Danny Hiebel
FTEET4

Werner-Siemens-Schule Stuttgart
Fachschule für Technik

Technikerarbeit
2004/2005
Dokumentation

µController geregeltes Ladegerät

Werner-Siemens-Schule Stuttgart Fachschule für Technik	**Technikerarbeit 2004/2005** **Dokumentation**	

Vorwort

Dieses Projekt entstand aus der Erfordernis Modellbau-Akkus für den Anwender bedienerfreundlich und darüber hinaus technisch zuverlässig, sowie unter Berücksichtigung der Akkulebensdauer diese schonend zu laden. Derzeit sind am freien Markt zwar unterschiedlichste Ladegeräte erhältlich, jedoch sind diese i.d.R. nicht in der Lage die aktuelle Vielfalt an unterschiedlichen Akkutypen zu berücksichtigen, wobei diejenigen, die unterschiedliche Akkutypen laden können, üblicherweise wegen zu geringer Ladespannung die Aufgabe nicht optimal lösen. Oft besteht bei diesen Geräten die Gefahr einer Fehlbedienung durch den Anwender, was dann zur schnellen Zerstörung der Akkus führen kann. Im Modellbau, speziell dem Flugmodellbau, werden in letzter Zeit immer häufiger sog. Akkupacks eingesetzt, die aus bis zu 20 Zellen und mehr bestehen können. Trotz dieser ungewöhnlich großen Zahl an gekoppelten Einzelzellen sind Flüge von mehr als 10 Minuten kaum durchführbar. Hier bietet die Industrie nunmehr die neueren LiPo-Akkus als Lösung an, die eine relativ große Kapazität bieten und damit längere Flugzeiten ermöglichen können. Dennoch werden diese heute wegen der hohen Anschaffungskosten kaum verwendet. Aus den genannten Gründen soll im Rahmen dieser Projektarbeit die Möglichkeit geschaffen werden beliebige Akkus vor Ort - also auf dem Flugplatz – über die Kfz-Batterie, möglichst schnell und schonend zu laden.
Hierbei soll keineswegs versucht werden ein Konkurrenzprodukt zu entwickeln. Es geht viel mehr darum ein Gerät zu schaffen, dass die o.g. speziellen Bedürfnissen der Anwender berücksichtigt; d.h. die technische Funktionalität wird priorisierend und unabhängig wirtschaftlicher Gesichtspunkte (z.B. Serienfertigung) betrachtet. Ein interessanter Aspekt dieser Lösung besteht darin, dass durch Anpassen bzw. Erweitern der Betriebssoftware des Ladegeräts neue oder bisher unbekannte Akkutypen mit deren technischen Spezifikationen einfach mit berücksichtigt werden können oder sogar, falls dies erforderlich sein sollte, neue Ladeverfahren einfach realisierbar sind. Insofern ist das Gerät aussergewöhnlich anpassungsfähig, was den sinnvollen Einsatz vermutlich über dessen gesamte Lebensdauer sichert.

Projektbeschreibung

Definition der Aufgaben

Im Rahmen dieses Projekts ist ein bedienerfreundliches Schnellladegerät für Modellbauakkus zu entwickeln, welches in der Standartausstattung NiCd-, NiMH-, LiIon-, Blei- und LiPo[1]-Akkus mit jeweils erforderlichem (frei wählbaren) Ladestrom laden soll. Der vom Gerät maximal zur Verfügung zu stellende Ladestrom ist mit 6A vorgegeben. Dieser max. Ladestrom wurde deswegen gewählt, um die neuen LiPo-Akkus, die eine größere Kapazität haben als herkömmliche Akkus, in einer ebenfalls akzeptablen Zeit laden zu können. Größere Ströme wären durchaus durch Austauschen der MOSFETs und Anpassen der Software ohne weiteres möglich, machen jedoch auf Basis der aktuellen Aufgabenstellung keinen Sinn.
Zusätzlich ist ein Betriebszustand des Ladegeräts erforderlich, der das gezielte, schonende Entladen des angeschlossenen Akkus ermöglicht. Der Entladestrom soll hierbei 6A nicht überschreiten. Weiterhin ist die Option wünschenswert, die Kapazität des angeschlossenen Akkus bestimmen und ablesen zu können, sowie alte, evtl. durch den Memory-Effekt geschädigte Zellen durch mehrere Lade-/ Entlade-Zyklen wieder aufzufrischen. Als eine der Aufgabe angemessene Benutzerschnittstelle wurde ein Touchdisplay gewählt, welches sowohl die komfortable Eingabe als auch die übersichtliche Ausgabe der erforderlichen Informationen und Parameter bietet.
Die vom Anwender jeweils zu bestimmenden Parameter wie Ladestrom, Entladestrom, Zellentyp, etc. werden dabei mittels des Touch-Displays visualisiert. Diese Benutzerschnittstelle ermöglicht deswegen eine einfache und übersichtliche Handhabung des Ladegeräts. Um die Akkutypabhängigen unterschiedlichen Ladecharakteristiken technisch realisieren zu können, bietet sich zur zentralen Steuerung aller erforderlicher Funktionen und Betriebszustände die Verwendung eines geeigneten Mikrocontrollers an.

Anforderungen an das Ladegerät

Da Modellflugakkus aus bis zu 30 Zellen bestehen können, ergibt sich bei NiCd-Zellen eine Ladespannung von ca. 48 V, wenn man eine Ladeschlussspannung von 1,6 V pro Zelle dieser Berechnung zu Grunde legt. Die höchst mögliche Betriebsspannung der MOSFETs liegt bei 55 V. Da die max. Ladespannung 7V unterhalb der zulässigen Spannung der MOSFETs liegt, wurde dem Aspekt einer möglichen Überspannung für dieses Ladegerät keine weitere Bedeutung bemessen.
Als unteres Limit für die Versorgungsspannung wurde 10,8 V festgelegt, damit der sichere Betrieb des Geräts an einer Kfz-Batterie gewährleistet ist. Die Obergrenze ist theoretisch offen. Unabhängig davon wurde unter Berücksichtigung des mobilen Einsatzes eine maximale Versorgungsspannung von 13,8V festgelegt. Dieses Spannungsfenster soll vom Mikrocontroller überwacht werden und bei der Unterschreitung ein Lade- bzw. Entladevorgang verhindert werden.

[1] Lithium-Polymer

Um Schäden am Ladegerät oder Akku durch Controllerausfall zu verhindern, überwacht dieser sich selbst mittels Watchdog-Funktionalität. Dieser Watchdog prüft zyklisch ob der Microcontroller noch arbeitet und führt gegebenenfalls einen Reset des Controllers aus.

Anforderungen an den Mikrocontroller

Der Mikrocontroller bildet das zentrale Steuerelement bzw. Herzstück der gesamten Schaltung. Deshalb müssen dessen Auswahl basierend auf dessen Anforderungen besondere Aufmerksamkeit gewidmet werden.
Der aus dem Unterricht bekannte Z80 von ZiLOG ist für die genannte Aufgabenstellung gänzlich ungeeignet. Dieser bietet lediglich digitale I/Os, ist sehr langsam in der Befehlsabarbeitung und bedarf eines relativ großen Schaltungsaufwandes. Aus diesen Gründen war ein aktueller und der Aufgabe angemessener Mikrocontroller zu wählen. Da ich zu Projektbeginn keine Erfahrung in dieser Richtung hatte, ging ich auf Herrn Bullinger zu. Er riet mir die Controllern der Firma Atmel näher zu untersuchen. Nach einigem Suchen kam ich auf die AT90-Reihe und die Megas von Atmel. Diese verfügen alle über einen internen Flash und einen internen EEPROM. Viele dieser Mikrocontroller verfügen darüber hinaus bereits über einen integrierten Quarz, was den externen Schaltungsaufwand zusätzlich reduziert. Ein weiterer Vorteil von diesen Prozessoren ist ihre Geschwindigkeit. Die meisten Befehle werden innerhalb eins Quarztaktes ausgeführt. Das entspricht 16 Millionen Befehlen pro Sekunde bei einem Systemtakt von 16 MHz. Des weiteren gibt es für diese Microcontroller zahlreiche kostenlose Compiler, Programmieradapter und Foren in denen man schnell und kompetente Hilfe finden kann.
Aus Kostengründen wollte ich auf einen dieser Programmieradapter verzichten. Auch dies ist bei den Controllern von Atmel kein Problem, da diese über eine ISP[2] - Programmierschnittstelle verfügen. Das bietet den Vorteil, den Controller vor der Programmierung in das Zielsystem einbauen zu können und anschließend mittels der seriellen- oder parallelen Schnittstelle des PCs diesen nachträglich zu programmieren. Des weiteren sollte der zu wählende Mikrocontroller über eine serielle Schnittstelle, zum Ansteuern des Displays verfügen, analoge Eingänge um die Messwerte zu erfassen und mindestens zwei PWM[3]-Ausgänge für die Ansteuerung der Ladetransistoren besitzen.

