Diplomarbeit

Stephan Plaschke

Experimentalsystem für drahtlose Batteriesensorik

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Stephan Plaschke

Experimentalsystem für drahtlose Batteriesensorik

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Informationstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Henry Reetmeyer

Abgegeben am 24.07.2008

Stephan Plaschke

Thema der Diplomarbeit

Experimentalsystem für drahtlose Batteriesensorik

Stichworte

Mikroprozessor, Funkübertragung, Datenspeicherung, Datenauswertung

Kurzzusammenfassung

Aktuelle technische Entwicklungen stellen sehr hohe Anforderungen an die Leistungskapazität einer Fahrzeugbatterie. Sicherheitsrelevante Aspekte und die Verfügbarkeit der Fahrzeuge stehen im Vordergrund. Um dies zu gewährleisten ist eine Überwachung der Betriebsspannung unabdingbar. Konventionelle Systeme überwachen die Spannung über die Summe aller Batteriezellen. Durch die Messung der Batteriespannung ist ein Rückschluss auf den Zustand einer einzelnen Zelle so gut wie unmöglich. Die Praxis beweist aber, dass dies häufig der Grund für unerwartete Ausfälle ist.

Der Umfang der vorliegenden Arbeit umfasst die Entwicklung eines zentralen Steuergerätes und einer Software zur Auswertung der gespeicherten Daten.

Stephan Plaschke

Title of the paper

Experimental system for wireless battery sensors

Keywords

Microprocessor, radio transmission, digital data storage, electronic data processing

Abstract

Recent technical developments create high demands on the performance level of a car battery. At the focal point in all considerations are the safety aspects and availability of the vehicle are key considerations. To guarantee both, control of the car battery is inevitable. Conventional systems measure the overall battery voltage, not the voltage of each single battery cell.

With this kind of monitoring inference of one battery cell is all but impossible. In practice the failure of one cell is common cause for an unexpected battery malfunction.

Content of this paper at hand is the development of a controlling device and software to evaluate the collected data.

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg
Department Informations- und Elektrotechnik
Prof. Dr.-Ing. Karl-Ragmar Riemschneider

6. April 2008

Diplomarbeit:
Experimentalsystem für drahtlose Batteriesensorik

Aktuelle technische Entwicklungen stellen immer höhere Anforderungen an Fahrzeugbatterien. Herausfordernd ist die Ausrüstung von Automobilen mit sicherheitskritschen elektrischen Großverbrauchern, wie die elektrische Servolenkung und die elektrische Bremse. Bei elektrisch angetriebenen Spezialfahrzeugen, wie z.B. bei Gabelstaplern, ist die Batterieverfügbarkeit ein leistungsbestimmendes Merkmal.
Drahtlose Sensoren könnten zukünftig dazu beitragen, den Batteriezustand wesentlich besser und zuverlässiger erfassen zu können. Diese Sensoren würden die Spannung an jeder der in Reihe geschalteten Zellen ständig messen und gegebenenfalls die Schwächung auch einer einzelnen Zelle anzeigen können. Eine ungleichmäßige Zellenalterung ist – über die Klemmen der Gesamtbatterie gesehen – bisher schwer beobachtbar, jedoch häufig die Ursache unerwarteter Ausfälle.
Die drahtlose Wirkungsweise der Sensoren hat erhebliche Aufwandsvorteile. Es kann damit auf eine zusätzliche robuste Messverkabelung und Anschlusskontaktierung auf der Batterie verzichtet werden. Eine Potentialtrennung ist systembedingt gegeben.

In der Diplomarbeit soll ein Experimentalsystem entstehen, mit dem das Konzept der drahtlosen Batteriezellenüberwachung näher untersucht werden kann. Hierzu sollen die Entwicklungsarbeiten umfassen:

- Aufbau eines Steuergerätes mit dem Mikrocontroller TI MSP430F169, einem integrierten 433 MHz-Empfänger und Flashspeicher sowie einfacher Anzeige und Bedienung
- Empfangssoftware des Steuergerätes für das digitale Übertragungsprotokoll der drahtlosen Sensoren
- Entwicklung eines Konzeptes und einer Software für die autonom arbeitende Sensordatenspeicherung im Flashspeicher auf dem Steuergerät
- Konzeption und Entwicklung einer Shellsoftware für die Konfiguration, den Dialogbetrieb und die Ausgabe von Messdatenreihen mit Matlab

Die Erprobungen und Vorversuche sollen umfassen:

- Aufbau eines Demonstrators mit mehreren drahtlosen Sensoren auf einer Batterie
- Durchführung und Aufzeichnung von einigen exemplarischen Entlade/Ladezyklen
- Unterstützung von Konfiguration und Auswertung durch PC-Softwaremodule
- Darstellung, Auswertung und Beurteilung der erfassten Messreihen

Die Fachliteratur und kommerzielle Unterlagen sind zielgerichtet zu recherchieren. Dabei ist auch die vorgesehene Fahrzeuganwendung näher zu betrachten. Die Controllersoftware ist in der Programmiersprache C modular und erweiterbar zu entwickeln. Die Funktionsweise und die Softwarestruktur ist gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungen und wesentliche Entwurfsentscheidungen sollen beschrieben werden. Die Erprobungsergebnisse sind als Spannungsverläufe und als zusammenfassende Diagramme anwendungsbezogen darzustellen. Die realisierten Lösungen und Messreihen sind kritisch zu bewerten (z.B. Aussagefähigkeit, Genauigkeit, Zeitauflösung). Ansätze für Verbesserungen und weitere Arbeiten sind aufzuzeigen.

Danksagung

Die vorliegende Diplomarbeit ist im Labor für Informationstechnik an der HAW Hamburg entstanden.

Bedanken möchte ich mich bei Prof. Dr.-Ing. Karl-Ragmar Riemschneider, der es überhaupt erst ermöglichte diese Arbeit zu schreiben, sowie bei Prof. Dr.-Ing. Henry Reetmeyer, der meine Arbeit als Zweitkorrektor bewertet hat.

Weiterhin gilt mein Dank den Mitarbeitern des Labors für Informationstechnik, Herrn Gerhard Wolff und Herrn Jörg Pflüger. Beide standen mir bei Problemen, die während der Diplomarbeit auftauchten, fachkundig zur Seite. Für die Unterstützung bei der Versuchsdurchführung danke ich Herrn Bernd Schröder, Mitarbeiter des Fahrzeuglabors der HAW Hamburg.

