezimaleingabe und hole Zahl
; erwartet: Textzeiger in Z, Rueckadresse in X, Abbruchadresse
; in sKeyJ1
;
KeyDec:
      rcall KeyMClr ; loesche Menuebereich
      rcall LcdDisplZ ; Textausgabe
      ldi ZH,HIGH(sKeyJR) ; auf Ruecksprungadresse zeigen
      ldi ZL,LOW(sKeyJR)
      st Z+,XL
      st Z+,XH
      lds XL,sKeyJ1 ; speichere Ruecksprungsadresse
      lds XH,sKeyJ1+1
      st Z+,XL
      st Z+,XH
      ldi XL,LOW(KeyDecL) ; Taste 1 Adresse
      ldi XH,HIGH(KeyDecL)
      st Z+,XL
      st Z+,XH
      ldi XL,LOW(KeyDecU) ; Taste 2 Adresse
      ldi XH,HIGH(KeyDecU)
      st Z+,XL
      st Z+,XH
      ldi XL,LOW(KeyDecR) ; Taste 3 Adresse
      ldi XH,HIGH(KeyDecR)
      st Z+,XL
      st Z+,XH
      ldi rmp,4 ; Cursor auf letzte Ziffer
      sts sKeyNC,rmp
      rjmp KeyDecDispl ; Zeige Zahl an und setze Cursor
;
```

```
KeyDecL: ; Linke Taste
      lds rmp,sKeyNC ; Cursor Position eins links
      dec rmp
      cpi rmp,0xFF ; ueber Anfang hinaus?
      breq KeyDecL1
      sts sKeyNC,rmp ; setze Cursor
      rjmp KeyDecDispl ; Zahlanzeige und Cursor setzen
KeyDecL1:
      lds ZL,sKeyJS ; lese Abbruchadresse
      lds ZH,sKeyJS+1
      ijmp ; springe zur Abbruchroutine
;
KeyDecU: ; Mittlere Taste
      ldi ZH,HIGH(2*KeyDecTbl) ; Z auf Dezimaltabelle
      ldi ZL,LOW(2*KeyDecTbl)
      lds rmp,sKeyNC ; plus aktuelle Position
      lsl rmp ; * 2
      add ZL,rmp ; zeige auf Wert in Dezimaltabelle
      clr rmp
      adc ZH,rmp
      lpm ; lese Dezimalwert
      mov XL,R0
      adiw ZL,1
      lpm
      mov XH,R0
      lds ZL,sKeyNW ; lese aktuelle Dezimalzahl
      lds ZH,sKeyNW+1
      add ZL,XL ; Addiere beide
      adc ZH,XH
      lds XL,sKeyNM ; lese Maximalzahl
      lds XH,sKeyNM+1
      sec
      cpc ZL,XL ; vergleiche mit Maximum
      cpc ZH,XH
      brcs KeyDecU1
      clr ZH ; > Max., loesche Zahl
      clr ZL
KeyDecU1:
      sts sKeyNW,ZL ; speichere Zahl
      sts sKeyNW+1,ZH
      rjmp KeyDecDispl ; zeige Dezimalzahl an, setze Cursor
;
KeyDecR: ; Rechte Taste
      lds rmp,sKeyNC ; lese Cursor
      inc rmp ; naechste Position
      cpi rmp,5 ; rechts raus?
      brcc KeyDecR1 ; ja
      sts sKeyNC,rmp ; speichere Position
      rjmp KeyDecDispl
KeyDecR1:
      lds ZL,sKeyJR ; lese Sprungadresse
      lds ZH,sKeyJR+1
      ijmp ; springe zum Beenden der Eingabe
;
KeyDecTbl: ; Dezimaltabelle
.DW 10000
.DW 1000
.DW 100
.DW 10
.DW 1
```

```
;
; ================= Menues ====================
;
; Menue 1, Auswahl
;
KeyM1:
      ldi XH,HIGH(2*KeyTxtM1) ; Menuetext
      ldi XL,LOW(2*KeyTxtM1)
      rcall KeyMOut ; zeige Menuetext an
KeyM1a:
      ldi rmp,LOW(KeyMenueFirst) ; Setze linke Sprungadresse
      sts sKeyJ1,rmp
      ldi rmp,HIGH(KeyMenueFirst)
      sts sKeyJ1+1,rmp
      ldi rmp,LOW(KeyM1Up) ; Setze mittlere Sprungadresse
      sts sKeyJ2,rmp
      ldi rmp,HIGH(KeyM1Up)
      sts sKeyJ2+1,rmp
      clr rmp ; beginne mit Menueeintrag 0
      sts sKeyNC,rmp
      ret
;
; Menue 1 Erhoehen Routine
;
KeyM1Up:
      lds rmp,sKeyNC ; naechst hoeherer Menueeintrag
      inc rmp
      sts sKeyNC,rmp
      ldi XH,HIGH(2*KeyTxtM1) ; zeige mit X auf Menuetabelle
      ldi XL,LOW(2*KeyTxtM1)
      rjmp KeyMOut
;
; Menue 11: Auswahl von Akku-Id und Kanal
;
KeyM11:
      clr rmp ; Setze ausgewaehlten Menueeintrag auf 0
      sts sKeyNC,rmp
      ldi XH,HIGH(2*KeyTxtM11) ; Zeiger X auf Menuetabelle
      ldi XL,LOW(2*KeyTxtM11)
      rcall KeyMOut ; Zeige Menueeintrag an
KeyM11a:
      ldi rmp,LOW(KeyM1) ; Setze Sprungadresse fuer linke Taste auf Menue 1
      sts sKeyJ1,rmp
      ldi rmp,HIGH(KeyM1)
      sts sKeyJ1+1,rmp
      ldi rmp,LOW(KeyM11Up) ; Setze mittlere Sprungadresse auf M11Up
      sts sKeyJ2,rmp
      ldi rmp,HIGH(KeyM11Up)
      sts sKeyJ2+1,rmp
      ret
;
; Menue 11 Up Tastenroutine
;
KeyM11Up:
      lds rmp,sKeyNC ; naechster Menueeintrag
      inc rmp
      sts sKeyNC,rmp
      ldi XH,HIGH(2*KeyTxtM11) ; ausgewaehlter Menuetext
      ldi XL,LOW(2*KeyTxtM11)
      rjmp KeyMOut ; Ausgabe des Menueeintrags
```

```
;
; Tastenauswahl-Routinen
;
KeyM111: ; Waehle Akku-Id
      rcall KeyGetChR0 ; lese Kanal
      brcs KeyM111e
      dec R0
      ldi ZH,HIGH(sAdcAI1) ; lese Akku-Id
      ldi ZL,LOW(sAdcAI1)
      add ZL,R0
      add ZL,R0
      ld rmp,Z
      sts sKeyNW,rmp
      clr rmp
      sts sKeyNW+1,rmp
      lds rmp,sEANm ; lese Anzahl der Id's
      sts sKeyNM,rmp ; setze Maximaleingabe
      clr rmp
      sts sKeyNM+1,rmp
      ldi XH,HIGH(KeyM111R) ; Rueecksprungadresse in X
      ldi XL,LOW(KeyM111R)
      ldi ZH,HIGH(2*KeyTxtM111) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM111)
      rjmp KeyDec ; hole Dezimalzahl
KeyM111e:
      ret
;
KeyTxtM111:
.DB "Akku=",0
;
; Rueckkehr von der Dezimalahleingabe mit Akku-Id
;
KeyM111R:
      rcall KeyGetChR0
      brcs KeyM111Re
      dec R0
      ldi ZH,HIGH(sAdcAI1)
      ldi ZL,LOW(sAdcAI1)
      add ZL,R0
      add ZL,R0
      lds rmp,sKeyNW ; kopiere Dezimalzahl in Kanal
      st Z,rmp
      rcall IdSetPar ; Setze Parameter
      ldi XH,HIGH(KeyM11a) ; Sprung zurueck
      ldi XL,LOW(KeyM11a)
      ldi ZH,HIGH(2*KeyTxt112R)
      ldi ZL,LOW(2*KeyTxt112R)
      rjmp KeyMDone
KeyM111Re:
      ret
;
KeyTxt111R:
.DB "Akku-Id ok",0,0
;
KeyM112: ; Waehle Kanalnummer aus
      lds rmp,sKeyCh ; lese Kanal, kopiere nach Dezimalzahleingabe
      sts sKeyNW,rmp
      clr rmp
      sts sKeyNW+1,rmp
      ldi rmp,4 ; Setze Maximumwert
```

```
      sts sKeyNM,rmp
      clr rmp
      sts sKeyNM+1,rmp
      ldi XH,HIGH(KeyM112R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM112R)
      ldi ZH,HIGH(2*KeyTxtM112) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM112)
      rjmp KeyDec ; hole Dezimalzahl
;
KeyTxtM112:
.DB "Kanal=",0,0
;
; Setze Kanal auf Eingabezahl
;
KeyM112R:
      lds rmp,sKeyNW ; kopiere Eingabezahl in Kanal
      sts sKeyCh,rmp
      ldi XH,HIGH(KeyM11a) ; springe zurueck
      ldi XL,LOW(KeyM11a)
      ldi ZH,HIGH(2*KeyTxt112R)
      ldi ZL,LOW(2*KeyTxt112R)
      rjmp KeyMDone
;
KeyTxt112R:
.DB "Kanal ok",0,0
;
KeyM113: ; Direkt laden
      ldi rmp,LOW(500)
      sts sKeyNW,rmp
      ldi rmp,HIGH(500)
      sts sKeyNW+1,rmp
      ldi rmp,LOW(9999)
      sts sKeyNM,rmp
      ldi rmp,HIGH(9999)
      sts sKeyNM+1,rmp
      ldi XH,HIGH(KeyM113R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM113R)
      ldi ZH,HIGH(2*KeyTxtM113) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM113)
      rjmp KeyDec ; hole Dezimalzahl
;
KeyTxtM113:
.DB "Kapaz.=",0
;
KeyM113R:
      rcall KeyGetChR0
      brcs KeyM113Re
      rcall KeyClrUnl
      ldi rmp,0x02 ; setze Ladestatus
      rcall KeyChSet
      ldi rmp,128
      rcall KeySetPwm
      rcall KeyGetChR0
      dec R0
      lsl R0
      lsl R0
      lsl R0
      ldi ZH,HIGH(sCh1LC) ; zeige auf Ladestrom
      ldi ZL,LOW(sCh1LC)
      add ZL,R0
```