Auswahlkriterien zusammengefasst:

- Integrierter A/D-Wandler
- PWM-Funktion
- serielle Schnittstelle
- interner Programmspeicher
- geringer externer Schaltungsaufwand

[2] In System Programming
[3] Puls-Weiten-Modulation

Auswahl des Mikrocontrollers

Aus einer Übersicht aller Mikrocontroller auf der Internetseite von Atmel habe ich dann den Mega32 im TQFP Gehäuse gewählt. Dieser beinhaltet Lösungsmöglichkeiten bzgl. aller o.g. Anforderungen. Ich habe darauf verzichtet das komplette Datenblatt im Anhang einzufügen, da es 317 Seiten umfasst. Selbstverständlich befindet es sich aber auf der CD der Dokumentation.

Display

Als Schnittstelle zwischen Ladegerät und Anwender sollte ein Display mit Touch-Panel verwendet werden. Die Firma Electronic Assembly GmbH stellte mir auf Anfrage ein solches Display kostengünstig zu Verfügung. Dieses EA-KIT 240-6 mit einer Größe von 240*64 Pixel hat einen internen Speicher für Bilder und Makros. Mit diesen Makros kann unter anderem eine Menüführung realisiert werden, die gleichzeitig die hierfür erforderlichen Ressourcen im µController spart. Außerdem kann jede der 48 Touch-Tasten einzeln (oder mehrere auf einmal) programmiert werden. Die Kommunikation mit dem µController erfolgt wahlweise über RS-232- oder I²C-Schnittstelle. Der Einfachheit halber erfolgte die Kommunikation in diesem Projekt über die RS232-Schnittstelle.
Das Datenblatt des Displays befindet sich in Anhang A.

Festlegen der Programmiersprache

Zur Auswahl standen drei mögliche Programmiersprachen:
Assembler, Basic oder C.
Der Vorteil von Basic liegt darin, dass diese Programmiersprache sehr einfach zu erlernen und zu handhaben ist. Der große Nachteil jedoch liegt darin, dass der übersetzte Quellcode sehr überladen ist und bei einem umfangreichen Projekt schnell den Programmspeicher im Prozessor übersteigt. Um dieses Risiko von vorne herein zu verhindern wurde Basic nicht weiter in Betracht gezogen.
Die Programmiersprache Assembler erzeugt den kleinsten Programmcode, da es sich um reine Befehle handelt, die direkt von der CPU verstanden werden. Dennoch fiel die Wahl letztendlich auf C. Diese Programmiersprache ist einfach erlernbar, bietet darüber hinaus jedoch den Vorteil, dass der Quellcode übersichtlicher ist als in Assembler. Dies gilt insbesondere für möglicherweise nachträglich Programmierarbeiten, die dann eine erneutes Einarbeiten ins Projekt wesentlich vereinfachen.

Erlernen der Programmiersprache

Um die Sprache C im Umgang mit dem gewählten Mikrocontroller zu erlernen, habe ich auf ein AVR Controllerboard von Kai Sachsenheimer zurückgegriffen, welches ich nach intensiver Suche im Internet als sehr günstige und sehr gute alternative zu den käuflich zu Erwerbenden Entwicklungskits identifizierte. Dieses wäre im Übrigen, in leicht abgewandelter Weise, auch für den Unterricht an der Schule eine gute Alternative zum aktuellen Z80. Das Layout der Platine, sowie eine Beschreibung

stehen unter http://www.home.fh-karlsruhe.de/~saka0012/ zu Verfügung. Die Beschreibung des Controllerboards befindet sich in Anhang B.
Die Platine wurde freundlicherweise von der Lehrlingswerkstatt der DaimlerChrysler AG in Sindelfingen geätzt.

Test einzelner Funktionen

Um den µController kennen zu lernen und die Programmiersprache C zu erlernen, wurden erst einfache Dinge, wie beispielsweise digitale Eingänge und Ausgänge programmieren, getestet. Schritt für Schritt folgten komplexere Funktionen hinzu. Im einzelnen wurden folgende Funktionen getestet:

- Digitale I/O
- Analoge Eingänge einlesen und über LED anzeigen
- PWM-Ausgang
- PWM-Steuerung über Analogwerte
- RS-232 Schnittstelle
- Kombination von PWM-Steuerung über Analogwerte und RS-232 Ausgabe
- Kommunikation zwischen µ-Controller und Display

Die Kommunikation zwischen µController und Display stellte sich als schwieriger heraus als ursprünglich erwartet, da einzelne Bytes unterschiedliche Interpretationen des Displays zulassen. Es ist dem Display also explizit bekannt zu geben, wie es die vom µController gesendeten Bytes zu verstehen bzw. weiter zu verarbeiten hat. So kann beispielsweise ein Byte als einfache Hex-Zahl verarbeitet, oder als darstellbares ASCII-Zeichen auf dem Display wiedergegeben werden. So wird z.B. 4Fh als Zahl 79 dargestellt oder aber durch den Buchstaben ‚O'. Zur Unterscheidung benötigt das Display deswegen spezielle Kommandosequenzen, die alle mit „ESC" (Escape) beginnen. Dem entsprechend interpretiert es alle Informationen, die nicht mit „ESC" beginnen, zunächst als ASCII-Zeichen.
Auf der anderen Seite kann es ebenso leicht zu Fehlinterpretationen kommen, möchte man beispielsweise eine „1" (im ASCII-Code 31h) in eine Mathematische 1 umrechnen, um mit dieser arithmetische Operationen durchzuführen, so muss 30h oder 48 dezimal davon subtrahiert werden, um den korrekten Wert im binären Format zu erhalten. Diese Aufgaben waren nach Kenntnis jedoch schnell in den Griff zu bekommen.

Prinzip der Menüführung

Um die Kommunikation zwischen Display und Prozessor zu verringern werden die einzelnen Menüs als Makros im Display gespeichert. Diese werden dann nach und nach von µController angesprochen. Dies hat den Vorteil, dass dem Prozessor nicht mitgeteilt werden muss in welchen Menü man sich befindet, erspart also unnötige und komplizierte Kommunikation.
Außerdem können etwaige Hilfetexte dort abgelegt werden. Des weiteren ermöglicht das Display das Darstellen von Piktogrammen und Bildern, sofern dies für die Bedienung relevant sein sollte oder diese vereinfachen. Sogar kleine Animationen sind möglich.

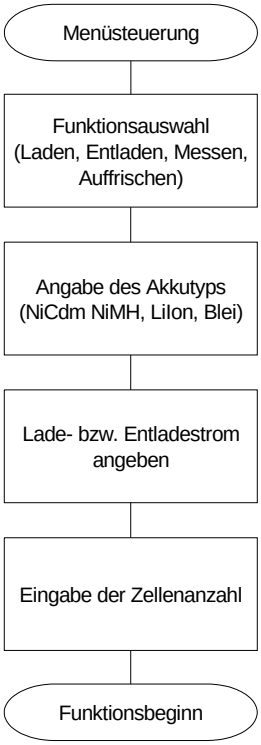

Abbildung 1: Grobstruktur Menüführung

Die Hardware

Da die zu ladenden Akkupacks aus mehreren Zellen (in Reihe geschaltet) bestehen die dadurch eine gesamte Spannungen von bis zu 45 V bereitstellen können, die Versorgung des Ladegerätes jedoch aus einer 12 V Autobatterie erfolgen sollte, musste ein Schaltnetzteil integriert werden. Als überschlägig berücksichtigte Leistung wurde hierfür ca. 300 W berechnet. Die exakte Leistung wird später in der Dokumentation genannt.

Die Hardware besteht im wesentlichen aus drei Teilen:

- Step-Down Wandler
- Step-Up Wandler
- µController für Regelung/Steuerung

Step-Down Wandler

Der Step-Down Wandler (Abb. 2) erzeugt alle Spannungen, die kleiner als die Eingangsspannung sind. Hierbei wird über einen P-Kanal FET die Eingangsspannung mittels PWM so geregelt, dass die gewünschte Spannung (oder Strom) erzielt wird. Für die Dimensionierung habe ich im Internet die Seite von Prof. Dr.-Ing. Heinz Schmidt-Walter von der FH-Darmstadt identifiziert (http://schmidt-walter.fbe.fh-darmstadt.de/), auf dessen Basis ich die erforderliche Drossel bestimmt habe. Dies war besonders hilfreich, da die exakte Dimensionierung von Schaltnetzteilen von vielen Faktoren abhängig sind, deren präzise Berechnung den Rahmen dieser Arbeit übersteigen würde. So spielt die Schaltfrequenz eine maßgebliche Rolle. Je höher die Schaltfrequenz, desto kleiner kann die Drossel gewählt werden. Gleichzeitig erhöhen sich mit der Schaltfrequenz aber auch die Verluste am Transistor. Da ich bei der Schaltfrequenz sehr gebunden war, fiel die Wahl auf 31,25 kHz, da diese Frequenz vom µController bei 16 MHz Taktfrequenz und einer PWM-Auflösung von 9 Bit erzeugt wird.