Außerdem bedanke ich mich bei Tobias Krannich und Thorsten Eger, die ebenfalls ihre Diplomarbeit im Labor für Informationstechnik an der HAW Hamburg geschrieben haben, und mit denen ein reger Interessenaustausch stattfand.

Des Weiteren bedanke ich mich bei Anne-Kathrin Baum, Christian Ritter und Mario Steinke, die meine Arbeit in Schrift und Form korrigiert haben.

Zuletzt gilt mein besonderer Dank meiner Lebensgefährtin Katrin Tantzen, die mich im Laufe der Arbeit in vielfältiger Weise unterstützt hat.

Stephan Plaschke Hamburg, Juli 2008

Formelzeichen und Abkürzungen

$A10$	10 Bit Analogwert
E^0	Standardelektrodenpotenziale
E^0_{neg}	Standardelektrodenpotenzial an der negativen Elektrode
E^0_{pos}	Standardelektrodenpotenzial an der positiven Elektrode
HSO_4^-	Chemisches Zeichen für Hydrogensulfat
H^+	Chemisches Zeichen für ein Wasserstoffion (Proton)
H_2O	Chemisches Zeichen für Wasser
H_2SO_4	Chemisches Zeichen für Schwefelsäure
Hz	Hertz
MHz	Megahertz
PP	Abkürzung für Polypropylen
Pb	Chemisches Zeichen für Blei
PbO	Chemisches Zeichen für Bleioxid
PbO_2	Chemisches Zeichen für Bleidioxid
$PbSO_4$	Chemisches Zeichen für Bleisulfat
Sb	Chemisches Zeichen für Antimon
U_0	Ruhespannung in Volt
V	Formelzeichen für Volt
λ	Wellenlänge
e^-	Symbol für ein Elektron
kB	Kilobyte
kHz	Kilohertz
$kbps$	Kilobit per second
kg/l	Maßeinheit der Dichte (Kilogramm pro Liter)
m	Tastgrad
u_0	Maximale Amplitude einer Trägerschwingung
u_1	Minimale Amplitude einer Trägerschwingung
ADC	Analog Digital Converter
ASK	Amplitude Shift Keying
BFSK	Binary Frequency Shift Keying
BPSK	Binary Phase Shift Keying
CEPT	European Conference of Postal and Telecommunications Administrations
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
EDA	Electronic Design Automation
ERO	European Radio Office

FET	Flash emulation tool - Programmieradapter
FSK	Frequency Shift Keying
GND	Ground
IO	Input Output
ISM	Industrial, Scientific, and Medical Band
ITU	International Telecommunication Union
ITU-R	ITU Telecommunication Standardization Sector
JTAG	Joint Test Action Group
LSB	Lower Side Band
lsb	Least Significant Bit
OOK	On-Off-Keying
PSK	Phase Shift Keying
RAM	Random Access Memory
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computers
SDK	Software development kit - Entwicklungsumgebung
SOC	State of Charge
SOH	State of Health
TDMA	Time Domain Multiple Access
uC	Microcontroller
UHF	Ultra-High-Frequency
USB	Upper Side Band

Inhaltsverzeichnis

1. Einleitung

Akkumulatoren sind heutzutage wichtiger denn je. Kleinstgeräte wie Mobiltelefone und MP3-Player oder elektrisch betriebende Fahrzeuge wie Gabelstapler, Hybrid- und Kraftfahrzeuge nutzen Akkumulatoren. Auf der einen Seite werden Forschungen betrieben, um neue Akkumulatortypen mit höherer Speicherkapazität und längerer Lebensdauer zu entwickeln. Auf der anderen Seite nimmt für bestehende Systeme die Ladezustandsüberwachung an Bedeutung zu. Besonders bei elektrisch betriebenen Fahrzeugen beschreibt der Ladezustand des Akkumulators die Verfügbarkeit des Fahrzeuges.

Eine andere Neuerung bahnt sich seit Mitte der 1990er Jahre im Automobilbereich an. Die Boardnetzspannung soll von heute üblichen 14 V auf 42 V erhöht werden. Die Gründe liegen nahe: Es wird eine immer größere Anzahl elektrischer Verbraucher im Kraftfahrzeug eingesetzt und einige davon, wie zum Beispiel die elektrische Bremse und die elektrische Lenkung, benötigen mehr Leistung als heutzutage bereitgestellt werden kann. Bei einer Änderung von 14 V auf 42 V würde bei gleicher Leistung ein um den Faktor 3 geringerer Strom benötigt werden. Daraus resultiert ein geringerer Leiterquerschnitt, der eine Gewichtsersparnis von 6 kg bis 12 kg zur Folge hätte. Der geringere Querschnitt würde es möglich machen, Leiterbahnen direkt auf Karrosserieteile anzubringen und die Produktionskosten und den Energiebedarf der Produktion somit zu senken. [16]

Der Nachteil des 42 V Boardnetzes ist die Versorgung. Die mit dem Starter gekoppelte Lichtmaschine ist nicht ausreichend erforscht und die Akumulatortechnik ist noch nicht weit genug vorangetrieben. Fernerhin müssten mehr Zellen verbaut werden, um die benötigten 36 V (Leerlaufspannung) bereitzustellen. Mehr Zellen erhöhen aber das Risiko eines Ausfalls des gesamten Akkumulators, da schon bei dem Ausfall einer einzelnen Zelle der Akkumulator unbrauchbar wird. [16]

Betrachtet man neue und alte Techniken ist es unabdingbar eine intelligente kostengünstige Überwachung bereitzustellen. Jede einzelne Zelle des Akkumulators ist zu überwachen, um frühzeitig einen Ausfall vorhersehen zu können. Dies ist besonders im Automobilbereich sehr wichtig, wenn es zum Einsatz elektrischer Bremsen und Lenkungen kommt, da ein Ausfall des Akkumulators den Ausfall beider kritischen Systeme zur Folge hätte. Bei Flurförderzeugen ist der Ausfall des Antriebs oder der Lenkung nicht ausschlaggebend. Hier zählt die Verfügbarkeit als Berechnungsgrundlage für die Produktionskosten. Ein unerwarteter Ausfall zieht hohe Kosten nach sich. Sind Probleme frühzeitig zu erkennen können Gegenmaßnahmen eingeleitet werden um die Verfügbarkeit wieder herzustellen.