```
	clr R0
	adc ZH,R0
	lds XL,sKeyNW ; lese Kapazitaet
	lds XH,sKeyNW+1
	mov R2,XL ; kopiere
	mov R3,XH
	lsr R3 ; / 2
	ror R2
	lsr R3 ; / 4
	ror R2
	lsr R3 ; / 8
	ror R2
	st Z+,R2 ; setze Ladestrom
	st Z+,R3
	add XL,R2 ; addiere zur Nennkapazitaet
	adc XH,R3
	st Z+,XL ; setze Kapazitaet
	st Z+,XH
	lsl XL ; * 2
	rol XH
	clr rmp
	rol rmp
	st Z+,XH ; setze den Erhaltungsstrom
	st Z,rmp
	ldi XH,HIGH(KeyM11a) ; Ruecksprungadresse
	ldi XL,LOW(KeyM11a)
	ldi ZH,HIGH(2*KeyTxt113D) ; Textausgabe
	ldi ZL,LOW(2*KeyTxt113D)
	rjmp KeyMDone
KeyM113Re:
	ret
;
KeyTxt113D:
.DB "Autoladen",0
;
; Menue "Setze" Routinen
;
KeyM12:
	clr rmp ; Setze gewaehlten Menueeintrag auf 0
	sts sKeyNC,rmp
	ldi XH,HIGH(2*KeyTxtM12) ; X zeigt auf Menuetabelle
	ldi XL,LOW(2*KeyTxtM12)
	rcall KeyMOut ; Zeige Menueeintrag an
KeyM12a:
	ldi rmp,LOW(KeyM1) ; Setze linke Sprungadresse auf Menue 1
	sts sKeyJ1,rmp
	ldi rmp,HIGH(KeyM1)
	sts sKeyJ1+1,rmp
	ldi rmp,LOW(KeyM12Up) ; Setze mittlere Sprungadresse auf M12Up
	sts sKeyJ2,rmp
	ldi rmp,HIGH(KeyM12Up)
	sts sKeyJ2+1,rmp
	ret
;
; Menue 12 Up Tastenroutine
;
KeyM12Up:
	lds rmp,sKeyNC ; naechster Menueeintrag
	inc rmp
	sts sKeyNC,rmp
```

```
      ldi XH,HIGH(2*KeyTxtM12) ; gewaehlter Menuetext
      ldi XL,LOW(2*KeyTxtM12)
      rjmp KeyMOut ; gib Menueeintrag aus
;
KeyM121:
      ldi XH,HIGH(sCh1UV) ; Kanalnummer auf Entladespannung
      ldi XL,LOW(sCh1UV)
      ldi ZH,HIGH(1300) ; Maximalwert setzen
      ldi ZL,LOW(1300)
      rcall KeyGetParam ; Parameter in X auf Eingabe und Maximum auf Z
      ldi XH,HIGH(KeyM121R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM121R)
      ldi ZH,HIGH(2*KeyTxtM121) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM121)
      rjmp KeyDec; hole Dezimalzahl
;
KeyM122:
      ldi XH,HIGH(sCh1LC) ; Kanalnummer auf Ladestrom
      ldi XL,LOW(sCh1LC)
      ldi ZH,HIGH(350) ; Maximalwert
      ldi ZL,LOW(350)
      rcall KeyGetParam ; Parameter in X auf Eingabe und Maximum auf Z
      ldi XH,HIGH(KeyM122R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM122R)
      ldi ZH,HIGH(2*KeyTxtM122) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM122)
      rjmp KeyDec; Hole Dezimalzahl
;
KeyM123:
      ldi XH,HIGH(sCh1LW) ; Kanalnummer auf Ladekapazitaet
      ldi XL,LOW(sCh1LW)
      ldi ZH,HIGH(9999) ; Maximawert
      ldi ZL,LOW(9999)
      rcall KeyGetParam ; Parameter in X auf Eingabe und Maximum auf Z
      ldi XH,HIGH(KeyM123R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM123R)
      ldi ZH,HIGH(2*KeyTxtM123) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM123)
      rjmp KeyDec; hole Dezimalzahl
;
KeyM124:
      ldi XH,HIGH(sCh1MC) ; Kanalnummer auf Erhaltungsstrom
      ldi XL,LOW(sCh1MC)
      ldi ZH,HIGH(100) ; Maximalwert
      ldi ZL,LOW(100)
      rcall KeyGetParam ; Parameter in X auf Eingabe und Maximum af Z
      ldi XH,HIGH(KeyM124R) ; Ruecksprungadresse in X
      ldi XL,LOW(KeyM124R)
      ldi ZH,HIGH(2*KeyTxtM124) ; Textadresse in Z
      ldi ZL,LOW(2*KeyTxtM124)
      rjmp KeyDec; Hole Dezimalzahl
;
KeyTxtM121:
.DB "Entlade=",0,0
;
KeyTxtM122:
.DB "Laden I=",0,0
;
KeyTxtM123:
.DB "Kapaz.=",0
```

```
;
KeyTxtM124:
.DB "Erhalt.=",0,0
;
; Setze die Entladespannung auf den Wert sKeyNW
;
KeyM121R:
      ldi XH,HIGH(sCh1UV) ; X auf Entladespannung
      ldi XL,LOW(sCh1UV)
      ldi ZH,HIGH(2*KeyTxtM121D) ; Z auf Text
      ldi ZL,LOW(2*KeyTxtM121D)
      rjmp KeySetParam ; setze Parameter
;
; Setze den Ladestrom auf den Wert in sKeyNW
;
KeyM122R:
      ldi XH,HIGH(sCh1LC) ; X auf Ladestrom
      ldi XL,LOW(sCh1LC)
      ldi ZH,HIGH(2*KeyTxtM122D) ; Z auf Text
      ldi ZL,LOW(2*KeyTxtM122D)
      rjmp KeySetParam ; setze Parameter
;
; Setze die Nennkapazitaet auf den Wert in sKeyNW
;
KeyM123R:
      ldi XH,HIGH(sCh1LW) ; X auf Kapazitaet
      ldi XL,LOW(sCh1LW)
      ldi ZH,HIGH(2*KeyTxtM123D) ; Z auf Text
      ldi ZL,LOW(2*KeyTxtM123D)
      rjmp KeySetParam ; setze Parameter
;
; Setze den Erhaltungsstrom auf den Wert in sKeyNW
;
KeyM124R:
      ldi XH,HIGH(sCh1MC) ; X auf Erhaltungsstrom
      ldi XL,LOW(sCh1MC)
      ldi ZH,HIGH(2*KeyTxtM124D) ; Z auf Text
      ldi ZL,LOW(2*KeyTxtM124D)
      rjmp KeySetParam ; setze Parameter
;
KeyTxtM121D:
.DB "Entladen ok",0
;
KeyTxtM122D:
.DB "Laden ok",0,0
;
KeyTxtM123D:
.DB "Kapaz. ok",0
;
KeyTxtM124D:
.DB "Erhalt. ok",0,0
;
;
; Menueeintraege Befehl ausfuehren
;
KeyM13:
      clr rmp
      sts sKeyNC,rmp
      ldi XH,HIGH(2*KeyTxtM13) ; Menuetext
      ldi XL,LOW(2*KeyTxtM13)
```

```
      rcall KeyMOut ; Zeige Menue an
KeyM13a:
      ldi rmp,LOW(KeyM1) ; Setze linke Sprungadresse
      sts sKeyJ1,rmp
      ldi rmp,HIGH(KeyM1)
      sts sKeyJ1+1,rmp
      ldi rmp,LOW(KeyM13Up) ; Setze mittlere Sprungadresse
      sts sKeyJ2,rmp
      ldi rmp,HIGH(KeyM13Up)
      sts sKeyJ2+1,rmp
      ret
;
; Menue 13 Up Tastenroutine
;
KeyM13Up:
      lds rmp,sKeyNC
      inc rmp
      sts sKeyNC,rmp
      ldi XH,HIGH(2*KeyTxtM13) ; Menuetext
      ldi XL,LOW(2*KeyTxtM13)
      rjmp KeyMOut
;
; Befehls-Menues, setzt Status auf Wert in sKeyCh
;
KeyM131: ; setze sState auf Entladen
      ldi rmp,0x01
      rcall KeyChSet
      clr rmp
      rcall KeySetPwm
      rcall KeyClrCap
      ldi rmp,0x08 ; setze sState auf Entladen
      lds R0,sKeyCh
KeyM131b:
      lsl rmp
      dec R0
      brne KeyM131b
      or rPwmOut,rmp ; setze Entladebit des PWM-Kanals
      ldi XH,HIGH(KeyM13a)
      ldi XL,LOW(KeyM13a)
      ldi ZH,HIGH(2*KeyTxtM131D)
      ldi ZL,LOW(2*KeyTxtM131D)
      rjmp KeyMDone
;
KeyM132: ; setze sState auf Laden
      ldi rmp,0x01
      rcall KeyChSet
      rcall KeyClrUnl ; loesche Entladebit
      rcall KeyClrCap
      ldi rmp,128 ; setze PWM auf 1100 mV
      rcall KeySetPwm
      ldi XH,HIGH(KeyM13a)
      ldi XL,LOW(KeyM13a)
      ldi ZH,HIGH(2*KeyTxtM132D)
      ldi ZL,LOW(2*KeyTxtM132D)
      rjmp KeyMDone
;
KeyM133: ; setze sState auf Erhaltungsladung
      ldi rmp,0x01
      rcall KeyChSet
      rcall KeyClrUnl ; loesche Entladebit
```