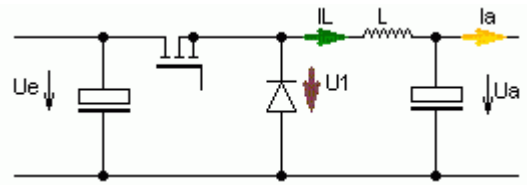

Abbildung 2: Prinzipschaltbild Step-Down Wandler

Step-Up Wandler

Der Step-Up Wandler (Abb. 3) erzeugt ebenfalls mittels PWM alle Spannungen die größer als die Eingangsspannung sind. So wurden für dieses Projekt folgende Daten angestrebt:

U_a = 50 V bei I_a = 6A

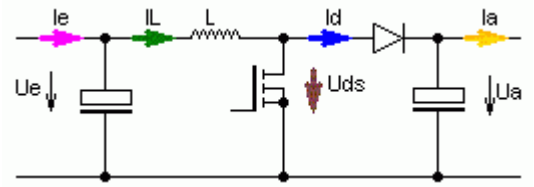

Abbildung 3: Prinzipschaltbild Step-Up Wandler

Die hohe Ausgangsspannung wird hierbei durch das Laden einer Drossel mit Ferritkern mit anschließendem Entladen und nutzen der Induktionsspannung erzeugt. Zum Laden der Drossel wird die Ausgangsseite der Drossel kurzzeitig über den FET nach Masse geschalten. Der dabei fließende Strom erzeugt ein Magnetfeld, dessen Energie im Ferritmaterial gespeichert wird. Die gespeicherte Energie wird, wenn der Transistor wieder sperrt, dann anschließend über die Diode an den Ausgang entladen. Der Kondensator am Ausgang soll die Spannung glätten und den Strom während den Pulspausen zu Verfügung stellen. An die Kondensatoren wer-

den dabei besondere Anforderungen gestellt. Sie müssen sehr strombelastbar sein, um den geforderten Strom am Ausgang auch liefern zu können und die hohe Frequenz verkraften. Die hier verwendeten Kondensatoren sind Low-ESR[4] Kondensatoren, die speziell für die Anwendung in Schaltnetzteilen konstruiert sind. Diese Kondensatoren sind in der Lage einen größeren Strom zu liefern als herkömmliche. Den errechneten Spannungs- bzw. Stromverlauf zeigt Abbildung 4 auf.
Da Step-Down sowie Step-Up Wandler eine Drossel benötigen, werden beide Schaltungen kombiniert (siehe Leistungsteil). Für beide Schaltungen wurde jeweils eine Drossel errechnet, was bedeutet, dass eine der beiden Drosseln nicht ideal sein kann. Die Wahl viel dann auf die Drossel für den Step-Up Wandler, damit sichergestellt ist, dass die Spannung am Ausgang auch erreicht wird. Für den Step-Down Wandler ist die Abweichung von der idealen Drossel nicht so relevant, da durch geringere Pulsweite auf jeden Fall die geringen Spannungen erzielt werden können.

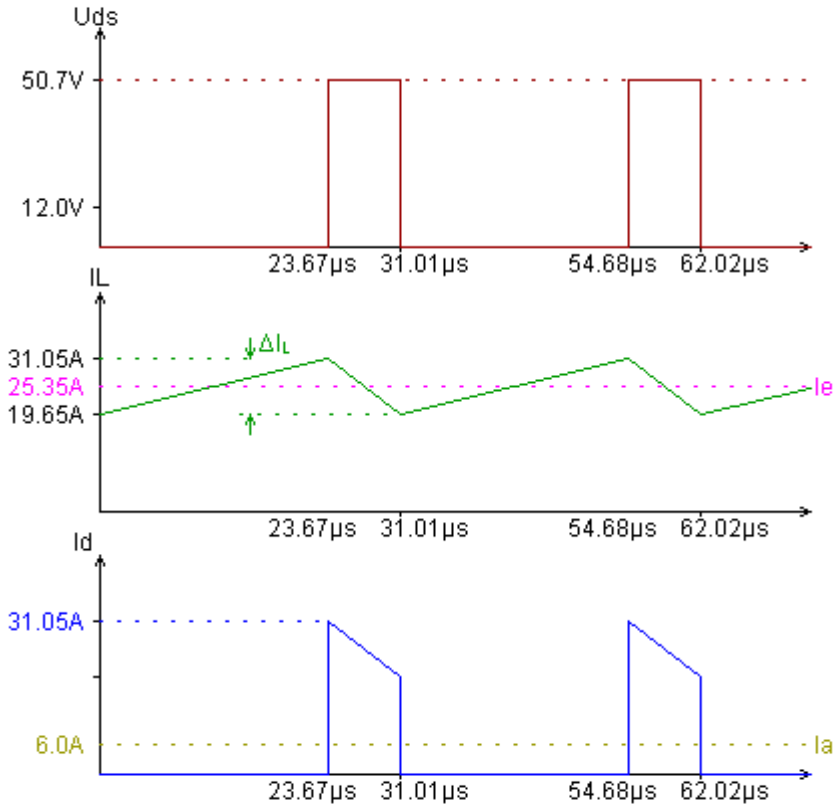

Abbildung 4: Strom- / Spannungsverlauf

[4] Low Equivalent Series Resistance = kleiner Ersatzreihenwiderstand

Ich habe diese Art von Schaltnetzteil gewählt, weil sich beide Schaltungen leicht miteinander kombinieren lassen und weil die Schaltung relativ leicht zu verstehen ist. Der Steuerungsaufwand der Transistoren ist hierbei nicht groß. Ein weiterer Vorteil liegt darin, dass die Stromaufnahme (ohne Verluste, also theoretisch) nur 31 A beträgt. Bei anderen Schaltungen die auf der Internetseite aufgezeigt werden, ist die Stromaufnahme wesentlich höher. Beim Sperrwandler sind es, bei gleichen Ausgangsdaten, z.B. 99 A, um nur ein Beispiel zu nennen.

Regelung / Steuerung

Der anfangs geplante Atmel 90S8535 musste aufgrund der geringen Taktfrequenz und der daraus resultierenden geringen PWM-Frequenz gegen einen schnelleren µController ersetzt werden. Nach erneuter Suche fiel die Wahl auf den Mega32, ebenfalls von Atmel. Dies hatte den Vorteil, dass die Software nicht neu geschrieben, sondern lediglich dem Compiler der neue Controller mitgeteilt werden musste. Für die Steuerung wurden zyklisch Daten wie Eingangsspannung, Kühlkörpertemperatur und Akkutemperatur abgefragt. Ausgangsspannung, Ladestrom bzw. Entladestrom wurden für die Regelung ebenfalls zyklisch abgefragt. Um diese Daten zu erfassen werden sechs Analogeingänge benötigt. Um die Ausgangsspannung, Ladestrom bzw. Entladestrom zu regeln, werden 3 PWM-Ausgänge benötigt, wobei immer nur einer der drei PWM-Ausgänge aktiv sein darf.
Um dies zu realisieren wurden beide PWMs abhängig von einander geregelt. So konnte der Step-Up Wandler erst starten, wenn der Step-Down seinen max. Wert erreicht hat.
Der Strom für das Laden und das Entladen wird mit einer „Softstart“-Funktion auf den gewünschten Wert geregelt. Das heißt der Wert, ausgehend von 0, für den PWM-Ausgang wird mit jedem Durchlauf des Programms jeweils um eins inkrementiert. Dies wird solange wiederholt bis der gewünschte Strom erreicht ist.
Dadurch wird ein hoher Einschaltstrom vermieden. Außerdem lässt sich nicht errechnen, welcher Wert nun vorgegeben werden muss, um den Ladestrom zu erreichen, da immer verschiedene Akkus mit verschiedenen Strömen und Spannungen geladen werden. Die einzelnen Ladecharakteristiken wurden in eigenen Lade-/Entladeprogrammen hinterlegt, da sich die Ladeschlussspannungen sowie das ΔU für die Ladeabschaltung unterscheiden.