Im Laufe dieser Arbeit soll ein System entwickelt werden, welches den Zustand jeder einzelnen Batteriezelle berechnen kann. Die aufgenommenen Daten sollen hierbei grafisch am PC dargestellt und ausgewertet werden. Weiterhin sind verschiedene Versuche durchzuführen, an denen die Notwendigkeit eines solchen Überwachungssystems erkannt werden können. Die Betrachtung der Ergebnisse soll Aufschluss über die Funktion des Systems

geben und bei Bedarf Mängel aufzeigen. Für die Versuchsdurchführung ist ein geeignetes Testobjekt zu finden. Bei der Entwicklung ist darauf zu achten, dass das gesamte System kostengünstig herzustellen ist. Ebenfalls sollte es auf schon vorhandene Systeme aufsetzbar sein.

1.1. Aktueller Stand der Technik

Viele Firmen haben bereits erkannt, dass eine intelligente Batterieüberwachung die Lebenszeit von Batterien um ein vielfaches verlängern und zudem auch noch Kosten sparen kann. Aktuelle Systeme, wie das *Sentinel* von LEM Holding, übertragen die Daten über ein verdrahtetes Bus System. Hierbei sind Busleitungen von jedem Sensor zum Steuergerät zu verlegen. Diese Verdrahtung birgt recht hohe Kosten. Daher wird in der vorligenden Arbeit mit einer drahtlose Datenübertragung gearbeitet. Die Art der Messung ist vergleichbar mit der in dieser Arbeit angewendeten. Hier wird an jeder Zelle die Spannung, Temperatur und zusätzlich noch die Impedanz gemessen. Daraus läßt sich zuverlässig der SOC und SOH bestimmen. [23]

Andere Geräte wie *Der intelligente Batteriesensor* von Hella KGaA Hueck & Co sind direkt für den Einbau im Kraftfahrzeug entwickelt. Hierbei handelt es sich um eine Batterieklemme mit integrierter Messsensorik. Der Nachteil hierbei ist, dass nicht jede einzelne Zelle überwacht werden kann. Um diesen Nachteil zu kompensieren, ist neben der Spannungs- und Temperaturmessung noch eine Strommessung implementiert. Für die Berechnung des SOC oder SOH birgt es keinen Vorteil. Es können aber defekte Geräte anhand des Ruhestroms erkannt werden. [12]

Im Bereich der Unterhaltungselektronik sind ebenfalls einige Systeme zu finden. Dort reicht das Spektrum von einzelnen Chips, die Ladealgorithmen an das Netzteil senden, bis hin zu Systemen, die die Temperatur und die Spannung kontinuirlich messen, um daraus den SOC und SOH berechnen. Diese Systeme sind in den Batterien integriert. Batterien diesen Typs werden *intelligente Batterien* genannt. Diese Technik hat sich bis jetzt nur in Kleinstbatterien durchgesetzt.

Betrachtet man die derzeit auf dem Markt erhältlichen Geräte, wird deutlich, dass hier noch viel Entwicklungspotenzial vorhanden ist. Geräte, die universal einsetzbar sind, werden benötigt. Zudem sind noch wenige Erkenntnisse über die genau Bestimmung des Ladezustandes der verschiedenen Batterietypen bekannt.

1.2. Bleiakkumulator

Für einen besseren Überblick wird der Aufbau und die Funktion handelsüblicher Bleiakkumulatoren erläutert. Im nachfolgenden Text wird der Begriff Akkumulator oder Bleiakkumulator häufig durch den umgangssprachlichen Ausdruck Batterie ersetzt.

1.2.1. Geschichte des Bleiakkumulators

Die Geschichte der elektrochemischen Zellen beginnt im Jahre 1789 mit einer Beobachtung des italienischen Naturforschers und Professors für Anatomie Luigi Galvani. Dieser entdeckte durch Experimente mit Froschschenkeln eine Kontraktion der Muskeln durch Einfluß statischer Elektrizität. Diese so genannte „tierische Elektrizität“ regte in den neunziger Jahren des 18. Jahrhunderts den Physiker Alessandro Conte di Volta zum Bau der nach ihm benannten Volta-Säule an. Die Volta-Säule ist nach heutigen Sprachgebrauch eine sogenannte Primärzelle, die chemische Reaktion ist nicht oder nur teilweise umkehrbar. [2]

Als Entdecker des Vorläufers der heutigen Akkumulatoren gilt Johann Wilhelm Ritter (1776 bis 1810). Anhand seiner Experimente mit der Volta-Säule und verschiedener Metalldrähte beobachtete er die elektrolytische Wasserzersetzung und fing dabei Wasser- und Sauerstoff getrennt auf. Nach Laden einer Batterie mit der Volta-Säule konnte er sie wieder entladen. Diese von ihm benannten A-Säulen gelten als Vorläufer der heutigen Akkumulatoren oder Sekundärzellen. [2]

Nach seinem Tod geriet seine Erfindung erst einmal in Vergessenheit. Erst im Jahre 1854 wurde sie vom Arzt Wilhelm Josef Sinsteden (1803 bis 1891) zum heutigen Bleiakkumulator weiterentwickelt. Durch die Erfindung von Sinsteden inspiriert entwickelte der Franzose Gaston Planté einen kompakten Bleiakkumulator mit aufgerollten Bleiplatten (Bild 1.1). [2]

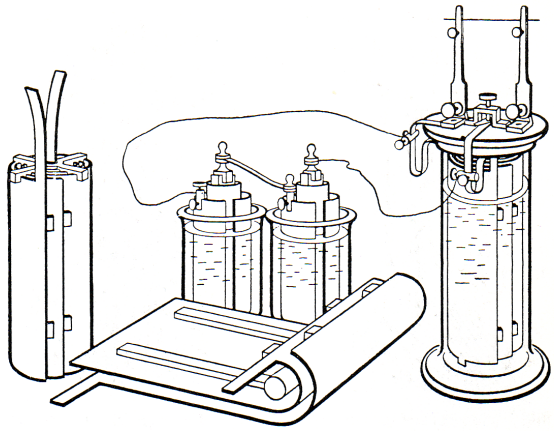


Bild 1.1.: Prinzipaufbau des Planté-Akkumulators [2]