```
      ldi rmp,128 ; setze die PWM auf 1100 mV
      rcall KeySetPwm
      ldi XH,HIGH(KeyM13a)
      ldi XL,LOW(KeyM13a)
      ldi ZH,HIGH(2*KeyTxtM133D)
      ldi ZL,LOW(2*KeyTxtM133D)
      rjmp KeyMDone
;
KeyM134: ; loesche sState (Kanal aus)
      clr rmp
      rcall KeyChSet
      rcall KeyClrUnl ; loesche Entladebit
      clr rmp ; PWM auf Null
      rcall KeySetPwm
      rcall KeyClrCap ; Kapazitaet loeschen
      ldi XH,HIGH(KeyM13a)
      ldi XL,LOW(KeyM13a)
      ldi ZH,HIGH(2*KeyTxtM134D)
      ldi ZL,LOW(2*KeyTxtM134D)
      rjmp KeyMDone
;
KeyTxtM131D:
.DB "Entladen",0,0
KeyTxtM132D:
.DB "Laden",0
KeyTxtM133D:
.DB "Erhalten",0,0
KeyTxtM134D:
.DB "Aus",0
;
; ================== Menuetabellen ==================
;
; Menue 1
;
KeyTxtM1:
.DB "Auswahl...# ",0,19
.DW KeyM11
.DB "Setze...# ",0,17
.DW KeyM12
.DB "Befehl...#",0,18
.DW KeyM13
.DW 0xFFFF
;
KeyTxtM11:
.DB "Akku-Id...# ",0,19
.DW KeyM111
.DB "Kanal...# ",0,17
.DW KeyM112
.DB "Direktlade# ",0,19
.DW KeyM113
.DW 0xFFFF
;
KeyTxtM12:
.DB "Entlade...# ",0,19
.DW KeyM121
.DB "Laden...# ",0,16
.DW KeyM122
.DB "Kapazit...# ",0,19
.DW KeyM123
.DB "Erhaltg...# ",0,19
```

```
.DW KeyM124
.DW 0xFFFF
;
KeyTxtM13:
.DB "Entlade...# ",0,19
.DW KeyM131
.DB "Lade...#",0,16
.DW KeyM132
.DB "Erhaltg...# ",0,19
.DW KeyM133
.DB "Aus...# ",0,15
.DW KeyM134
.DW 0xFFFF
;
```

Include-Datei akkulcd.asm

```
; ************************************************************
; * Include-Datei fuer LCD-Routinen Akkuload, Version 0.1 02/05*
; * (C)2005 by Gerhard Schmidt, info@avr-asm-tutorial.net      *
; ************************************************************
;
; -------------------------------------------------------------
; Routine   Register   Funktion
; -------------------------------------------------------------
; LcdInit   rmp,R0,Z   Initiiert das LCD-Display zu Beginn auf:
;                      8-Bit-Interface, 4 Zeilen, kein Shift,
;                      aktiver Cursor und blinkend
; LcdR0     rmp        gibt das Zeichen in R0 auf der aktuellen
;                      Position des LCD aus, R0 bleibt erhalten
; LcdChar   rmp,R0     gibt das Zeichen in rmp auf der aktuellen
;                      Position des LCD aus, rmp wird geaendert,
;                      Zeichen bleibt in R0 erhalten
; LcdDisplZ rmp,R0,Z   gibt den Text im Flash ab Z auf der LCD
;                      aus, stoppt beim NULL-Zeichen
; LcdLine   rmp        setzt die Ausgabe der LCD auf die Zeile
;                      in rmp, erwartet 0..3 in rmp fuer die
;                      Zeilen 1..4
; LcdLineCl rmp,R0,Z   loescht die Zeile des LCD, auf die rmp
;                      zeigt, durch Schreiben von Leerzeichen
; LcdPos    rmp,Z      setzt die LCD auf die Position in Z,
;                      ZH ist die Zeile, ZL ist die Position in
;                      dieser Zeile (0..19)
; LcdMClr   rmp        loescht den Menueteil und setzt die
;                      Position in dieser Zeile auf den Anfang
; LcdMStrt  rmp,R0,Z   Setzt die LCD Cursor Position auf den
;                      Menueanfang
; LcdCursor rmp,Z      Setzt den LCD-Cursor auf seine korrekte
;                      Position nach Ausgaben,Position<8: zeigt
;                      Menueausgabe 1 an
; LcdHex2   rmp        Schreibt den Inhalt von rmp in Hex an die
;                      aktuelle LCD-Position (debug)
; LcdDec5   rmp,R0,Z   schreibt die Binaerzahl in R3:R2 als
; LcdDec4              Dezimalzahl an die aktuelle LCD-Position,
; LcdDec3              5, 4, 3 oder 2 Ziffern werden angezeigt
; LcdDec2
; -------------------------------------------------------------
;
; Konstanten
;
.EQU pLcdCO = PORTD
.EQU pLcdCD = DDRD
.EQU pLcdCI = PIND
.EQU pLcdDO = PORTB
.EQU pLcdDD = DDRB
.EQU pLcdDI = PINB
.EQU bLcdE = 2 ; E-Eingang des Displays
.EQU bLcdRw = 3 ; Read/Write Kontrolle des Display
.EQU bLcdRs = 4 ; Register Select des Display
;
; Verzoegerungsroutinen fuer die Display-Initiierung
; Anzahl der Takte einschliesslich rcall: 3+7+4*Z-4+7 = 4*Z+13
; Verzoegerung in Mikrosekunden: (4*Z+13)/8
; Berechnung von Z = (8*t[µs]-13)/4
;
```

```
; Verzoegerung 15 ms
;
.EQU cLcdWt15000 = (8*15000-13)/4 ; = 29996
LcdWt15000:
      ldi ZH,HIGH(cLcdWt15000) ; 1
      ldi ZL,LOW(cLcdWt15000) ; 2
      rjmp LcdWt ; 4
;
; Verzoegerung 4,1 ms
;
.EQU cLcdWt4100 = (8*4100-13)/4 ; = 8196
LcdWt4100:
      ldi ZH,HIGH(cLcdWt4100) ; 1
      ldi ZL,LOW(cLcdWt4100)
      rjmp LcdWt
;
; Verzoegerung 4,5 ms
;
.EQU cLcdWt4500 = (4500*8-13)/4
LcdWt4500:
      ldi ZH,HIGH(cLcdWt4500) ; 1
      ldi ZL,LOW(cLcdWt4500) ; 2
      rjmp LcdWt ; 4
;
; Verzoegerung 1,64 ms
;
.EQU cLcdWt1640 = (1640*8-13)/4 ; = 3276
LcdWt1640:
      ldi ZH,HIGH(cLcdWt1640) ; 1
      ldi ZL,LOW(cLcdWt1640) ; 2
      rjmp LcdWt ; 4
;
; Verzegerung 100 µs
;
.EQU cLcdWt100 = (100*8-13)/4
LcdWt100:
      ldi ZH,HIGH(cLcdWt100)
      ldi ZL,LOW(cLcdWt100)
      rjmp LcdWt
;
; Verzoegerung 40 µs
;
.EQU cLcdWt40 = (40*8-13)/4 ; = 76
LcdWt40:
      ldi ZH,HIGH(cLcdWt40) ; 1
      ldi ZL,LOW(cLcdWt40) ; 2
      nop ; 3
      nop ; 4
;
; Wartet auf Z Verzoegerung
; Anzahl Takte einschlieszlich ret = 4*(Z-1)+7
; Verzoegerung in µs bei 8 MHz: (4*(Z-1)+7)/8
;
LcdWt:
      sbiw ZL,1 ; 2
      brne LcdWt ; 2/1
      ret ; 4
;
; Verzoegerung fuer einen aktiven E-Puls ist 1 µs
;
```

```
LcdDel1:
      nop
      ret
;
; Aktiviere E fuer 1 µs
;
LcdE:
      sbi pLcdCO,bLcdE ; setze E aktiv
      rcall LcdDel1 ; 1 µs Verzoegerung
      cbi pLcdCO,bLcdE ; loesche E
      ret
;
; Pruefe ob die LCD busy ist nach Ausgabe eines Zeichens
; stellt zu Beginn fest, ob die LCD angeschlossen ist
;
LcdChck:
      cbi pLcdCO,bLcdRw ; Schreibe Daten an LCD
      sbi pLcdCO,bLcdRs ; setze RS-Eingang aktiv
      clr rmp ; alle Datenbits auf Eingang
      out pLcdDD,rmp
      ser rmp ; Pullup-Widerstaende einschalten
      out pLcdDO,rmp
      cbi pLcdCO,bLcdRs ; loesche Rs
      sbi pLcdCO,bLcdRw ; setze RW
      sbi pLcdCO,bLcdE ; activiere E
      in rmp,pLcdDI ; lese Display
      tst rmp ; springe wenn Adresse und Busy nicht Null sind
      sbr rFlg,1<<bLcdOk ; setze die LCD-ok-Flagge
      cbi pLcdCO,bLcdE ; loesche E
      cbi pLcdCO,bLcdRw ; loesche RW
      ret
;
; Funktionseinstellung der LCD zu Beginn
;
LcdInitFunct:
      cbi pLcdCO,bLcdRs ; loesche RS
      cbi pLcdCO,bLcdRw ; loesche RW
      ldi rmp,0x38 ; Setze 8 bit interface, 2 Zeilen
      out pLcdDO,rmp ; an Datenausgang
      rcall LcdE
      rcall LcdWt4100 ; warte 4100 s
      rcall LcdE
      rcall LcdWt100 ; warte 100 s
      rcall LcdE
      rcall LcdWt100 ; warte 100 s
      ldi rmp,0x08 ; Display aus
      out pLcdDO,rmp
      rcall LcdE
      rcall LcdWt40 ; warte 40 s
      ldi rmp,0x01 ; loesche Display
      out pLcdDO,rmp
      rcall LcdE
      rcall LcdWt4500 ; warte 4,5 ms
      ldi rmp,0x06 ; Eingabemodus, erhoehen, keine Shift
      out pLcdDO,rmp
      rcall LcdE
      rcall LcdWt40
      ldi rmp,0x10 ; Display Cursor Shift
      out pLcdDO,rmp
      rcall LcdE
```