Strommessung

Die ursprüngliche Idee war es, den eingebauten Differenzverstärker des µControllers hierfür zu verwenden. Diese Idee musste leider verworfen werden, da praktische Versuche folgende Probleme aufgezeigt hatten:

- nur 9-Bit Auflösung, da der Messbereich von $-V_{Ref}$[5] bis $+V_{Ref}$ reichte, d.h. Werte von -512 bis +511, das allein kein Problem darstellte, aber gegenüber 0V (Masse) nicht negativ sein darf
- die Linearität war nicht gegeben
- der zu erfassende Messbereich reicht von –60 mV bis +60 mV, ist also sehr klein
- mit internen Differenzverstärker war nur eine Auflösung im Ampere-Bereich möglich

Aus diesen Gründen wurden diese kleinen Spannungen mittels einer nicht-invertierenden Operationsverstärkerschaltung um das 75-fache auf 4,5 V (V_{Ref}) verstärkt. Das funktioniert aber nur solange, bis der Akkupack entladen wird. Dann ist aus schaltungstechnischen Gründen der zu messende Strom negativ. Also wird mit einem zweiten OP der Messwert invertierend verstärkt und auf einen zweiten Kanal des A/D-Wandlers geführt (Abb. 5).
So erhält man einen Kanal für den Ladestrom und einen Kanal für den Entladestrom. Die somit erzielte Auflösung von 5,8mA auf beiden Kanälen ist mehr als ausreichend.

$$\frac{6\ A}{2^{10}\ Bit} = \underline{\underline{5{,}86\ mA}}$$

Zu Test- und Programmierzwecken wurde eine kleine Platine mit den notwendigen Änderungen an die bestehende Platine angefügt. Eine neue Platine wurde erst kurz vor Beendigung des Projektes geätzt, da mit noch mehr Änderungen zu rechnen war.

[5] V_{Ref} Referenzspannung des A/D-Wandlers

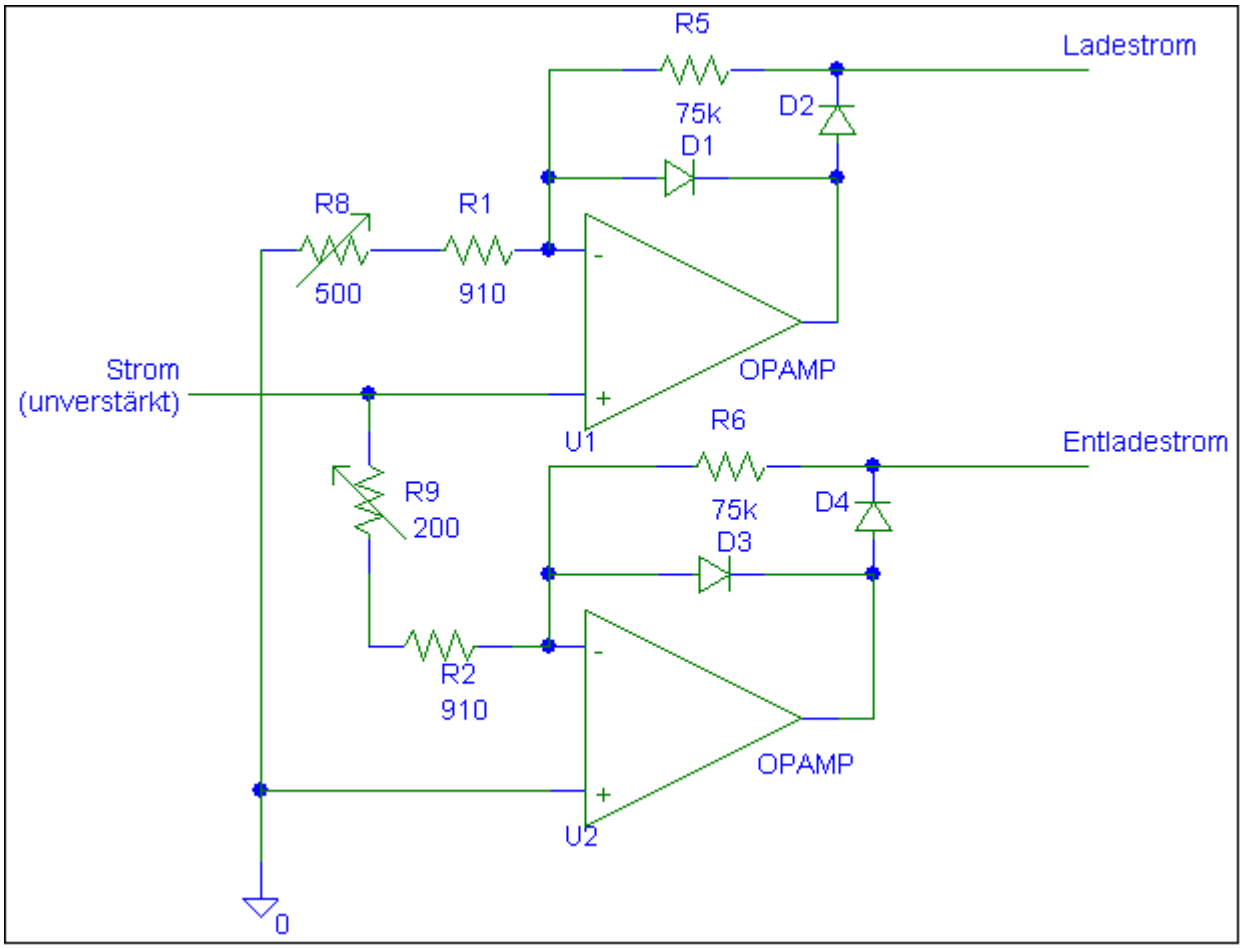

Abbildung 5: Prinzip der OP-Verstärkerschaltung

Spannungsmessung

Gemessen wird außer der Akkuspannung noch die Eingangsspannung, mit der der Lader versorgt wird. Da dieses Ladegerät für den mobilen Einsatz konzipiert ist und die Speisung aus einer Kfz-Batterie erfolgen soll, ist ein Spannungsbereich von 10,8 V – 13,8 V vorgesehen. Unterschreitet die Eingangsspannung die Mindestspannung von 10,8 V, wird der momentane Ladevorgang mit Fehlermeldung abgebrochen. Das Starten eines Ladevorgangs wird dann verhindert. Die Messung erfolgt über einen Spannungsteiler, der so bemessen ist, dass bei 13,8 V ca. 4,5 V am Analogeingang Port A0 des µControllers anliegen.
Durch die ungenügende Auflösung des internen A/D-Wandlers von 10 Bit ist es leider nicht möglich, das ΔU von ca. 8 mV zu messen, das entsteht, wenn der Akku (z.B. NiMH) voll ist. Ausgehend von 50 V max. Spannung, die am Akkupack entstehen kann, wäre mit dem internen A/D-Wandler nur eine Auflösung von ca. 49 mV möglich. Deshalb musste ein externer A/D-Wandler eingebaut werden. Bei 12 Bit ist eine Auflösung von ca. 14mV möglich. Für die Δ/U-Erfassung ist dies noch immer nicht ausreichend. Also müssen es mehr als 12 Bit sein. Die Wahl fiel dann auf den ADS 8320 mit 16 Bit Auflösung. Einen besonderen Grund (außer seiner großen Auflösung) gab es nicht, nur war dieser eben gerade greifbar. Möglich wäre jeder A/D-Wandler ab 14 Bit. Die Auflösung bei 16 Bit beträgt ca. 760 µV. Die Daten ge-

langen dann mittels einer Takt- und einer Datenleitung in den µController. Dieser erzeugt für die Wandlung an Port B0 (Pin 40) einen Takt von ca. 3,5kHz. Mit den ersten fünf Takten wird der Messwert gewandelt. Mit den nächsten 16 Takten werden die Daten an den µController gesendet und werden am Port B1 (Pin 41) empfangen. Die gesamte Wandlungszeit inklusive der Übertragung beträgt bei 3,5 kHz 6,3 ms. Der Ausgang PB2 (Pin 42) aktiviert nur den A/D-Wandler, der, wenn er nicht gebraucht wird, in den Power-Down Mode geht. Theoretisch ist eine Wandlungsgeschwindigkeit von 220 µs möglich. Dann muss aber mit einer größeren Ungenauigkeit gerechnet werden. Da die Anwendung hier nicht zeitkritisch ist, genügen die 3,5 kHz vollkommen. Das Datenblatt zum ADS8320 befindet sich im Anhang A, sowie auf der CD der Dokumentation.