Weitere Entwicklungen folgten, um die Kapazität und das Gewicht des Bleiakkumulators zu verringern. Die erste Verbesserung kam 1881 von Camille Faure, der die Bleiplatten mit Bleiverbindungen beschichtete und als Trennwände Filz verwendete. Ernest Volckmar verbesserte diese Idee, indem er Gitterplatten verwendete, welche mit Bleiverbindungen überzogen waren. Im Prinzip wurden damit zwei Plattentypen erfunden, die auch heute noch Anwendung finden: die Großflächenplatte und die Gitterplatte. [2]

Planté interessierte sich für die Reaktion, die in seinem Akkumulator stattfand und schrieb seine Erfahrungen in seinem Werk „Recherches sur l’Electricité“ nieder. Nachgewiesen wurde die genau Reaktion jedoch erst durch die *Doppelsulfat-Theorie* von J. H.

Gladstone und Alfred Tribe. Diese beschreibt die Reaktionsgleichung im Bleiakkumulator [2, 6]

$$(-)Pb + 2H_2SO_4 + PbO_2(+) \underset{entladen}{\overset{laden}{\rightleftharpoons}} 2PbSO_4 + 2H_2O \tag{1.1}$$

Weitere Nachforschungen folgten, um die Lebensdauer und die Kapazität des Bleiakkumulators zu erhöhen. Seit Anfang des 20. Jahrhunderts hat sich die Energiedichte in etwa vervierfacht, wobei das Gewicht durch den Einsatz von Bleigitterplatten verringert wurde. Bild 1.2 verdeutlicht dies und zeigt die Verbesserung der Energiedichte, gemessen in Wh/kg oder Wh/l, bezogen auf eine dreistündige Entladung. [2]

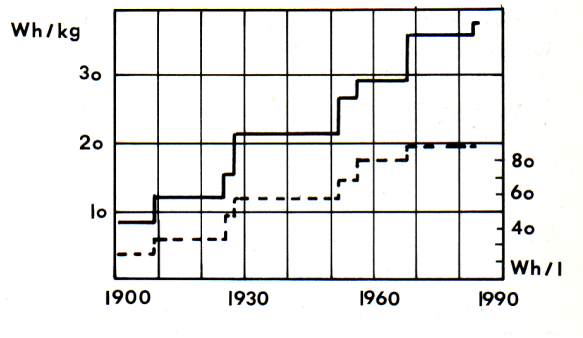


Bild 1.2.: Verbesserung der Energiedichte [2]

Der Bleiakkumulator ist ein gutes Beispiel dafür, dass eine Technik, die schon mehr als 150 Jahre besteht, immer noch weiterentwickelt werden kann. Neue Akkumulatortypen sind bis heute nicht in der Lage, den Bleiakkumulator abzulösen. Um aber den zukünftigen Anforderung gerecht werden zu können, sind weitere Untersuchungen notwendig. Schwerpunkte sind die Lebensdauer und die Zuverlässigkeit eines Bleiakkumulators. Um höchstmögliche Zuverlässigkeit zu gewährleisten, ist eine Überwachung jeder einzelnen Batteriezelle unabdingbar. Diese Thematik wird in der vorliegenden Arbeit behandelt.

1.2.2. Elektrochemische Reaktion

Ein typischer Bleiakkumulator besteht aus mehreren in Reihe geschalteten, elektrischen Zellen, auch bekannt als galvanische Elemente. Allgemein betrachtet bestehen galvanische Elemente aus einem Elektrolyt, einer positiven und einer negativen Elektrode. Der Elektrolyt ist meist flüssig (herkömmliche Zelle) oder pastenartig (dryfit Zelle) und dient als Ionenleiter. Im Bleiakkumulator kommt Schwefelsäure (H_2SO_4) als Elektrolyt zum Einsatz. Um übermäßige Korrosion der Elektroden zu vermeiden, wird die Konzentration der Schwefelsäure auf etwa 30% reduziert. Die positive Elektrode besteht aus Bleioxid (PbO_2) und die negative Elektrode aus Blei (Pb). Dies ist einmalig, da in der Regel ein galvanisches Element aus zwei verschiedenen Metallen besteht. Der Aufbau einer solchen Zelle ist in Bild 1.3 schematisch dargestellt. [17, 26]

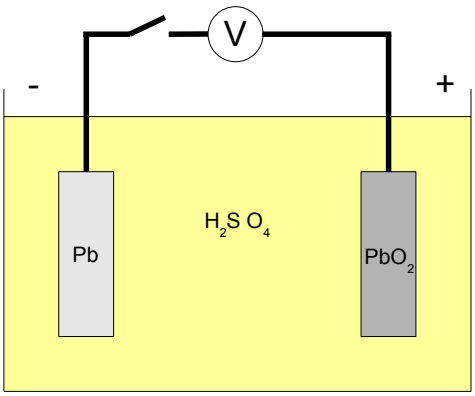


Bild 1.3.: Schematische Darstellung einer galvanischen Zelle

Um die genaue chemische Reaktion zu verstehen, ist es notwendig, den Lade- und Entladevorgang zu betrachten. Beim Entladevorgang (Bild 1.4) werden Elektronen von der Kathode über den Verbraucher an die Anode abgegeben. Die Anode entspricht in diesem Fall dem positiven Pol und die Kathode dem negativen Pol. Anders als in der allgemeinen Elektrochemie sind die Definitionen der Anode und der Kathode abhängig von der Richtung des Stromflusses. Die Anode ist definiert als Elektrode, an der eine Oxidation (Abgabe von Elektronen) stattfindet. Betrachtet man den Elektrolysevorgang aus diesem Blickwinkel, entspricht die Anode dem positiven Pol während des Ladevorgangs. Beim Entladevorgang entspricht die Anode allerdings dem negativen Pol. [10]