```
      rcall LcdWt40 ; warte 40 s
      ldi rmp,0x02 ; Display/Cursor Home
      out pLcdDO,rmp
      rcall LcdE
      rcall LcdWt1640 ; warte fuer 1,64 ms
      ldi rmp,0x0F ; Display aktiv, Cursor an, blinkend
      out pLcdDO,rmp
      rcall LcdE
      rcall LcdWt40 ; warte 40 s
      rjmp LcdChck ; pruefe die LCD
;
; Warte bis die LCD nicht mehr busy ist
;
LcdBusy:
      clr rmp ; Setze Datenport auf Eingang
      out pLcdDD,rmp
      cbi pLcdCO,bLcdRs ; Loesche RS
      sbi pLcdCO,bLcdRw ; Setze Display auf Lesen
      sbi pLcdCO,bLcdE ; Enable LCD
LcdBusy1:
      nop ; warte ein bisschen
      nop ; noch ein bisschen
      sbic pLcdDI,7 ; springe wenn busy-Bit Null ist
      rjmp LcdBusy1 ; warte weiter
      cbi pLcdCO,bLcdE ; Loesche E
      ret
;
; Sende Kontrollanweisng in rmp an LCD
;
LcdCtrl:
      push rmp ; sichere rmp
      rcall LcdBusy ; warte auf nicht mehr busy
      ldi rmp,0xFF ; Setze Datenport auf Ausgang
      out pLcdDD,rmp
      pop rmp
      out pLcdDO,rmp ; Daten an Port
      cbi pLcdCO,bLcdRs ; loesche RS
      cbi pLcdCO,bLcdRw ; loesche R/W
      rcall LcdE ; aktiviere E fuer 1 s
      clr rmp ; Datenport aus
      out pLcdDD,rmp
      ret
;
; Schreibe Zeichen in rmp an LCD
;
LcdChar:
      mov R0,rmp ; kopiere Zeichen in R0
;
; Schreibe Zeichen in R0 an LCD
;
LcdR0:
      rcall LcdBusy
      ldi rmp,0xFF ; Datenausgaberichtung
      out pLcdDD,rmp
      out pLcdDO,R0 ; Zeichen auf Ausgabeport
      sbi pLcdCO,bLcdRs ; Setze RS
      cbi pLcdCO,bLcdRw ; Loesche R/W
      rcall LcdE ; aktiviere E fuer 1 s
      clr rmp ; deaktiviere Datenport
      out pLcdDD,rmp
```

```
      ret
;
; Schreibe Text im Flash ab Z auf Display
;
LcdDisplZ:
      lpm ; lese ein Zeichen aus Flash
      adiw ZL,1
      tst R0 ; NULL-Zeichen?
      breq LcdDisplZ1
      rcall LcdR0 ; Zeichen an Display
      rjmp LcdDisplZ
LcdDisplZ1:
      ret
;
; Initiiere die LCD
;
LcdInit:
      ser rmp ; Setze Datenport alle als Ausgang
      out pLcdDD,rmp
      cbi pLcdCO,bLcdE ; Bit E auf Null
      sbi pLcdCD,bLcdE ; Richtungsbit E als Ausgang
      cbi pLCDCO,bLcdRw ; Bit R/W auf Null
      sbi pLcdCD,bLcdRw ; Richtungsbit R/W als Ausgang
      cbi pLcdCO,bLcdRs ; Bit RS auf Null
      sbi pLcdCD,bLcdRs ; Richtungsbit RS als Ausgang
      rcall LcdWt15000 ; Warten fuer 15 ms
      rcall LcdInitFunct ; Setze 8-bit interface und Anzahl Zeilen
      rcall LcdChck ; pruefe die LCD
      sbrs rFlg,bLcdOk ; springe wenn LCD ok
      ret
      clr rmp ; setze LCD auf Zeile 1
      rcall LcdLine
      ldi ZH,HIGH(2*LcdTxtInit1) ; Z auf Text
      ldi ZL,LOW(2*LcdTxtInit1)
      rcall LcdDisplZ ; schreibe Text auf LCD
      ldi rmp,1 ; setze LCD-Zeile 2
      rcall LcdLine
      rcall LcdDisplZ
      ldi rmp,2 ; setze LCD-Zeile 3
      rcall LcdLine
      rcall LcdDisplZ
      ldi rmp,3 ; setze LCD-Zeile 4
      rcall LcdLine
      rjmp LcdDisplZ
;
; LcdLine setzt den Eingabecursor in die ausgewahlte Zeile in rmp
;
LcdLine:
      cpi rmp,1 ; rmp = 0 oder 1
      brcs LcdLine1 ; rmp=0, setzt Zeile 1
      breq LcdLine2 ; rmp=1, setzt Zeile 2
      cpi rmp,2
      breq LcdLine3 ; rmp=2, setzt Zeile 3
      ldi rmp,0x54 ; rmp>2, setzt Zeile 4
      rjmp LcdLine4
LcdLine1:
      clr rmp ; Zeile 1 startet bei 0
      rjmp LcdLine4
LcdLine2:
      ldi rmp,0x40 ; Zeile 2 startet bei 0x40
```

```
      rjmp LcdLine4
LcdLine3:
      ldi rmp,20 ; Zeile 3 startet bei 20
LcdLine4:
      sbr rmp,0x80 ; setze Kontrollbit
      rjmp LcdCtrl
;
; LCD loesche Zeile
;
LcdLineCl:
      mov ZH,rmp ; sichere Zeilennummer
      rcall LcdLine ; setze Zeile
      ldi rmp,' ' ; Leerzeichen
      mov R0,rmp ; nach R0
      ldi ZL,20 ; ZL ist Zaehler
LcdLineCl1:
      rcall LcdR0 ; schreibe Zeichen in R0
      dec ZL ; naechstes
      brne LcdLineCl1 ; noch mal
      mov rmp,ZH ; stelle rmp wieder her
      rjmp LcdLine
;
; LcdPos setzt die LCD auf die Position in Z
;
LcdPos:
      cpi ZH,1 ; Zeile = 0 oder 1
      brcs LcdPos1 ; ZH=0, setzt Zeile 1
      breq LcdPos2 ; ZH=1, setzt Zeile 2
      cpi ZH,2
      breq LcdPos3 ; ZH=2, setzt Zeile 3
      ldi rmp,0x54 ; ZH>2, setzt Zeile 4
      rjmp LcdPos4
LcdPos1:
      clr rmp ; Zeile 1 startet bei 0
      rjmp LcdPos4
LcdPos2:
      ldi rmp,0x40 ; Zeile 2 startet bei 0x40
      rjmp LcdPos4
LcdPos3:
      ldi rmp,20 ; Zeile 3 startet bei 20
LcdPos4:
      add rmp,ZL ; addiere Position in Zeile
      sbr rmp,0x80
      rjmp LcdCtrl
;
; Setzt die LCD Cursor Position
;
LcdCursor:
      sbrs rFlg,bLcdOk ; LCD ok?
      ret
      lds rmp,sLcdCs ; lese Cursor-Position
      cpi rmp,8 ; mindest-Position
      brcc LcdCursor1
      rcall KeyMenueFirst ; Zeige Menue in Zeile an
LcdCursor1:
      ldi ZH,3 ; Zeile 4
      lds ZL,sLcdCs ; Lese erneut Position
      rjmp LcdPos
;
; LCDHEX2 schreibt den Inhalt von rmp in Hex (debug)
```

```
;
LcdHex2:
      push rmp
      swap rmp
      rcall LcdHexN
      pop rmp
LcdHexN:
      andi rmp,0x0F
      cpi rmp,0x0A
      brcs LcdHexN1
      subi rmp,-7
LcdHexN1:
      subi rmp,-'0'
      rcall LcdChar
      ret
;
; LcdDecDigit zieht die Dezimalzahl ab und schreibt eine Ziffer
;
LcdDecDigit:
      clr rmp ; zaehlt waehrend der Subtraktion
LcdDecDigit1:
      cp R2,ZL ; vergleiche LSB
      cpc R3,ZH ; dto., MSB
      brcs LcdDecDigit2 ; Subtraktion beendet
      sub R2,ZL ; subtrahiere LSB
      sbc R3,ZH ; dto., MSB
      inc rmp ; erhoehe Anzahl der Subtraktionen
      rjmp LcdDecDigit1 ; weiter
LcdDecDigit2:
      tst rmp ; null mal?
      brne LcdDecDigit3 ; nein
      tst R0 ; pruefe of fuehrende Nullen aktiv
      brne LcdDecDigit3
      ldi rmp,' ' ; fuehrende Null loeschen
      rcall LcdChar ; Leerzeichen ausgeben
      clr R0
      ret
LcdDecDigit3:
      subi rmp,-'0' ; addiere ASCII-Null
      rcall LcdChar ; stelle Ziffer im LCD dar
      clr R0 ; keine fuehrende Nullen mehr
      inc R0
      ret
;
; LcdDec schreibt die Zahl R3:R2 in Dezimal auf das Display
; 5, 4, 3 or 2 Ziffern werden dargestellt
;
LcdDec5:
      clr R0 ; fuehrende Nullen unterdruecken
      ldi ZH,HIGH(10000)
      ldi ZL,LOW(10000)
      rcall LcdDecDigit
LcdDec4:
      ldi ZH,HIGH(1000)
      ldi ZL,LOW(1000)
      rcall LcdDecDigit
LcdDec3:
      ldi ZH,HIGH(100)
      ldi ZL,LOW(100)
      rcall LcdDecDigit
```

```
LcdDec2:
      clr ZH
      ldi ZL,10
      rcall LcdDecDigit
      ldi rmp,'0'
      add rmp,R2
      rjmp LcdChar
;
; Ende der LCD-Routinen
;
; Lcd texte
;
; Init Text auf Display
;
LcdTxtInit1:
.DB " Akkulader V1.0  ",0
LcdTxtInit2:
.DB "(C)2005 by DG4FAC",0
LcdTxtInit3:
.DB "Vier Kanal Akku- ",0
LcdTxtInit4:
.DB " lader Prozessor ",0
;
; Ende der LCD Include-Datei
;
```

Include-Datei akkuuart.asm