Temperaturmessung

Es werden zwei Temperaturen gemessen, die für die Laderegelung relevant sind. Zum einen wird die Temperatur mittels NTC am Akkupack gemessen. Um den Akku nicht zu beschädigen, wird das ΔT gemessen. Übersteigt dieses einen gewissen Wert, wird auf Erhaltungsladung umgeschaltet. Steigt die Temperatur über einen maximalen Absolutwert, wird die Ladung mit Fehlermeldung abgebrochen.
Zum zweiten wird die Temperatur des Kühlkörpers gemessen, um einen Schaden an der Elektronik zu verhindern. Am Kühlkörper wird ebenfalls das ΔT überwacht und gegebenenfalls der Strom reduziert. Falls dennoch die Temperatur den Grenzwert übersteigt, wird mit der entsprechenden Fehlermeldung der Ladevorgang abgebrochen. Wie beim Akkupack wird die Temperatur hier mit einem NTC gemessen. Die Nichtlinearität des NTC wird mittels Software kompensiert. Die daraus entstehende Ungenauigkeit spielt hierbei keine Rolle.

Kühlkörper

Die angestrebte Ausgangsleistung des Laders beträgt 300 W bei 50 V und 6 A. Dies bedeutet dass bei minimaler Eingangsspannung von 10,8 V am Eingang mindestens 28 A fließen. Da kein Bauteil frei von Verlusten ist, wird für die Kühlkörperberechnung ein Eingangsstrom von max. 50 A angenommen.
Die P-Kanal FETs haben jeweils ein R_{DSOn}[6] von 20 mΩ. Dies hat zur Folge, dass an ihnen eine max. Verlustleistung von 25 W entsteht. Die N-Kanal FETs erzeugen wegen des geringeren Stromes und des besseren R_{DSOn} von 8 mΩ nur eine Verlustleistung von ca. 1 W. Dieser Wert gilt nur, wenn die Transistoren statisch angesteuert werden. Da sie aber in einem DC/DC-Wandler eingesetzt werden und daher ständig schalten, wird aus Sicherheitsgründen der 10fache Wert angenommen.

[6] R_{DSOn} Durchlasswiderstand des Transistors

Kühlkörperberechnung

Für die Berechnungen wird eine max. Umgebungstemperatur von 30°C angenommen.

Berechnung für IRF4905:

ϑ_j[7]=175 °C R_{thJC}[8]=0,75 K/W R_{thCS}[9]=0,5 K/W

$$P_V = (R_{DSOn1} \| R_{DSOn2}) \times I^2 = (0{,}02\ \Omega \| 0{,}02\ \Omega) \times (50\ A)^2 = 25\ W$$

$$R_{th} = \frac{\vartheta_j - \vartheta_U}{P_V} = \frac{175\ °C - 30\ °C}{25\ W} = 5{,}8\ K/W$$

$$R_{thS} = R_{th} - R_{thjC} - R_{thCS} = 5{,}8\ K/W - 0{,}75\ K/W - 0{,}5\ K/W = 4{,}55\ K/W$$

Berechnung für STP60NE06L-16:

ϑ_j=175°C R_{thJC}=0,94 K/W R_{thCS}=0,5 K/W

$$P_V = (R_{DSOn1} \| R_{DSOn2}) \times I^2 = (0{,}016\ \Omega \| 0{,}016\ \Omega) \times (10\ A)^2 = 0{,}8\ W$$

$$R_{th} = \frac{\vartheta_j - \vartheta_U}{P_V} = \frac{175\ °C - 30\ °C}{10\ W} = 14{,}5\ K/W$$

$$R_{thS} = R_{th} - R_{thjC} - R_{thCS} = 14{,}5\ K/W - 0{,}94\ K/W - 0{,}5\ K/W = 13{,}06\ K/W$$

[7] ϑ_j max. zulässige Sperrschichttemperatur
[8] R_{thJC} Wärmewiderstand Sperrschicht - Gehäuse
[9] R_{thCS} Wärmewiderstand Gehäuse - Kühlkörper (inkl. Leitpaste)

Der Kühlkörper, der vor der Berechnung und zu Testzwecken gewählt worden war, hat einen Wärmewiderstand von 2,3 K/W. Diese intuitive Wahl bleibt auch weiterhin im Einsatz.
Temperaturmessungen in der Entwicklungsversion des Laders, während des Betriebes mit einem NiCd-Akkupack mit 8 Zellen und einem Ladestrom von 1,5 A, hat keinen nennenswerten Temperaturanstieg gezeigt.

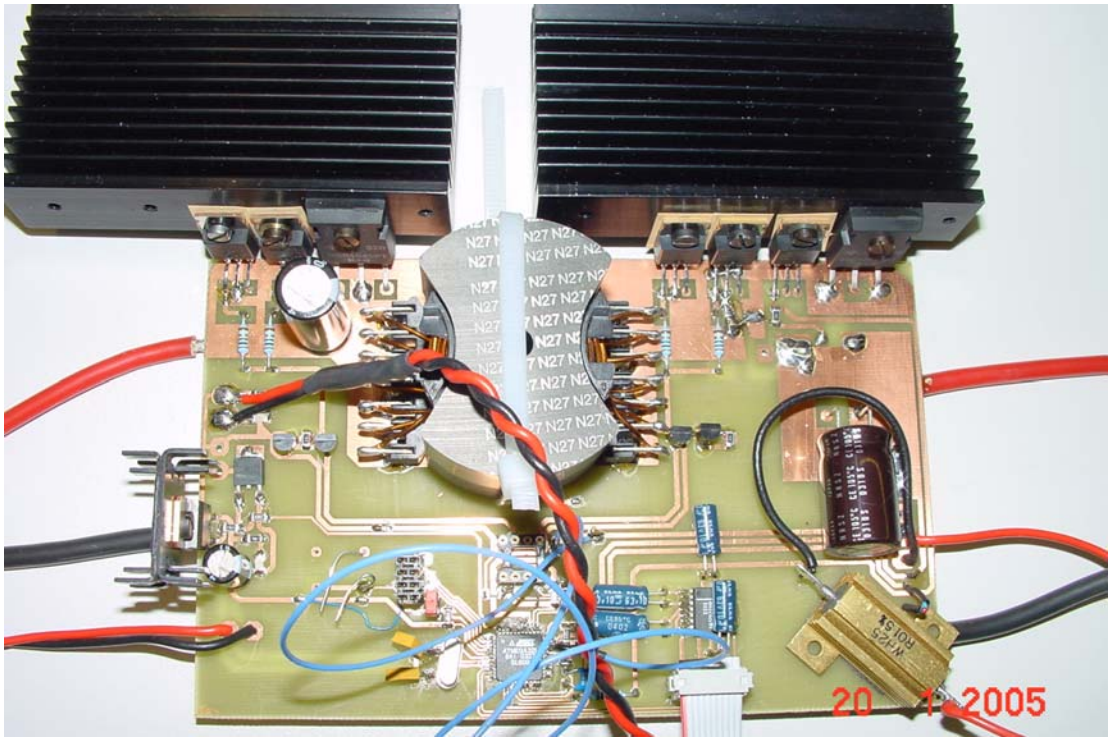

Abbildung 6: Prototyp des Ladegerätes

Auch die in Abb. 6 gezeigte Version des Ladegerätes existierte in der ursprünglichen Form nicht lange. So wurden ein externer A/D-Wandler und eine Operationsverstärkerschaltung hinzugefügt. So ging die Entwicklung von der ersten Version auf einer Lochrasterplatine über insgesamt 21 Schritte, bis zur endgültigen Form.

Platine

Für die Entwicklung der Schaltung sowie für das Entwerfen des Platinen-Layouts wurde TARGET 3001! vom Ingenieurbüro Friedrich in der Schüler-Version verwendet. Es wurden am Anfang auch Versuche mit Eagle gemacht, aber schnell verworfen, weil die Software zum Abstürzen neigte. Was ebenfalls nur schwer möglich war, war das Erstellen von eigenen Bauteilen, die nicht in der Bauteilbibliothek hinterlegt sind. Auch dies ist in TARGET problemlos möglich. Das sind jedoch nur subjektive Betrachtungen.