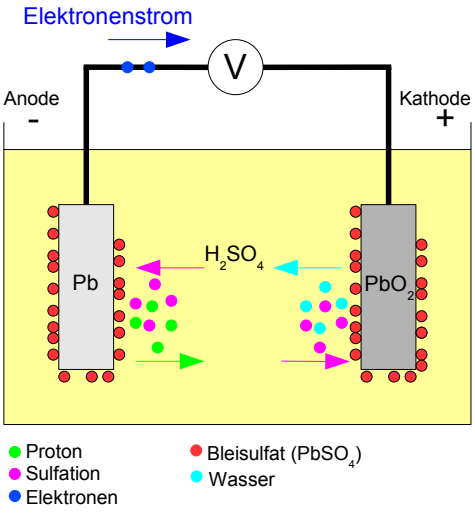


Bild 1.4.: Entladevorgang einer chemischen Zelle

Die chemischen Reaktionen, die dabei stattfinden, können durch folgende Gleichungen

beschrieben werden. Aufnahme von Elektronen an der Kathode (Reduktion), somit ein Übergang vom 4-wertigen zum 2-wertigen Bleiion:

$$Pb^{4+} + 2e^- \rightarrow Pb^{2+}$$

Abgabe von Elektronen an der Anode (Oxidation), somit ein Übergang von Blei zum 2-wertigen Bleiion:

$$Pb \rightarrow Pb^{2+} + 2e^-$$

Die Summengleichung der Reduktion und Oxidation (Redoxreaktion) entspricht der Entladungsgleichung (Redoxgleichung) in einer Zelle:

$$Pb^{4+} + Pb \rightarrow 2Pb^{2+} \tag{1.2}$$

Der Ladevorgang entspricht der Abgabe von Elektronen an der Anode, hier der negative Pol, und der Aufnahme von Elektronen an der Kathode (Bild 1.5). Somit sind die chemischen Vorgänge nach Formel 1.2 umgekehrt und es kann folgende Beziehung als gültig angesehen werden:

$$Pb^{4+} + Pb \underset{entladen}{\overset{laden}{\rightleftharpoons}} 2Pb^{2+} \tag{1.3}$$

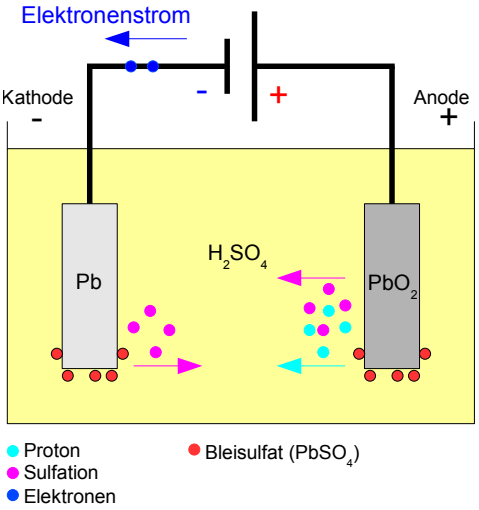


Bild 1.5.: Ladevorgang einer chemischen Zelle

Vergleicht man Formel 1.3 mit Formel 1.1, der *Doppelsulfat-Theorie*, stellt man fest, dass diese nicht übereinstimmen. Es wird deutlich, dass außer den Bleispezies noch ein weiterer Stoff an der Reaktion beteiligt sein muss. Da sich in der Zelle nur noch der Elektrolyt (die Schwefelsäure) befindet, muss dieser maßgeblich an der Reaktion beteiligt sein. Man vervollständigt die Reaktionsgleichungen mit dem Elektrolyt und erhält somit an der positiven Elektrode:

$$PbO_2 + 3H^+ + HSO_4^- + 2e^- \rightleftharpoons PbSO_4 + 2H_2O \tag{1.4}$$

Daraus resultiert die Reaktionsgleichung an der negativen Elektrode:

$$Pb + HSO_4^- \rightleftharpoons PbSO_4 + H^+ + 2e^- \tag{1.5}$$

Die Summe dieser beiden Gleichungen ergibt die wirklichkeitsnahe Reaktionsgleichung:

$$Pb + PbO_2 + 2H^+ + 2HSO_4^- \underset{entladen}{\overset{laden}{\rightleftharpoons}} 2PbSO_4 + 2H_2O \tag{1.6}$$

1.2.3. Selbstentladung

Selbstentladung beschreibt den Vorgang, der in einem Bleiakkumulator abläuft, wenn er nicht an einen Verbraucher oder an ein Ladegerät angeschlossen ist. An den Elektroden laufen (auch im ruhenden Zustand) elektrochemische Reaktionen ab. Diese Reaktionen führen dazu, dass sich der Bleiakkumulator mit der Zeit selbst entlädt. In der Praxis spricht man von monatlich 2 bis 10%, je nach Typ und Alter des Akkumulators.

Diese Reaktionen liegen unter anderem daran, dass Blei auch in Lösung gehen kann, wenn kein Strom durch die Elektroden fließt. Die folgende Gleichung beschreibt den Lösungsvorgang von Bleidioxid in Schwefelsäure:

$$PbO_2 + 2H_2SO_4 \rightarrow 2PbSO_4 + 2H_2O + O_2 \tag{1.7}$$

1.2.4. Ladezustand

Der Ladezustand, im englischen State of Charge (SOC), kann über mehrere mehr oder minder genaue Techniken ermittelt werden. Die genaueste ist die Messung der Konzentration des Elektrolyts. Durch Bildung von Wasser beim Entladungsvorgang (Formel 1.7) sinkt die Konzentration der Schwefelsäure. Somit ist die höchste Säurenkonzentration bei voller Ladung erreicht. Die Säurekonzentration schwankt zwischen 37 und 15 Massenprozent beziehungsweise die Elektrolytdichte zwischen 1,28 kg/l und 1,10 kg/l. Allerdings ist die Messung der Elektrolytdichte sehr kostenintensiv und daher für konventionelle Systeme kaum sinnvoll einsetzbar. [2, 11]

Eine andere Möglichkeit ist das Messen der Zellenspannung in Ruhe und unter Belastung. Parallel dazu ist die Temperatur zu ermitteln. Eine Auswertung beider Messwerte läßt einen Rückschluss auf den Ladezustand zu. Diese Technik ist einfach und kostengünstig umzusetzen und wird in vorliegender Arbeit verwendet.