```
; ************************************************************
; * Include-Datei fuer die UART-Kommandozeilen-Verarbeitung bei*
; * Akkuload (C)2005 by Gerh.Schmidt info@avr-asm-tutorial.net *
; ************************************************************
;
; Stellt die folgenden Routinen fuer die externe Verwendung bereit:
;
; Routine          Register Funktion
; ------------------------------------------------------------
; UartInit         rmp      Initiiere den UART-Empfaenger und
;                           Sender
; UartSendChar              Sendet das Zeichen in rmp ueber
;                           das UART
; UartSendOpenText rmp,R0,Z Sendet die Eroeffnungsmeldung beim
;                           Programmstart
; UartDec5         X,rmp    Sendet die Binaerzahl in R3:R2 in
; UartDec4                  Dezimalformat, R0>0: zeige
; UartDec3                  fuehrende Nullen an
; UartDec2
;
; UartSendN5       X,rmp    Sendet die Binaerzahl in R3:R2 in
; UartSendN4                Dezimalformat mit fester Laenge,
; UartSendN3                R0>0: zeigt fuehrende Nullen an
; UartSendN2
;
; UartHexB         rmp      Sendet das Hex Byte in rmp ueber das
;                           UART
; UartHex16        rmp,R0,Z Sendet 16 Bytes mit Adresse ab Z
;                           ueber das UART
; ------------------------------------------------------------
;
;
; Init UART
;
.EQU cUbrr = fClock/16/baud - 1 ; Baudratengenerator-Konstante
;
UartInit:
      ldi rmp,HIGH(cUbrr) ; Init UART Baudratengenerator
      out UBRRH,rmp
      ldi rmp,LOW(cUbrr)
      out UBRRL,rmp
      ldi rmp,(1<<RXEN)|(1<<TXEN) ; Enable RX und TX
      out UCSRB,rmp
      ldi rmp,(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0) ; 8N1
      out UCSRC,rmp
      in rmp,UDR; Loesche Rx-Zeilenpuffer
      in rmp,UDR
      in rmp,UDR
      ldi rmp,LOW(sUartRxBs) ; Setze Pufferzeiger an Anfang
      sts sUartRxBp,rmp
      clr rmp ; Init aktuellen Kanal
      sts sUartCurCh,rmp
      clr rFlg ; Loesche Flag
      ldi rmp,(1<<RXEN)|(1<<TXEN)|(1<<RXCIE) ; RX-Interrupts
      out UCSRB,rmp
      ret
;
;
```

```
; *************** Subroutinen ********************************
;
; Sende das Zeichen in rmp an das UART
;
UartSendChar:
      sbis UCSRA,UDRE ; warte auf leeres Senderegister
      rjmp UartSendChar
      out UDR,rmp ; sende das Zeichen in rmp
      ret
;
; Sende den Text im Flash (Z zeigt darauf) an das UART
;
UartSendZ:
      lpm ; lese naechstes Zeichen aus dem Flash
      adiw ZL,1
      tst R0 ; Null-Zeichen?
      brne UartSendR0 ; Nein, sende
      ret
UartSendR0:
      mov rmp,R0 ; kopiere Zeichen
      rcall UartSendChar ; sende Zeichen
      rjmp UartSendZ ; naechstes Zeichen
;
; Sende Eroeffnungstext
;
UartSendOpenText:
      ldi ZH,HIGH(2*UartTxtOpening) ; Zeiger Z auf Text
      ldi ZL,LOW(2*UartTxtOpening)
      rcall UartSendZ ; Sende den Text ab Z
      lds R2,sEANm ; lese Anzahl der gespeicherten Akku-Id's
      clr R3
      clr R0
      rcall UartSendN2 ; sende Anzahl Id's
      ldi rmp,cCr ; Ende der Zeile
      rcall UartSendChar
      ldi rmp,cLf
      rjmp UartSendChar
;
; Subtrahiere wiederholt und sende Zeichen wenn R0>0
;
UartSubt:
      clr rmp ; Ergebnisregister
UartSubt1:
      sub R2,XL ; subtrahiere LSB
      sbc R3,XH ; subtrahiere MSB
      brcs UartSubt2
      inc rmp
      rjmp UartSubt1
UartSubt2:
      add R2,XL ; addiere wieder
      adc R3,XH
      add R0,rmp
      breq UartSubt3
      subi rmp,-'0' ; addiere ASCII-Null
      rcall UartSendChar
UartSubt3:
      ret
;
; Sende eine Biaerzahl ueber das UART, Zahl ist in R3:R2
;
```

```
UartSendNmbr:
      clr R0
UartDec5:
      ldi XH,HIGH(10000)
      ldi XL,LOW(10000)
      rcall UartSubt
UartDec4:
      ldi XH,HIGH(1000)
      ldi XL,LOW(1000)
      rcall UartSubt
UartDec3:
      clr XH
      ldi XL,100
      rcall UartSubt
UartDec2:
      clr XH
      ldi XL,10
      rcall UartSubt
      ldi rmp,'0'
      add rmp,R2
      rjmp UartSendChar
;
; Sende eine feste Anzahl Dezimalziffern
;
UartSubN:
      clr rmp ; Ergebnisregister
UartSubN1:
      sub R2,XL ; subtrahiere LSB
      sbc R3,XH ; subtrahiere MSB
      brcs UartSubN2
      inc rmp
      rjmp UartSubN1
UartSubN2:
      add R2,XL ; addiere wieder
      adc R3,XH
      add R0,rmp
      subi rmp,-'0' ; addiere ASCII-Null
      tst R0
      brne UartSubN3
      ldi rmp,' '
UartSubN3:
      rjmp UartSendChar
;
; Sende eine Zahl in R3:R2 mit fester Laenge
;
UartSendN5:
      ldi XH,HIGH(10000)
      ldi XL,LOW(10000)
      rcall UartSubN
UartSendN4:
      ldi XH,HIGH(1000)
      ldi XL,LOW(1000)
      rcall UartSubN
UartSendN3:
      clr XH
      ldi XL,100
      rcall UartSubN
UartSendN2:
      clr XH
      ldi XL,10
```

```
      rcall UartSubN
      ldi rmp,'0'
      add rmp,R2
      rjmp UartSendChar
;
; Sendet ein Byte in rmp in Hex ueber das UART
;
UartHexB:
      push rmp ; sichere Byte
      swap rmp
      rcall UartHexN
      pop rmp ; stelle Bye wieder her
UartHexN:
      andi rmp,0x0F
      cpi rmp,10
      brcs UartHexN1
      subi rmp,-7
UartHexN1:
      subi rmp,-'0'
      rjmp UartSendChar
;
; UartHex16 sendet 16 Bytes in Hex ab Z
;
UartHex16:
      mov rmp,ZH
      rcall UartHexB
      mov rmp,ZL
      rcall UartHexB
      ldi rmp,16
      mov R0,rmp
UartHex16a:
      ldi rmp,' ';
      rcall UartSendChar
      ld rmp,Z+
      rcall UartHexB
      dec R0
      brne UartHex16a
      ldi rmp,cCr
      rcall UartSendChar
      ldi rmp,cLf
      rcall UartSendChar
      rjmp UartSendCursor
;
; UART Kommandozeile bearbeitet, stelle RX-Puffer wieder her, sende Cursor
;
UartRxRet:
      ldi rmp,LOW(sUartRxBs) ; Setze Pufferzeiger auf Anfang
      sts sUartRxBp,rmp
      cbr rFlg,(1<<bUartRxLine) ; loesche Zeilenflagge
UartSendCursor:
      ldi rmp,'A' ; sende aktuellen Kanal
      rcall UartSendChar
      lds rmp,sUartCurCh
      subi rmp,-'1' ; addiere ASCII-Eins
      rcall UartSendChar
      ldi rmp,'>' ; sende Cursor
      rjmp UartSendChar
;
; Ueberliest fuehrende Nullen, Gleichheitszeichen, Anfuehrungszeichen und Komma
; kehrt mit gesetztem Carry zurueck und Fehlermeldung in Z zurueck, wenn
```

```
; vorzeitig das Ende der Zeile erreicht wird
;
UartSkipLeading:
      lds R1,sUartRxBp ; lese Ende der Eingabezeile
UartSkipLeading1:
      cp XL,R1
      brcs UartSkipLeading2 ; nicht am Ende
      ldi ZH,HIGH(2*UartTxtUnexEol) ; Fehl rmeldung
      ldi ZL,LOW(2*UartTxtUnexEol)
      sec ; setze Carry
      ret
UartSkipLeading2:
      ld rmp,X+ ; lese Zeichen von der Zeile
      cpi rmp,'=' ; Gleichheitszeichen?
      breq UartSkipLeading1
      cpi rmp,',' ; Komma?
      breq UartSkipLeading1
      cpi rmp,' ' ; Leerzeichen?
      breq UartSkipLeading1
      cpi rmp,'"' ; Anfuehrungszeichen?
      breq UartSkipLeading1
      sbiw XL,1
      clc ; kein Ende der Zeile
      ret
;
; Hole naechste Ziffer einer Dezimalzahl, Carry gesetzt bei Zeilenende, Zero
; gesetzt bei Ende der Zahl
;
UartGetDigit:
      lds R1,sUartRxBp ; lese Ende der Eingabezeile
      cp XL,R1
      brcs UartGetDigit1 ; nicht am Ende
      ldi ZH,HIGH(2*UartTxtUnexEol) ; Fehlermeldung
      ldi ZL,LOW(2*UartTxtUnexEol)
      sec ; setze Carry
      ret
UartGetDigit1:
      ld rmp,X+ ; lese Ziffer
      cpi rmp,' ' ; Leerzeichen, Ende der Zahl?
      breq UartGetDigitEnd
      cpi rmp,cCr ; Ende der Zeile?
      breq UartGetDigitEnd
      cpi rmp,',' ; Komma?
      breq UartGetDigitEnd
      subi rmp,'0' ; mache Binarziffer
      brcs UartGetDigitIll
      cpi rmp,10 ; zu gross?
      brcc UartGetDigitIll
      clc ; alles ok
      clz
      ret
UartGetDigitIll:
      ldi ZH,HIGH(2*UartTxtNmbrErr) ; Ungueltiges Zeichen in der Zahl
      ldi ZL,LOW(2*UartTxtNmbrErr)
      sec ; setze Carry fuer Fehler
UartGetDigitEnd:
      ret
;
; Hole Dezimalzahl von der Empfangszeile
;
```