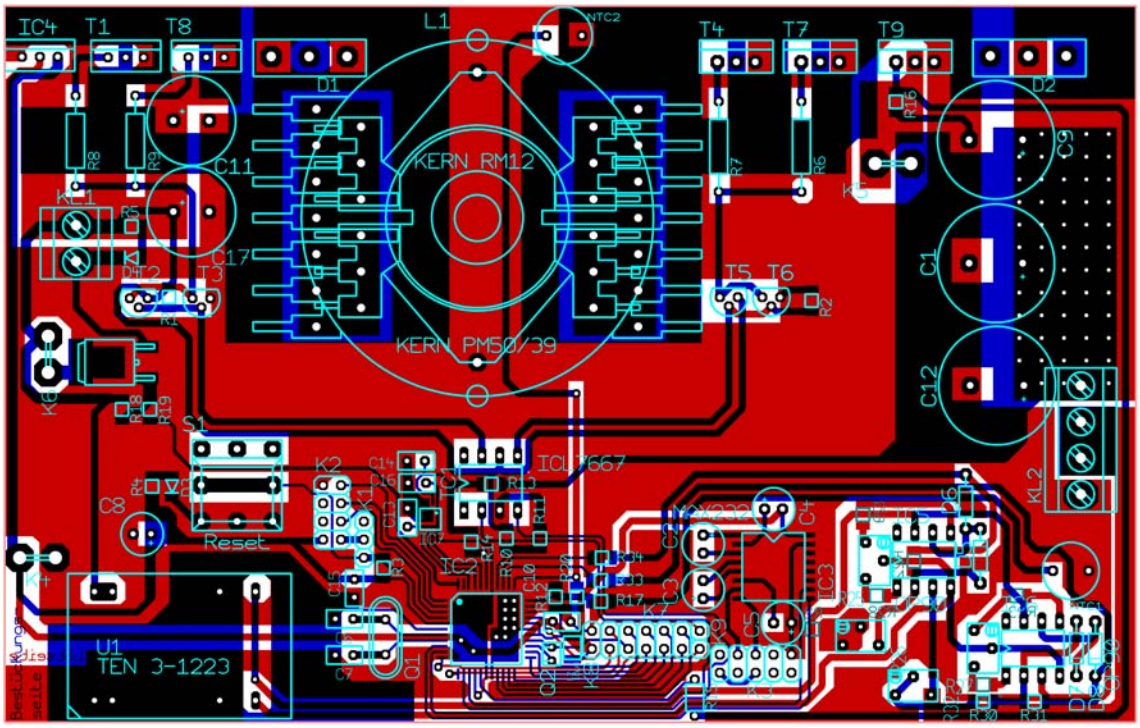

Abbildung 7: Platinen-Layout

Software

Menü

Vor Beginn der Programmierung sollte das Menü (jedenfalls grob) in seiner Struktur feststehen. Das soll sicherstellen, dass bei der Programmierung nichts vergessen wird. Die Menüstruktur wird nachstehend anhand eines NiCd-Akkus mit 8 Zellen und einem Ladestrom von 1,5 A beschrieben, da das Laden von Akkus die häufigste Funktion des Laders sein dürfte.
In der ersten Auswahl nach dem Einschalten des Laders erscheint das folgende Bild.

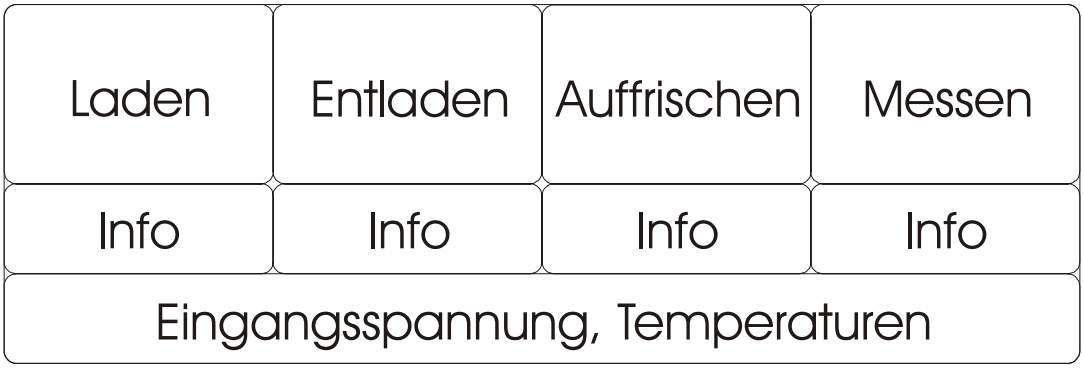

Abbildung 8: Auswahlmenü

Neben der Funktionsauswahl ist jeweils ein Info-Feld vorhanden, das bei Betätigung zur jeweiligen Funktion eine kurze Beschreibung anzeigt. Man hat dann die Möglichkeit, die Information in Ruhe zu lesen, denn erst bei erneutem Antippen des Displays kehrt man zurück in das vorherige Menü. Außerdem kann man sich die Eingangsspannung, sowie die Kühlkörper- und Akkutemperatur (sofern ein Akku angeschlossen ist) anzeigen lassen. Nach Betätigen der „Taste“ Laden erscheint die zweite Auswahl (Abb. 10). Nun wählt man den Akkutyp, den man laden möchte. Im Beispielfall „NiCd“.

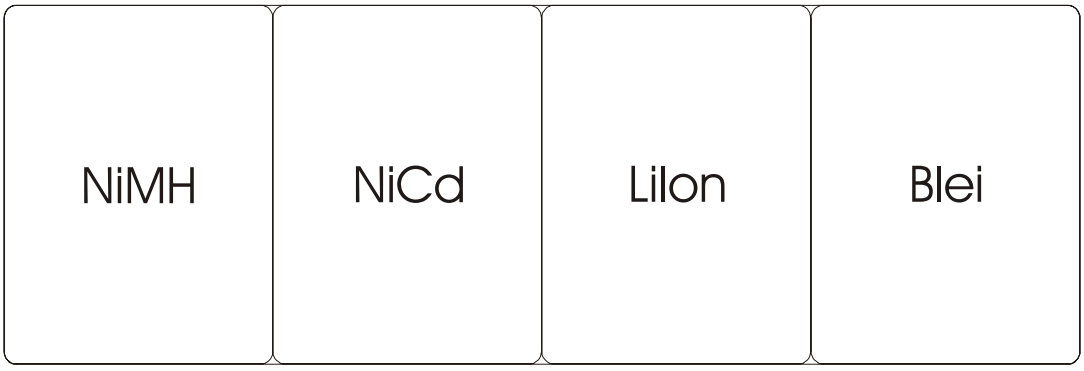

Abbildung 9: Auswahl Akkutyp

Bei der Eingabe des Ladestroms sollte man sich genau überlegen, ob der angeschlossene Akku diesen Strom auch verträgt. Bei Eingabe eines zu großen Ladestrom kann der Akku Schaden nehmen. Es wird zwar die Akkutemperatur gemessen und bei Überschreitung der Maximaltemperatur der Ladestrom reduziert bzw. der Ladevorgang abgebrochen, der Akku dürfte jedoch dann schon beschädigt worden sein. Deshalb im Zweifel einen kleineren Ladestrom wählen.

Abbildung 10: Eingabe Ladestrom

Falls man sich bei der Eingabe vertippt haben sollte, kann mit „C“ die Eingabe erneut getätigt werden. Es können Ströme zwischen 1 mA und 6000 mA eingegeben werden. Die Schrittweite beträgt hierbei 1 mA. Bei ungültiger Eingabe (0 mA oder >6000 mA) erscheint eine Fehlermeldung und die Eingabe muss erneut gemacht werden. Ist man sich sicher und bestätigt mit „OK“ die Eingabe, folgt die Eingabe der Zellenanzahl. Das Limit hierbei ist 20 Zellen. Sollte man dennoch null Zellen oder mehr als 20 angegeben haben, erfolgt wieder eine Fehlermeldung, und es bedarf einer erneuten Eingabe.

Abbildung 11: Eingabe Zellenzahl

Auch hier kann durch „C“ die Eingabe korrigiert werden. Nach dem Drücken von „OK“ ist der Lader bereit für den Ladevorgang. Ist noch kein Akku angeschlossen, wartet das Ladegerät solange, bis ein Akku angeschlossen ist. Ist jedoch schon ein Akku angeschlossen, startet der Ladevorgang unverzüglich.

Nach dem Start des Ladevorgangs erscheinen abhängig vom Ladestatus noch zwei Bilder im Display. Während des Ladens erscheint der Displayinhalt wie in Abb. 13 dargestellt.