Errechnet werden kann die Ruhespannung (die Potentialdifferenz) mit Hilfe der Elektrochemischen Spannungsreihe. Betrachtet werden dazu die Standardelektrodenpotenziale (E^0) von Oxidation und Reduktion während des Ladevorgangs. „Unter dem Standardpotential eines Redoxpaares (sind 2 Stoffe die miteinander unterschiedlich reagieren) versteht man die unter Standardbedingung (z.B.: Luftdichte, Standardtemperatur und Standarddruck) messbare elektrische Spannung zwischen einer Wasserstoffelektrode (wird allgemein als Elektrode bezeichnet) und der Halbzelle (setzt sich zusammen aus einem Metall und

seiner Salzlösung) jenes Redoxpaares.“ [7] Standardpotenziale sind in diversen Tabellen ablesbar, z.B. [15]. [2]

Entladungsvorgang an der positiven Elektrode nach Formel 1.4:

$$PbO_2 + 3H^+ + HSO_4^- + 2e^- \rightarrow PbSO_4 + 2H_2O \mid E^0_{pos} = +1,68\ V$$

Entladungsvorgang an der negativen Elektrode nach Formel 1.5:

$$Pb + HSO_4^- \rightarrow PbSO_4 + H^+ + 2e^- \mid E^0_{neg} = -0,36\ V$$

Die Potentialdifferenz der beiden Entladungsvorgänge entspricht dem Standardpotenzial E^0:

$$E^0 = E^0_{pos} - E^0_{neg} = 1,68V - (-0,36) = 2,04\ V \tag{1.8}$$

Wird bei Leerlauf eine Spannung von $U_0 = 2,04\,V$ ermittelt, entspricht der Ladezustand 100%. Eine Leerlaufspannung von 1,7 V entspricht einer vollständig entleerten Batterie.

Neben dem Ladezustand (SOC) ist noch der Zustand jeder Batteriezelle zu berücksichtigen, im englischen State of Health (SOH). Dieser kann als prozentuale Differenz der aktuellen zur Nennkapazität oder als Restkapazität definiert werden. In Bild 1.6 ist dieser Bezug grafisch dargestellt.

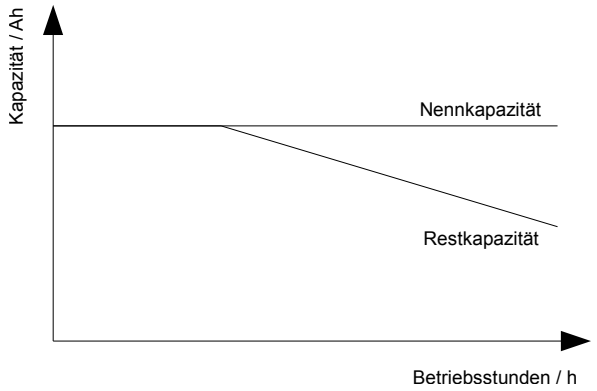


Bild 1.6.: Kapazitätsverlust eines Akkumulators

Im Laufe eines Batterielebens kann nicht verhindert werden, dass der SOH jeder Zelle unterschiedlich abnimmt. Bei einer Antriebsbatterie zum Beispiel leiden die inneren Zellen stärker, da sie sich stärker erhitzen. Wird aber darauf keine Rücksicht genommen und der Ladeprozess wie bisher fortgesetzt, wird die Lebenszeit mehr und mehr verkürzt. Ausfälle der Zellen mit schwachen SOH sind vorprogrammiert. Der Grund liegt auf der Hand. Die geschwächten Zellen sind schneller geladen und werden bei einem normalen Ladevorgang überladen. Bei einer Überladung tritt Gitterkorrosion (Kapitel 1.2.6) auf. Diese zersetzt die Gitterplatten und zerstört die Batterie. Um dies zu verhindern, können die stärkeren auf das Niveau der schwächeren Zellen gebracht werden. Dies kann durch direkes Entladen der stärkeren Zellen erfolgen. Wird daraufhin die Batterie wieder geladen, werden die schwächeren Zellen nicht überladen.

1.2.5. Mechanischer Aufbau

Die folgenden Beschreibungen beziehen sich auf eine handelsübliche 12 V Starterbatterie der Firma VARTA. Die vorliegende Arbeit behandelt einen etwas anderen Akkumulatortyp, den sogenannten Röhrchenplatten-Akkumulator. In der Funktion gleichen die beiden Typen sich. Der einzige Unterschied ist der Aufbau der positiven Elektroden. Dieser Unterschied wird am Ende des Kapitels verdeutlicht. Den Aufbau einer VARTA Blockdeckelbatterie der ULTRA dynamic Serie stellt Bild 1.7 dar.

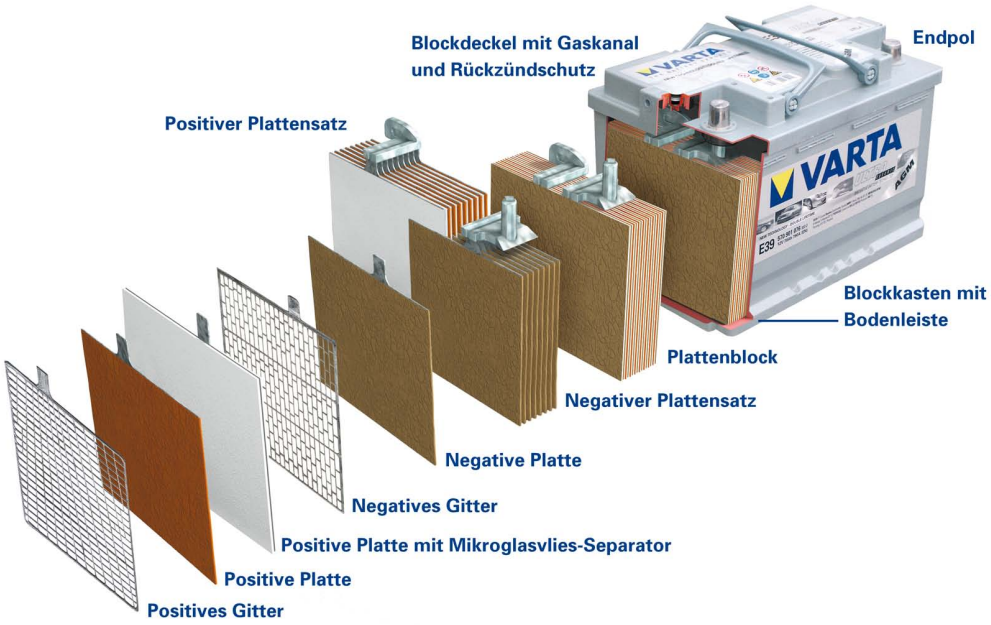


Bild 1.7.: Querschnitt durch einen Bleiakkumulator [9]