```
UartGetDec:
      rcall UartSkipLeading
      brcs UartGetDec3 ; Kehre mit gesetztem Carry zurueck, wenn Fehler
      clr R2 ; Resultat LSB
      clr R3 ; Resultat MSB
UartGetDec1:
      rcall UartGetDigit ; lese die naechst Ziffer
      brcs UartGetDec3 ; kehre mit gesetztem Carry zurueck, Fehler
      breq UartGetDec4 ; Ende der Zahl erreicht
      mov R0,R2 ; Kopiere Zwischenergebnis
      mov R1,R3
      add R2,R2 ; Ergebnis*2
      adc R3,R3
      brcs UartGetDec2 ; Zahlenueberlauf
      add R2,R2 ; Ergebnis*4
      adc R3,R3
      brcs UartGetDec2 ; Zahlenueberlauf
      add R2,R0 ; Ergebnis*5
      adc R3,R1
      brcs UartGetDec2 ; Zahlenueberlauf
      add R2,R2 ; Ergebnis*10
      adc R3,R3
      brcs UartGetDec2 ; Zalenueberlauf
      add R2,rmp ; addiere Ziffer
      ldi rmp,0
      adc R3,rmp
      brcc UartGetDec1 ; Kein Zahlenueberlauf, naechste Ziffer
UartGetDec2:
      ldi ZH,HIGH(2*UartTxtNmbrOvf) ; Zahlenueberlauf
      ldi ZL,LOW(2*UartTxtNmbrOvf)
      sec ; Carry setzen
UartGetDec3: ; Zahlenfehler
      ret
UartGetDec4: ; Zahl ist ok und komplett
      clc ; Carry loeschen
      ret
;
; ****************** Empfangszeile **********************
;
; Eine vollstaendige Emfangszeile empfangen, antworte
;
UartRxLine:
      ldi rmp,LOW(UartRxRet) ; Rueckkehradresse UartRxRet auf Stapel
      push rmp
      ldi rmp,HIGH(UartRxRet)
      push rmp
      ldi ZH,HIGH(2*UartCmds) ; Z zeigt auf Befehlsliste
      ldi ZL,LOW(2*UartCmds)
UartRxLine1:
      lpm ; Ende der Befehlsliste?
      mov rmp,R0 ; FF signalisiert Ende der Liste
      cpi rmp,0xFF
      brne UartRxLine3
UartRxLine2: ; Ende der Befehlsliste, unbekannter Befehl
      ldi ZH,HIGH(2*UartTxtUnkCmd) ; Sende Fehlermeldung
      ldi ZL,LOW(2*UartTxtUnkCmd)
      rjmp UartSendZ
UartRxLine3: ; beginne Vergleich
      ldi XH,HIGH(sUartRxBs) ; X auf emfangene Zeile
      ldi XL,LOW(sUartRxBs)
```

```
UartRxLine4:
      lds rmp,sUartRxBp ; Pufferzeiger lesen
      cp XL,rmp ; Ende des Puffers erreicht?
      brcs UartRxLine7 ; nein, weiter
UartRxLine5: ; ueberspringe diesen Befehl, gehe zum naechsten
      lpm ; lese naechstes Befehlszeichen
      adiw ZL,1 ; Z auf naechstes Zeichen
      mov rmp,R0 ; Endezeichen fuer Befehl?
      cpi rmp,'#' ; Endzeichen erreicht?
      brne UartRxLine5 ; nein
      lpm ; Ist das naechste Zeichen ein Leerzeichen?
      mov rmp,R0
      cpi rmp,' '
      brne UartRxLine6 ; nein
      adiw ZL,1
UartRxLine6:
      adiw ZL,7 ; springe ueber die Adressen
      rjmp UartRxLine1 ; naechster Befehl
UartRxLine7:
      lpm ; lese Zeichen aus Flash
      mov rmp,R0 ; kopiere
      cpi rmp,'#' ; Ende des Befehls?
      breq UartRxLine8 ; ja
      ld rmp,X+ ; lese naechstes Zeichen aus Puffer
      sbr rmp,0x20 ; in Kleinbuchstabe
      cp rmp,R0 ; Vergleich des Zeichens
      brne UartRxLine5 ; nicht gleich, springe zum naechsten Befehl
      adiw ZL,1 ; naechstes Zeichen
      rjmp UartRxLine4 ; naechstes Zeichen im Puffer pruefen
UartRxLine8:
      adiw ZL,1 ; Leerzeichen hinter #?
      lpm
      mov rmp,R0
      cpi rmp,' '
      brne UartRxLine9 ; nein
      adiw ZL,1 ; ja
UartRxLine9:
      lds rmp,sUartRxBp ; Ende des Puffers erreicht?
      cpc XL,rmp
      brcc UartRxLine2
      ld rmp,X+ ; lese naechstes Zeichen aus Puffer
      cpi rmp,cCr ; Ende der Zeile?
      brne UartRxLine10 ; mit Parameter
      ; Befehl ohne Parameter
      adiw ZL,5 ; Zeige Z auf Addresse
      lpm ; lese LSB
      push R0 ; auf Stapel
      adiw ZL,1 ; lese MSB
      lpm
      push R0 ; auf Stapel
      clc ; keine Parameter
      ret ; springe zum Befehl
UartRxLine10: ; Befehlszeile mit Parameter
      cpi rmp,'=' ; Parameter?
      brne UartRxLine5 ; Nein, suche naechsten Befehl
      lpm ; lese Anzahl Parameter
      tst R0
      brne UartRxLine11 ; Befehl hat Parameter
      ldi ZH,HIGH(2*UartTxtParaErr) ; Fehlermeldung
      ldi ZL,LOW(2*UartTxtParaErr)
```

```
      rjmp UartSendZ
UartRxLine11: ; Kommadozeilenparameter
      rcall UartGetDec ; hole Dezimalzahl
      brcc UartRxLine17 ; Zahl ist ok
      rjmp UartSendZ ; Fehlermeldung
UartRxLine17:
      adiw ZL,1 ; lese LSB minimum Parameter
      lpm
      adiw ZL,1
      cp R2,R0 ; vergleiche LSB
      lpm ; lese MSB minimum Parameter
      cpc R3,R0 ; vergleiche MB
      brcs UartRxLine18
      adiw ZL,1 ; lese LSB maximum Parameter
      lpm
      adiw ZL,1
      sec ; Setze Carry fuer Max+1
      cpc R2,R0 ; vergleiche LSB
      lpm ; lese MSB max Parameter
      cpc R3,R0 ; vergleiche MSB
      brcs UartRxLine19 ; Zahl ist ok
UartRxLine18: ; Zahl au rhalb des zulaessigen Bereichs
      ldi ZH,HIGH(2*UartTxtNmbrOor)
      ldi ZL,LOW(2*UartTxtNmbrOor)
      rjmp UartSendZ
UartRxLine19: ; lese Adresse und springe
      adiw ZL,1 ; lese LSB
      lpm
      push R0 ; LSB auf Stapel
      adiw ZL,1 ; lese MSB
      lpm
      push R0 ; MSB auf Stapel
      sec ; setze Carry, Parameter
      ret ; springe an Befehlsadresse
;
; ***************** Befehle *********************************
;
; UART-Befehl HILFE
;
UartHelp:
      ldi ZH,HIGH(2*UartTxtHelp) ; Hilfetext
      ldi ZL,LOW(2*UartTxtHelp)
      rjmp UartSendZ
;
; UART-Befehl RESTART
;
UartRestart:
      pop rmp ; Loesche Ruecksprungadresse
      pop rmp
      jmp 0 ; starte neu
;
; UART-Befehl Kanal
;
UartChannel:
      brcc UartChannel1
      dec R2
      sts sUartCurCh,R2
UartChannel1: ; gib aktuellen Kanal aus
      ldi ZH,HIGH(2*UartTxtCurrCh)
      ldi ZL,LOW(2*UartTxtCurrCh)
```

```
      rcall UartSendZ
      lds rmp,sUartCurCh ; lese aktuellen Kanal
      subi rmp,-'1' ; addiere ASCII-Eins
      rcall UartSendChar
      adiw ZL,1 ; vervollstaendige Ausgabe
      rjmp UartSendZ
;
; Holt die aktuelle Akku-Id-Adresse des Kanals
;
UartGetAkkuId:
      ldi ZH,HIGH(sAdcAI1) ; Z auf Akku-Id 1
      ldi ZL,LOW(sAdcAI1)
      lds rmp,sUartCurCh
      add ZL,rmp
      add ZL,rmp
      ret
;
; Setze die Akku-Id des aktuellen Kanals
;
UartAkkuId:
      brcc UartAkkuId1
      rcall UartGetAkkuId
      st Z,R2
      lds R0,sUartCurCh
      rcall IdSetPar
UartAkkuId1:
      ldi ZH,HIGH(2*UartTxtAkkuId)
      ldi ZL,LOW(2*UartTxtAkkuId)
      rcall UartSendZ
      rcall UartGetAkkuId
      ld R2,Z
      clr R3
      clr R0
      rcall UartDec2
      ldi rmp,cCr
      rcall UartSendChar
      ldi rmp,cLf
      rcall UartSendChar
      rjmp UartOutSettings
;
; Zeiger rmp auf das Entladebit
;
UartUnloadBit:
      lds R0,sUartCurCh
      ldi rmp,0x10
UartUnloadBit1:
      tst R0
      breq UartUnloadBit2
      lsl rmp
      dec R0
      rjmp UartUnloadBit1
UartUnloadBit2:
;
; Zeige mit X auf das PWM-Register
;
UartChannX:
      clr XH ; Zeiger MSB is Null
      ldi XL,cPwm1S ; LSB auf LSB der PWM von Kanal Eins
      lds R0,sUartCurCh ; addiere Kanal
      add XL,R0
```