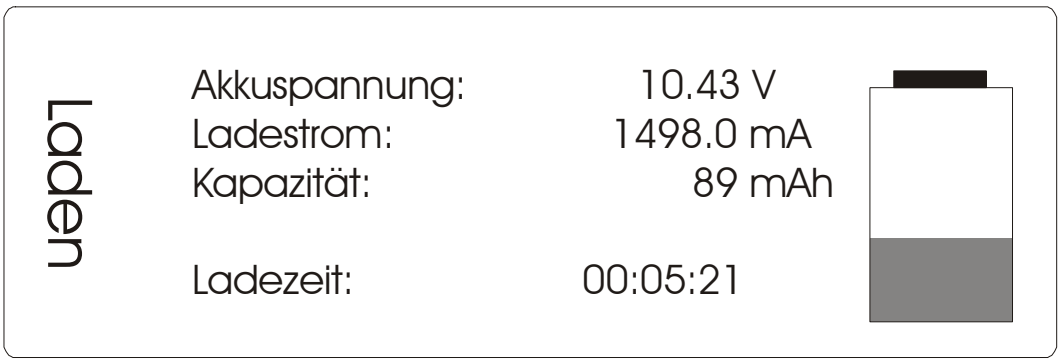

Abbildung 12: Displayinhalt während des Ladens

Die Akkuspannung sowie der Ladestrom werden ständig gemessen und angezeigt. Die eingeladene Kapazität wird jede Sekunde aktualisiert und errechnet sich wie Folgt:

$$neuer\ Wert = \frac{\frac{Strom}{3{,}6} + alter\ Wert}{1000}$$

Dabei ergibt der neue Wert nach einer Sekunde den alten Wert. Die Ladezeit wird im Format hh:mm:ss angezeigt.
Der rechts im Display stilisierte Akku visualisiert zusätzlich den Füllstand des Akkus mittels eines Bargraph. So kann mit einem schnellen Blick auf das Display festgestellt werden, wie voll der Akku ist. Ist der Akku dann voll aufgeladen, schaltet das Ladegerät automatisch vom Schnellladem odus in den Erhaltungsmodus. Der Strom wird dann auf ein Bruchteil des Ladestroms reduziert, um ein Überladen des Akkus zu verhindern. Dabei wechselt der Displayinhalt entsprechend und zeigt dann nur noch die Akkuspannung und die eingeladene Kapazität wie in Abb. 14 an.

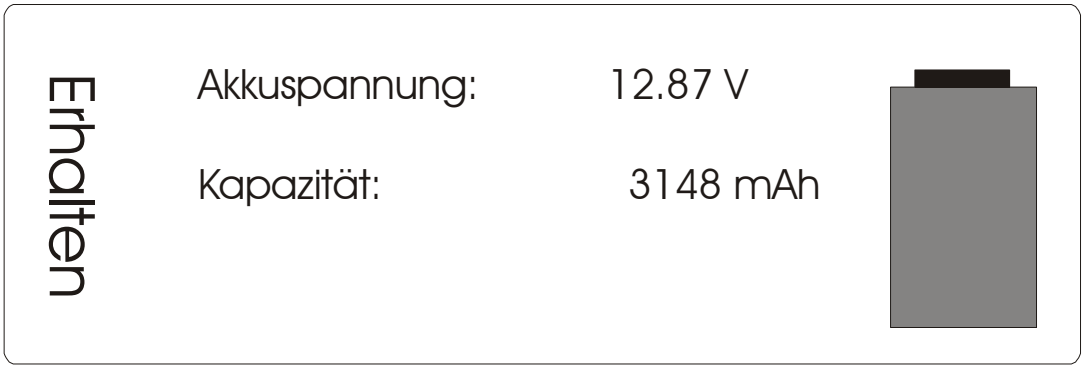

Abbildung 13: Erhaltungsmodus

Software

Für die Programmierung des µControllers wurde WinAVR verwendet. Die Software umfasst ein Packet das einen Editor, einen Compiler und verschiedene Tools beinhaltet. WinAVR kann kostenlos von http://sourceforge.net/projects/winavr herunter geladen werden. Zum Programmieren genügt allerdings „Programmers Notepad“ (der Editor). Von dort aus kann man den Compiler und den Programmer bedienen. Da das gesamte Programm zu umfangreich ist, werde ich hier nur auszugsweise ein paar einzelne Funktionen beschreiben. Ein Ausdruck der Software ist aber im Anhang D einzusehen. Der Sourcecode ist, soweit es Sinn macht, kommentiert und sollte somit relativ leicht nachvollziehbar sein.

Serielle Schnittstelle

Zum Ansteuern des Displays bediene ich mich, der Einfachheit halber, der seriellen Schnittstelle die der Controller bietet. Um diese zu konfigurieren sind nur wenige Befehle nötig. Zunächst wird die Baudrate im USART BAUD RATE REGISTER (UBRR) eingestellt. Dieses ist ein 16-Bit Register und erzwingt eine Umrechnung der Baudrate.

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

Mit dieser Formel (siehe auch S. 141ff. des Datenblatts) wird dann die Baudrate eingestellt. Danach wird das Senden und Empfangen freigegeben sowie das Datenformat 8 Datenbits, 1 Stopbit, keine Parität eingestellt. Im Programm wird dieses wie Folgt umgesetzt:

```
#define BAUDRATE 9600
#define CLOCK   16000000
unsigned char baud = (CLOCK / (BAUDRATE * 16L) - 1);       // Baudrate berechnen
UBRRH = (baud >> 8);                                       // Baudrate einstellen
UBRRL = baud;                                              // Baudrate einstellen
UCSRB = (1 << TXEN) | (1 << RXEN);                         // Senden und Empfangen freigeben
UCSRC = (1 << URSEL) | (3 << UCSZ0);                       // Datenformat festlegen
```

Um Daten über die Schnittstelle zu senden wurde die Funktion UART_SEND() entwickelt. Zum einen werden immer wieder Daten gesendet und zum anderen weil es den Programmcode reduziert.

```
void UART_SEND(char *senden)
{
        for (;*senden !='\0';senden++)       // Schleife bis Daten gesendet
        {
        loop_until_bit_is_set(UCSRA, UDRE);  // Warten bis Datenregister leer
        UDR = *senden;                       // Daten senden
        }
}
```

Nun werden nicht nur Daten gesendet, sondern auch Empfangen. Dies geschieht über das gleiche Register wie das Senden. Um zu erkennen ob ein Byte empfangen

wurde gibt es das Bit RXC des USART[10] Control and Status Register. Ist diese Bit 1 so wurde ein Byte empfangen und steht im UDR (USART Data Register) bereit. Das UDR wird automatisch nach dem Auslesen gelöscht.

A/D Wandler

Um eine annehmbare Genauigkeit zu erreichen sollte die Wandlerfrequenz zwischen 50 und 200kHz liegen. Dazu wird der Systemtakt über Frequenzteiler entsprechend eingestellt. Da ich einen Systemtakt von 16MHz verwende muss ich den Vorteiler aus 128 einstellen. Dazu werden die Bits ADPS0...ADPS2 auf „1“ gesetzt. Damit ergibt sich eine Wandelfrequenz von 125kHz.
Somit ergibt sich eine Initialisierungssequenz des A/D Wandlers die folgendermaßen aussieht:

```
ADCSRA = (1 << ADEN)|(1 << ADSC)|(1 << ADPS2)
         (1 << ADPS1)|(1 << ADPS0);          // AD-Wandler init    [S.214]
while (inp(ADCSRA) & BV(ADSC));              // Warten bis ADC bereit
```

Nun ist der A/D Wandler bereit und kann jederzeit mit gestartet werden wenn man das Startbit im Status und Kontrollregister des A/D Wandlers setzt. Das Ergebnis der Wandlung steht dann im ADCL- bzw. ADCH-Regiser, da es sich um 10 Bit handelt. Beim Auslesen muss man darauf achten, dass man erst das Low-Byte (ADCL) liest, bevor man das High-Byte liest. Liest man das High-Byte zuerst kann es vorkommen, dass das Low-Byte zu einer anderen Wandlung gehört, da der Wandler nach dem auslesen des High-Byte mit einer neuen Wandlung beginnt.

PWM-Generator

Um die Transistoren anzusteuern benötige ich mehrere PWM-Ausgänge. Nun kann man diese mittels Software realisieren oder man greift einfach auf die Hardwaremäßig vorhandenen zurück. Es sind drei PWM vorhanden: zwei mit 10-Bit Auflösung und einer mit 8-Bit Auflösung. Der PWM wird über Timer/Zähler realisiert. Ab Seite 103 im Datenblatt findet man zahlreiche Konfigurationsmöglichkeiten den Timer oder Zähler zu betreiben. Das Datenblatt ist sehr gut und leicht verständlich, so dass man sehr schnell die richtige Einstellung findet.
Für die Regelung der Ladespannung und des Stromes verwende ich die beiden 10-Bit PWM, wobei diese wegen der benötigten Frequenz mit 9-Bit betrieben werden. Den dritten PWM verwende ich um das Entladen zu regeln. Dazu reichen die 8-Bit völlig aus.

```
TCCR1A  = (1 << COM1A1)|(1 << COM1B1)|(1 << COM1B0)
        |(1 << WGM12)|(1 << WGM11);          // PWM1 konfiguration (9-Bit)       [S.105]
TCCR1B  = (1 << WGM12);                      // Prescaler 1 (kein Prescaler)     [S.108]
OCR0 = 0x00;                                 // PWM2                             [S.80]
ICR1 = 0x0000;
TCCR0 = (1 << CS00)|(1 << COM00)|(1 << COM01);// Timer0 konfigurieren           [S.79]
TIFR  = (1 << TOV1) | (1 << TOV0);           // lösche Counter1+0 overflow flag
TCCR2 = (1 << COM21)
        |(1 << WGM20)|(1 << WGM21);          // PWM3 konfiguration (8-Bit)       [S.123]
TCNT2 = 0x00;                                // Counter Register für Timer2 laden
OCR2  = 0x00;                                // Compare Timer2 Register für Timer2 löschen
OCR1A = 0x0000;                              // Compare Timer1 Register A löschen
```