Starterbatterien enthalten sechs Zellen, die nebeneinander angeordnet sind. Jede Zelle besteht aus mehreren positiven und negativen Platten, die miteinander verbunden sind. Die Platten bestehen nicht aus einem festen Körper. Dies war in den Anfängen der Fall, stellte sich aber schnell als Nachteil heraus. Heutzutage werden Gitter verwendet, die mit einer aktiven Masse beschichtet werden. Die Hauptbestandteile der Beschichtung, einer pastenartigen Masse, sind Bleioxid (PbO), Schwefelsäure (H_2SO_4) und Wasser (H_2O). Die Gitter bestehen aus einer Blei-Antimon-Legierung und werden in verschiedenen Verfahren hergestellt. Das positive Gitter wird durch Schmelzung und Gießen, das negative Gitter aus Streckmetall gewonnen. Die Beschichtungen wie auch die Gitter sind verschieden. Der Hauptbestandteil der positiven Platte ist Bleidioxid (PbO_2). Die Beschichtung der negativen Platte besteht hauptsächlich aus schwammigem Blei. [2, 9]

Um die negative Platte von der positiven zu trennen, werden Seperatoren eingesetzt. Diese bestehen aus einem säure- und temperaturbeständigem Mikroglasvlies und sind zu

„Taschen“ geformt. Diese „Taschen“ schützen vor internen Kurzschlüssen und umhüllen die positiven Platten. Eine positive und eine negative Platte werden zu einem Plattensatz geschaltet. Mehrere Plattensätze bilden einen Plattenblock, in dem jede negative und jede positive Platte untereinander verbunden ist. Die Größe und die Anzahl der Plattensätze sind für die Kapazität und den Innenwiderstand einer Batterie ausschlaggebend. Je größer die Platten, desto geringer ist der Innenwiderstand. Je mehr Plattensätze, desto höher ist die Kapazität. [9]

Die sechs Zellen sind entgegengesetzt im Blockkasten angeordnet. Dabei wird der negative Pol einer Zelle mit dem positiven Pol der nächsten Zelle verbunden. Durch diese Reihenschaltung entsteht eine Gesamtspannung von 12 V. Verschlossen wird der Blockkasten mit einem Block- oder Monodeckel. Die allgebräuchliche Variante ist der Monodeckel, der in Bild 1.7 zu sehen ist. Hier liegen die Pole der Batterie frei. Beim Blockdeckel sind die Pole in eine Mulde eingebettet. Das verwendete Material für Kasten und Deckel ist schlagfestes Polypropylen. Dieser Kunststoff verleiht der Batterie die Festigkeit, bei einem Auffahrunfall einer Verzögerung des 50-fachen der Erdbeschleunigung standzuhalten. Die Verbindung von Kasten und Deckel wird durch Spiegelschweißen hergestellt, wobei beide Teile erhitzt und längere Zeit aufeinander gepresst werden. [2, 9]

Um die im Blockkasten beim Ladevorgang enstehenden Gase abzuführen, wurden bei älteren Batterien Verschlussstopfen im Deckel mit kleinen Löchern versehen. Bei neueren, wartungsfreien Batterien ist im Deckel ein Gaskanal enthalten, der die Gase über eine kleine Öffnung entweichen läßt. [2, 9] Je nach Anwendung gibt es verschiedene Ausführungen:

- Kippsichere Stopfen (Flugzeug-, Bootsbatterien) [2]
- Katalysator- oder Rekombinationsstopfen (das Gasgemisch aus Wasser und Sauerstoff wird katalytisch zu Wasser rekombiniert, das der Zelle wieder zugeführt wird) [2]
- Flammenschutzstopfen (sie sind mit einem System ausgerüstet, das ein Zünden des Knallgasgemisches von außen verhindert) [2]
- Ventilstopfen (z.B. für tauch- und watfähige Fahrzeuge: der Stopfen schließt bei Überflutung). [2]

Zu Beginn dieses Kapitels wurde angesprochen, dass in dieser Arbeit mit einem anderen Batterietyp gearbeitet wird, der sogenannten Antriebsbatterie. Der wichtigste Unterschied zu einer Starterbatterie besteht darin, dass jede Zelle einzeln aufgebaut ist. Dies hat den Vorteil, dass bei einem Defekt einer einzelnen Zelle diese einfach ausgetauscht werden kann und somit die Funktion des gesamten Akkumulators wieder hergestellt ist. Intern gibt es ebenfalls einen wesentlichen Unterschied. Die positive Gitterplatte ist durch eine Röhrchenplatte ersetzt (Bild 1.8 Punkt 6). Dies hat den Vorteil, dass die Batterie selbst häufige und langanhaltende Tiefentladezyklen übersteht. In der Praxis sind dies 1000-2000 Tiefentladezyklen. [26]

Betrachtet man den Aufbau der Platten etwas genauer (Bild 1.9), ist der Unterschied klar deutlich erkennbar. Anstatt eines Gitters ist das Gerüst aus mehreren Röhrchen