```
	ret
;
; Aktuelle UART Parameteradresse in X
;
UartGetChPar:
	lds R1,sUartCurCh ; lese Kanal
	lsl R1 ; *2
	lsl R1 ; *4
	lsl R1 ; *8
	ldi XH,HIGH(sCh1UV) ; Basisaddresse Entladespannung
	ldi XL,LOW(sCh1UV)
	add XL,R1 ; addieren
	brcc UartGetChPar1
	inc XH
UartGetChPar1:
	ret
;
; Hole Adresse des Kanalstatus-Textes in Z
;
UartGetChAct:
	ldi ZH,HIGH(2*UartTxtAct)
	ldi ZL,LOW(2*UartTxtAct)
	lds R1,sUartCurCh ; aktueller Kanal
	lds rmp,sState ; Kanalstatus
UartGetChAct1:
	tst R1
	breq UartGetChAct2
	lsr rmp
	lsr rmp
	dec R1
	rjmp UartGetChAct1
UartGetChAct2:
	andi rmp,0x03 ; isoliere Bits
	breq UartGetChAct4
UartGetChAct3:
	lpm
	adiw ZL,1
	tst R0
	brne UartGetChAct3
	dec rmp
	rjmp UartGetChAct2
UartGetChAct4:
	ret
;
; Setze die Entladespannung
;
UartSetU:
	brcc UartSetU1
	rcall UartGetChPar ; Adresse in X
	st X+,R2 ; speichere Parameter dort
	st X,R3
UartSetU1:
	rjmp UartOutSettings
;
; Setze den Ladestrom dieses Kanals
;
UartSetI:
	brcc UartSetI1
	rcall UartGetChPar ; Adresse in X
	adiw XL,2 ; Ladestrom ist der zweite Parameter
```

```
      st X+,R2 ; speichere die empfangene Zahl dort
      st X,R3
UartSetI1:
      rjmp UartOutSettings
;
; Setze die Kapazitaet
;
UartSetW:
      brcc UartSetW1 ; kein Parameter angegeben
      rcall UartGetChPar ; hole Adresse in X
      adiw XL,4 ; Kapazitaet ist der dritte Parameter
      st X+,R2 ; speichern der empfangenen Zahl
      st X,R3
UartSetW1:
      rjmp UartOutSettings
;
; Setze den Erhaltungsstrom
;
UartSetM:
      brcc UartSetM1
      rcall UartGetChPar ; Adresse in X
      adiw XL,6 ; vierte Variable
      st X+,R2 ; speichern des Parameters
      st X,R3
UartSetM1:
      rjmp UartOutSettings
;
; Setze den Status auf den Wert in rmp
; Anfang: rmp Bit 1 und 0 = Aktivitaetsflags
;
UartSetState:
      lds R0,sUartCurCh ; Kanal nach R0
      lds R2,sState ; statusbits nach R2
      mov R3,rmp ; OR-Maske in R3 speichern
      ldi rmp,0xFC ; Maske fuer AND erstellen
      mov R1,rmp
      mov rmp,R3 ; Original wieder herstellen
UartSetState1:
      tst R0 ; Kanal erreicht?
      breq UartSetState2
      sec ; naechste Kanal
      rol R1
      sec
      rol R1
      lsl rmp ; Soll-Statusbits zwei links schieben
      lsl rmp
      lsr R2 ; Ist-Status-Bits zwei rechts
      lsr R2
      dec R0
      rjmp UartSetState1
UartSetState2:
      lds R0,sState
      and R0,R1 ; beide Bits loeschen
      or R0,rmp ; neue Bts setzen
      sts sState,R0
      mov rmp,R2 ; vorherigen Status pruefen
      andi rmp,0x03 ; isoliere Statusbits
      cp rmp,R3 ; Status veraendert?
      breq UartSetState4 ; nein, ueberspringe Korrektur
      lds R0,sUartCurCh ; R0 ist aktueller Kanal
```

```
      cpi rmp,0x02 ; vorher im Ladezustand?
      brne UartSetState3 ; inaktiv, Entladen oder Erhaltungsladen
      call EpStore ; geladene Kapazitaet im EEPROM speichern
      sec
UartSetState3: ; loesche akkumulierte Kapazitaet, wenn inaktiv oder Laden
      call ClearCap ; Kapazitaet in Kanal R0 loeschen
UartSetState4:
      rjmp UartOutSettings
;
; Setze Statusbits auf AUS
;
UartClear:
      ldi rmp,0x00
      rjmp UartSetState
;
; Setze das Entladebit und loesche die PWM
;
UartUnload:
      rcall UartUnloadBit
      or rPwmOut,rmp
      clr rmp
      st X,rmp
      ldi ZH,HIGH(2*UartTxtUnload)
      ldi ZL,LOW(2*UartTxtUnload)
      rcall UartSendZ
      ldi rmp,0x01 ; Entladeflagge
      rjmp UartSetState
;
; Statusbits auf Erhaltungsladung
;
UartMaint:
      ldi rmp,0x03
      rjmp UartSetState
;
; Starte Ladevorgang
;
UartLoad:
      ldi rmp,0x02
      rjmp UartSetState
;
; Gib alle Einstellungen im Kanal auf dem UART aus
;
UartOutSettings:
      ldi rmp,4 ; R4 ist Zaehler
      mov R4,rmp
      ldi ZH,HIGH(2*UartTxtSettings)
      ldi ZL,LOW(2*UartTxtSettings)
      rcall UartSendZ
      lds R1,sUartCurCh
      ldi rmp,'1'
      add rmp,R1
      rcall UartSendChar
      rcall UartSendZ
      ldi XH,HIGH(sAdcAI1) ; Schreibe Id
      ldi XL,LOW(sAdcAI1)
      lds rmp,sUartCurCh
      add XL,rmp
      add XL,rmp
      ld R2,X
      clr R3
```

```
      rcall UartSendNmbr
      rcall UartSendZ
      push ZH
      push ZL
      rcall UartGetChAct ; schreibe Statustext
      rcall UartSendZ
      pop ZL
      pop ZH
      rcall UartSendZ
      rcall UartGetChPar ; schreibe alle Parameter
UartOutSettings1:
      ld R2,X+
      ld R3,X+
      push XH
      push XL
      rcall UartSendNmbr
      rcall UartSendZ
      pop XL
      pop XH
      dec R4
      brne UartOutSettings1
      ret
;
; Texte fuer die Ausgabe der Einstellungen
;
UartTxtSettings:
.DB "Ch ",0
.DB ": Id=",0
.DB ", Status=",0
.DB ", U= ",0
.DB "mV, I= ",0
.DB "mA, C= ",0
.DB "mAh, Im= ",0
.DB "mA",$0D,$0A,$00,$00
;
; Aktivitaetsstatus des Kanals
;
UartTxtAct:
.DB "abgeschaltet ",0
.DB "Entladen ",0
.DB "Laden",0
.DB "Erhaltung",0
;
; Schalte Monitoring um
;
UartMoniOn:
      ldi rmp,1<<bUMoni ; Moniflag-Maske
      eor rFlgD,rmp ; umschalten
      ldi ZH,HIGH(2*UartTxtMoniOff) ; Monitor aus ausgeben
      ldi ZL,LOW(2*UartTxtMoniOff)
      and rmp,rFlgD
      breq UartMoniOn1
      ldi ZH,HIGH(2*UartTxtMoniOn) ; Monitor an ausgeben
      ldi ZL,LOW(2*UartTxtMoniOn)
      rcall UartSendZ
      rjmp DispTimeUd ; Systemzeit ueber UART ausgeben
UartMoniOn1: ; Monitoring umgeschaltet
      rjmp UartSendZ
;
; Setze die PWM eines Kanals
```

```
;
UartPwm:
      brcc UartPwm1
      rcall UartUnloadBit
      st X,R2 ; setze aktuellen Kanalwert
      com rmp
      and rPwmOut,rmp ; PWM-Ausgabe setzen
UartPwm1:
      ldi ZH,HIGH(2*UartTxtPwm) ; Text fuer PEWM-Ausgabe
      ldi ZL,LOW(2*UartTxtPwm)
      rcall UartSendZ
      rcall UartUnloadBit ; Entladebit lesen
      ld R2,X
      clr R3
      rcall UartSendNmbr
      adiw ZL,1
      rjmp UartSendZ
;
; Neue Akku-Id erzeugen
;
UartNew:
      brcs UartNew2 ; erster parameter ok
      ldi ZH,HIGH(2*UartTxtNewParamErr) ; Parameterfehler
      ldi ZL,LOW(2*UartTxtNewParamErr)
UartNew1: ; Fehlermeldung in Z
      rjmp UartSendZ
UartNew2:
      lds rmp,sEANm ; Anzahl an Id's lesen
      cpi rmp,cEepMax+1 ; mt Maximalzahl vergleichen
      brcc UartNew8 ; zu viele Id's
      ldi ZH,HIGH(sEAId) ; Z auf Puffer setzen
      ldi ZL,LOW(sEAId)
      lds rmp,sEANm ; Zahl der Id's lesen
      inc rmp ; naechste Akku-Id
      st Z,rmp ; in Puffer
      adiw ZL,2 ; zeige auf Kapazitaet
      st Z+,R2 ; Kapazitaet in R3:R2 in Puffer
      st Z,R3
      sbiw ZL,2 ; auf Akkutyp zeigen
      rcall UartGetDec ; Groesze aus Zeile lesen
      brcs UartNew1 ; Fehler
      tst R3 ; Groesze pruefen
      brne UartNew3 ; MSB>0, falsch
      mov rmp,R2
      cpi rmp,4 ; Groesze kleiner 4?
      brcs UartNew4 ; ja, ok
UartNew3: ; falsche Groesze
      ldi ZH,HIGH(2*UartTxtNewSize) ; Fehlermeldung
      ldi ZL,LOW(2*UartTxtNewSize)
      rjmp UartSendZ
UartNew4:
      subi rmp,-0xA0 ; A0 zu Groesze addieren
      st Z,rmp ; Groesze speichern
      adiw ZL,3 ; auf Anzahl Ladevorgaenge zeigen
      rcall UartGetDec ; Anzahl Ladevorgaenge aus Zeile
      brcs UartNew1 ; Fehler
      st Z+,R2 ; in Puffer speichern
      st Z+,R3
      clr rmp
      st Z+,rmp ; Restkapazitaet loeschen
```