[10] USART = Universal Synchronous and Asynchronous serial Receiver and Transmitter

```
OCR1B = 0x0000;                         // Compare Timer1 Register B löschen
TCNT1  = 0x0000;                        // Lösche Counter1 L-Byte
timer_enable_int _BV(TOIE0) ;           // Timer0 Overflow Interrupt zulassen
```

Den Timer0 des PWM 1 und 2 verwende ich gleichzeitig zur Erzeugung eines Sekundentaktes. Dabei läuft der Timer0 mit Systemtakt und erreicht somit ca. alle 31µs einen Überlauf. Dabei wird ein Interrupt ausgelöst bei dem in einer Interruptroutine ein Zähler dekrementiert wird und somit ein Sekundentakt realisiert wird. Mit diesem Sekundentakt lassen sich dann ebenfalls Minuten und Stunden realisieren.

```
INTERRUPT(SIG_OVERFLOW0)
{
 if (0x00 == --time.t_count)        // Gleichzeitiges dekrementieren und Vergleichen auf 0
 {                                  // (bei 0 eine Sekunde vergangen)
  if (60 == ++time.sec)             // Sekunden inkrementieren, wenn = 60
  {
   if (60 == ++time.min)            // Minuten inkrementieren, wenn = 60
   {
    if (24 == ++time.hour)          // Stunden inkrementieren, wenn = 24
    {
     time.hour = 0x00;              // Stunden zurücksetzen
    }
    time.min = 0x00;                // Minuten zurücksetzen
   }
   time.sec = 0x00;                 // Sekunden zurücksetzen
  }
  time.t_count = 0xf420;            // Zähler neu setzen
 }
}
```

Externer A/D-Wandler

Wesentlich einfacher ist es den externen A/D-Wandler zu initialisieren. Bei ihm werden lediglich die Pins CS uns CLK auf High-Pegel gelegt.

```
void INIT_EXT_ADC(void)
       {
       sbi (PORTB, ADS8320_CS);     // ChipSelect auf High
       sbi (PORTB, ADS8320_CLK);    // Clock auf High
       }
```

Der Start einer Wandlung erfolgt dann über ein Low-Pegel an /CS und dem Takt an CLK. Der ADS8320 sendet nach Ende der Wandlung die Daten seriell über den Pin DOUT wobei das MSB[11] zu erst gesendet wird. Das Empfangen geschieht in einer eigenen Funktion, in der auch der Takt erzeugt wird.

[11] MSB Most Significant Bit (höchstwertigste Bit)

```
unsigned int READ_EXT_ADC(void)
{
 unsigned int x = 0;                  // Schleifenzähler
 unsigned int data = 0;               // Mittelwert aus 8 Messungen
 unsigned long int average = 0;       // Summe aus 8 Messungen
 for(x=0;x<=7;x++)                    // Mittelwert aus 8 Messungen bilden
 {                                    // um Impulsspitzen zu unterdrücken
  unsigned int i = 0;                 // Schleifenzähler
  cbi(PORTB, ADS8320_CS);             // ADS8320 aktivieren
  for(i=0;i<=5;i++)                   // Starte Wandlung
  {
   cbi(PORTB, ADS8320_CLK);           // Erzeugen eines Rechtecksignals (5 Pulse)
   delay_us(5);
   sbi(PORTB, ADS8320_CLK);
   delay_us(5);
  }
  for(i=0;i<16;++i)                   // Empfangen der Wandlung
  {
   cbi(PORTB, ADS8320_CLK);           // Erzeugen eines Rechtecksignals (16 Pulse)
   delay_us(5);
   data <<= 1;                        // Linksschieben um 1
   if(bit_is_set(PINB,ADS8320_DOUT)) // Prüfen auf High-Pegel
   data |= 0x0001;                    // Dann eine 1 in data
   sbi(PORTB, ADS8320_CLK);
   delay_us(5);                       // 15µs Warten
  }
  sbi(PORTB, ADS8320_CS);             // ADS8320 deaktivieren
  for(i=0;i<=2;i++)                   // Power-Down ADS8320
  {
   cbi(PORTB, ADS8320_CLK);           // Erzeugen eines Rechtecksignals (2 Pulse)
   delay_us(5);
   sbi(PORTB, ADS8320_CLK);
   delay_us(5);
  }
  average += data;                    // Summe bilden
 }
 data = average/8;                    // Mittelwert bilden aus 8 Messungen
 return(data);                        // Rückgabe des Wandlerwertes
}
```

Um den Rahmen der Softwaredokumentation nicht zu sprengen, belasse ich es bei diesen Auszügen aus dem Programm und verweise noch einmal auf den Ausdruck des Codes. Dieser kann im Anhang D eingesehen werden und sollte anhand der Kommentare relativ leicht verständlich sein.

Anhang A

Datenblätter

In diesem Teil finden Sie die Datenblätter zu den verwendeten Bauteilen. Zu finden sind:

- **Mega32** (µController, Zusammenfassung)
- **ADS8320** (A/D-Wanlder)
- **ICL7667** (MOSFET Treiber)
- **IRF4905** (P-Kanal MOSFET)
- **STP60NE06L** (N-Kanal MOSFET)
- **MBR4045PT** (Schottky-Diode)
- **MAX232** (Schnittstellentreiber)
- **OP90** (Präzisions-Operationsverstärker)
- **NTC** (Temperatursensor)
- **KIT 240-6** (Touch-Display)

Anhang B

Schaltplan

Anhang C

Anleitung des Testboard

Anhang D

Software

Anhang E

Quellen und Dank

Das fehlende Wissen, Anregungen, Tipps und Lösungen wurde aus folgenden Quellen bezogen:

- www.mikrocontroller.net/forum/
  Ein Forum rund um Mikrocontroller und Elektronik
- http://schmidt-walter.fbe.fh-darmstadt.de
  Eine Seite zum Berechnen von Schaltnetzteilen aller Art
- AVR-Mikrocontroller Praxis (ISBN 3-89576-063-3)
  Ein gutes Buch für das Programmieren in Assembler, speziell die Controller von Atmel
- C für Mikrocontroller (ISBN 3-7723-4156-X)
  Ein Buch für den Einstieg in die C-Programmierung von Atmel Controllern
- Softwareentwicklung in C für Mikroprozessoren und Mikrocontroller (ISBN 3-7785-2943-9)
  Weiterführendes Buch für die C-Programmierung verschiedener Typen

An dieser Stelle möchte ich mich bei der Firma Electronic Assembly GmbH in Gräfelfing bedanken, die mir den günstigen Erwerb eines Touch-Displays ermöglicht hat, bei den Usern des Mikrocontroller.Net-Forums für die schnelle Hilfe bei kleinen und größeren Problemen, meinem Kollegen Jens Rösener, der mich bei der Fehlersuche unterstützt hat, bei der Firma DaimlerChrysler in Sindelfingen für das Ätzen des Prototyps und das Herstellen des Gehäuses. Dank geht auch an meine Familie, die meine Launen ertragen musste, besonders meine Freundin Claudia.

Erklärung

Hiermit erkläre ich, dass die vorliegende Technikerarbeit eine eigenständige Leistung darstellt und nicht auf der Basis eines bereits vorhandenen Techniker- , Diplom- oder ähnlicher Arbeit erstellt wurde. Bei der Durchführung und Ausarbeitung wurden nur die zulässigen Hilfsmittel verwendet.

Mir ist bewusst, dass bei einem Verstoß gegen diese Erklärung innerhalb der gesetzlichen Einspruchsfristen auch im Nachhinein die Leistungsbewertung aberkannt werden kann. Damit erlischt die Berechtigung zum Tragen der Berufsbezeichnung des staatlich geprüften Technikers.

Hiermit versichere ich, dass ich folgende praktischen Teile und Gliederungspunkte selbstständig bearbeitet und verfasst habe.

Sindelfingen, den 1. Mai 2005 Unterschrift:..... (Danny Hiebel)

Außerdem bedanke ich mich bei Herrn Halder für die Unterstützung und Betreuung bei der Erstellung der Technikerarbeit.