```
      st Z+,rmp
      rcall UartSkipLeading ; ueber Trennzeichen hinweglesen
      brcs UartNew1 ; Fehler
      ldi rmp,8 ; 8 Byte Text lesen
      mov R0,rmp
UartNew5:
      ld rmp,X+ ; lese Textzeichen
      cpi rmp,'"' ; Ende des Texts?
      breq UartNew6 ; ja
      cpi rmp,cCr ; Ende der Zeile?
      breq UartNew6 ; ja
      st Z+,rmp ; Zeichen in Puffer speichern
      dec R0 ; Zeichenzaehler
      brne UartNew5 ; naechstes Zeichen
      rjmp UartNew7 ; Rest ueberlesen
UartNew6:
      tst R0 ; Anzahl Zeichen auf 8 auffuellen
      breq UartNew7
      ldi rmp,' '
      st Z+,rmp
      dec R0
      rjmp UartNew6
UartNew7:
      call EpWrite ; Puffer ins EEPROM schreiben
      lds rmp,sEANm ; Anzahl Id's erhoehen
      inc rmp
      sts sEANm,rmp ; und speichern
      ldi ZH,HIGH(2*UartTxtList) ; Header
      ldi ZL,LOW(2*UartTxtList)
      rcall UartSendZ
      lds rmp,sEAId ; Anzahl Id's ausgeben
      rjmp UartList1
UartNew8: ; Fehler zu viele Id's
      ldi ZH,HIGH(2*UartTxtNewMany)
      ldi ZL,LOW(2*UartTxtNewMany)
      rjmp UartSendZ
;
; Auflisten der gespecherten Akku-Id's
;
UartList:
      ldi ZH,HIGH(2*UartTxtList) ; Header
      ldi ZL,LOW(2*UartTxtList)
      rcall UartSendZ
      clr R6 ; R6 ist Zaehler
UartList1:
      inc R6
      lds R0,sEANm ; lese Anzahl Id's
      sec
      cpc R6,R0 ; Ende der Liste?
      brcc UartList3
      mov rmp,R6
      call EpRead ; Id in rmp aus EEPROM lesen
      ldi ZH,HIGH(sEAId) ; Z auf Puffer
      ldi ZL,LOW(sEAId)
      ld R2,Z+ ; kopiere Id-Nummer
      clr R3
      clr R0
      rcall UartSendN2
      ldi rmp,' '
      rcall UartSendChar
```

```
      ld rmp,Z+ ; Schreibe Akkugroesze
      rcall UartHexB
      ldi rmp,' '
      rcall UartSendChar
      ld R2,Z+ ; Schreibe Nennkapazitaet
      ld R3,Z+
      clr R0
      rcall UartSendN5
      ldi rmp,' '
      rcall UartSendChar
      ld R2,Z+ ; Schreibe Anzahl Volladungen
      ld R3,Z+
      clr R0
      rcall UartSendN5
      ldi rmp,' '
      rcall UartSendChar
      ld R2,Z+ ; Schreibe Restkapazitaet
      ld R3,Z+
      clr R0
      rcall UartSendN5
      ldi rmp,' '
      rcall UartSendChar
      ldi rmp,8
      mov R1,rmp
UartList2:
      ld rmp,Z+ ; Lese Text
      rcall UartSendChar
      dec R1
      brne UartList2
      ldi rmp,cCr
      rcall UartSendChar
      ldi rmp,cLf
      rcall UartSendChar
      rjmp UartList1 ; und weiter
UartList3:
      ret
;
; ***************** Tabellen ************************
;
; Befehle, Parameter und Adresstabelle fuer UART-Befehle
;
UartCmds:
.DB "hilfe# ",0 ; HILFE.-Befehl, keine Parameter
.DW 0,0,UartHelp
.DB "a# ",1 ; Akku Kanalwahl
.DW 1,4,UartChannel
.DB "n# ",1 ; Akku-Id
.DW 0,32,UartAkkuId
.DB "u# ",1 ; Entlade-Abschaltspannung
.DW 700,1200,UartSetU
.DB "i# ",1 ; Ladestrom  in mA
.DW 5,350,UartSetI
.DB "c# ",1 ; Kapazitaet in mAh
.DW 10,2500,UartSetW
.DB "m# ",1 ; Erhaltungsstrom in mA
.DW 5,100,UartSetM
.DB "aus# ",0 ; Kanal abschalten
.DW 0,0,UartClear
.DB "entladen#",0 ; Entladen
.DW 0,0,UartUnload
```

```
.DB "laden# ",0 ; Laden
.DW 0,0,UartLoad
.DB "erhalten#",0 ; Erhalten
.DW 0,0,UartMaint
.DB "monitor# ",0 ; Monitor ein/aus
.DW 0,0,UartMoniOn
.DB "pwm# ",1 ; PWM-Wert setzen
.DW 0,255,UartPwm
.DB "neu# ",1 ; neue Akku-Id erzeugen
.DW 100,9000,UartNew
.DB "liste# ",0 ; Liste der gespeicherten Akkus
.DW 0,0,UartList
.DB "neustart#",0 ; RESTART-Befehl, keine Parameter
.DW 0,0,UartRestart
.DW $FFFF ; Ende der Befehlsliste
;
; UART Texte
;
UartTxtHelp:
.DB "Akkuload Befehlsliste:",$0D,$0A
.DB "<hilfe>, kein param: diese Liste. ",$0D,$0A
.DB "<a> oder <a=x>, param x (1..4): lese/setze Kanalnummer. ",$0D,$0A
.DB "<n> oder <n=x>, param x (1..32): lese/setze Akku-Id.",$0D,$0A
.DB "<u> oder <u=x>, param x (700..1200): lese/setze Entladespannung.",$0D,$0A
.DB "<i> oder <i=x>, param x (5..175): lese/setze Ladestrom in mA. ",$0D,$0A
.DB "<c> oder <c=x>, param x (10..9999): lese/setze Kapazitaet in mAh. ",$0D,$0A
.DB "<m> oder <m=x>, param x (5..100): lese/setze Erhaltungsstrom in mA",$0D,$0A
.DB "<aus>, kein param: schalte Kanal ab.",$0D,$0A
.DB "<entlade>, kein param: schalte Entladen ein.",$0D,$0A
.DB "<lade>, kein param: starte laden. ",$0D,$0A
.DB "<erhalte>, kein param: schalte Erhaltungsladung ein.",$0D,$0A
.DB "<monitor>, kein param: schalte monitoring ein/aus.",$0D,$0A
.DB "<pwm> oder <pwm=x>, param x (0..255): setze PWM-Wert. ",$0D,$0A
.DB "<neu=Kapazitaet, Groesze, Ladungen, Text> erzeugt neue Id,",$0D,$0A
.DB "   Kapazitaet 100..9000, Groesze 0..3, Ladungen 0..65535, Text”
.DB “ frei.",$0D,$0A
.DB "<liste>, kein param: Liste aller gespeicherten Akku-Id's. ",$0D,$0A
.DB "<neustart>, kein param: Neustart des Controllers.",$0D,$0A,$00
;
UartTxtOpening:
.DB $0D,$0A,$0D,$0A,"Akkulader (C)2005 by info@avr-asm-tutorial.net",$0D,$0A
.DB "-----------------------------------------------",$0D,$0A
.DB "Gespeicherte Akku-Id's : ",$00
.DB "Gib 'hilfe' <RETURN> fuer Menue ein.",$0D,$0A
;
UartTxtUnkCmd:
.DB "Unbekannter Befehl!",$0D,$0A,$00
;
UartTxtParaErr:
.DB "Befehl hat keine Parameter!",$0D,$0A,$00
;
UartTxtUnexEol:
.DB "Unerwartetes Ende der Zeile! ",$0D,$0A,$00
;
UartTxtNmbrErr:
.DB "Unzulaessiges Zeichen in Zahl! ",$0D,$0A,$00
;
UartTxtNmbrOvf:
.DB "Zahlenueberlauf! ",$0D,$0A,$00
;
```

```
UartTxtNmbrOor:
.DB "Zahl ausserhalb des zulaessigen Bereichs!",$0D,$0A,$00
;
UartTxtCurrCh:
.DB "Derzeitiger Akkukanal ist : ",$00,$20
.DB " ",$0D,$0A,$00
;
UartTxtPwm:
.DB "Derzeitiger PWM-Wert ist : ",$00
.DB " ",$0D,$0A,$00
;
UartTxtUnload:
.DB "Entladen eingeschaltet.",$0D,$0A,$00
;
UartTxtMoniOff:
.DB "Monitoring ist aus.",$0D,$0A,$00 ; Fehler in älteren Versionen!
;
UartTxtMoniOn:
.DB "Monitoring ist an. ",$0D,$0A,$00 ; Fehler in älteren Versionen!
;
UartTxtList:
.DB "Id Gr Kapaz Ladg. _Rest Text____",$0D,$0A,$00,$00
;
UartTxtAkkuId:
.DB "Aktuelle Id ist ",$00,$00
;
UartTxtNewParamErr:
.DB "Vermisse alle Parameter! ",$0D,$0A,$00
;
UartTxtNewSize:
.DB "Ungueltige Groesze!",$0D,$0A,$00
;
UartTxtNewMany:
.DB "Zu viele Akku-Id's! ",$0D,$0A,$00,$00
;
;
; Ende der UART-Include-Datei
;
```