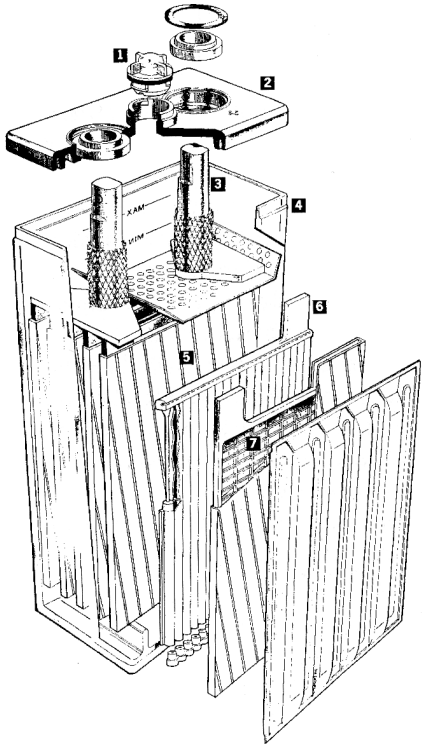


Bild 1.8.: Aufbau einer Antriebsbatterie [26]

gefertigt. Die Röhrchen bestehen aus einem Gitterstab (lead-alloy-spine), der, wie auch bei der Starterbatterie, aus einer Pb-Sb-Legierung gefertigt ist. Jeder Gitterstab ist von einer aktiven Masse (positive active-material) überzogen. Als Schutz umhüllt den Gitterstab ein säurefestes, elektrolytdurchlässiges Röhrchen. In den Anfängen wurde das Röhrchen aus Hartgummi gefertigt, welches diesem Plattentyp den Namen Panzerplatte verlieh. [2, 26]

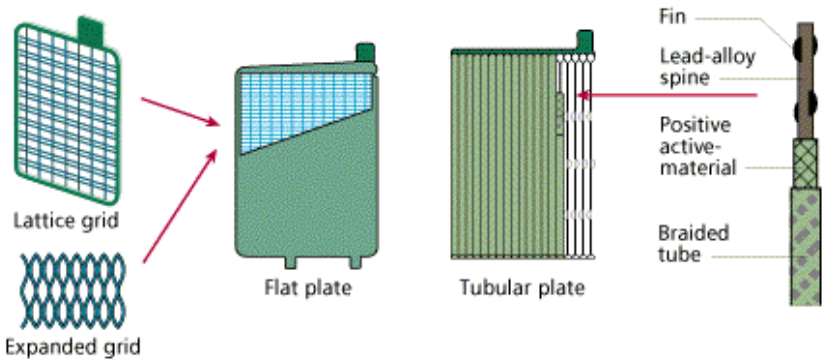


Bild 1.9.: Gitter- und Röhrchenplatte einer Antriebsbatterie [5]

1.2.6. Alterung

Die Alterung von Bleiakkumulatoren macht sich hauptsächlich durch eine geringere entnehmbare Kapazität bemerkbar. Ist diese unter 80% der Nennkapazität gesunken, so ist nach DIN 43539/Teil 4 ein Ende der Lebensdauer erreicht. Eine Alterungserscheinung ist unvermeidbar, durch sachgerechte Behandlung können jedoch viele Erscheinungen gemindert werden. [19]

Als Beispiel hierfür dient die Starterbatterie im Kraftfahrzeug. Diese muss in etwa alle fünf Jahre, bei schlechter Behandlung schon früher, ersetzt werden. Woran genau liegt das? Die Elektroden in einer Starterbatterie bestehen aus vielen Platten, die jeweils eine sehr große Fläche besitzen. Daraus resultiert ein extrem niedriger Innenwiderstand der Starterbatterie. Dieser niedrige Innenwiderstand erlaubt es, einen sehr hohen Strom zu entnehmen, wie er zum Beispiel für den Startvorgang eines Kraftfahrzeuges nötig ist. Doch gerade die dünnen Bleiplatten bergen erhöhtes Risiko: Sulfatierung, Gitterkorrosion und Abschlammung stellen dabei die größten Probleme dar.

Sulfatierung bezeichnet den einen Vorgang bei dem vermehrt Bleisulfat ($PbSO_4$) gebildet wird. Dies geschieht bei längeren Entladungen unterhalb der Entladeschlussspannung, der sogenannten Tiefentladung. Die Entladeschlussspannung bei einem Bleiakkumulator liegt bei 1,7 V. Bleibt dieser Zustand länger erhalten, setzt sich vermehrt Sulfat an den Platten fest. Die daraus resultierende Verbreiterung der Platten kann die Seperatoren sprengen, was zu einem Kurzschluss innerhalb einer Zelle führen kann (Bild 1.10). Diese permanente Sulfatierung führt dazu, dass sich Bleisulfatkristalle zusammenschließen. Ist eine kritische Größe der Kristalle erreicht, verwandelt sich das Bleisulfat in einen elektrischen Nichtleiter. Dieser ist dann unwiderbringlich für die Lade-/Entladereaktion verloren. Zudem vergrößert sich dadurch der Innenwiderstand und der Startstrom verringert sich. [19]

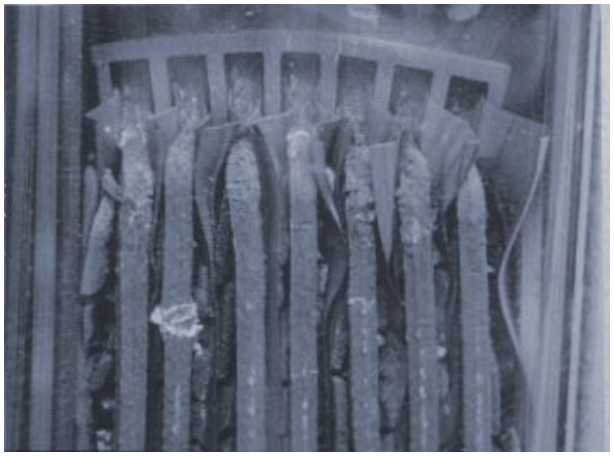


Bild 1.10.: Durch Sulfatierung und Korrosion gesprengte Separatoren [19]

Gitterkorrosion bezieht sich auf die Stoffe, die nicht zu Bleidioxid umgewandelt werden können. Dabei handelt es sich um Legierungsbestandteile wie Antimon, Zinn, Kupfer, Arsen, Silber und Calcium, die in den Gitterplattenträgern vorkommen. Diese

gelangen durch Diffusion über den Elektrolyt zur negativen Elektrode und werden dort als Metalle abgeschieden. Somit wird der Gitterträger in sich langsam abgebaut bis er zerfällt (Bild 1.11). Hervorrgerufen wird Gitterkorrosion durch längeres Überladen des Bleiakkumulators. Beim Überladen steigt die Temperatur überdurchschnittlich an, dies beschleunigt noch die Gitterkorrosion. [19]

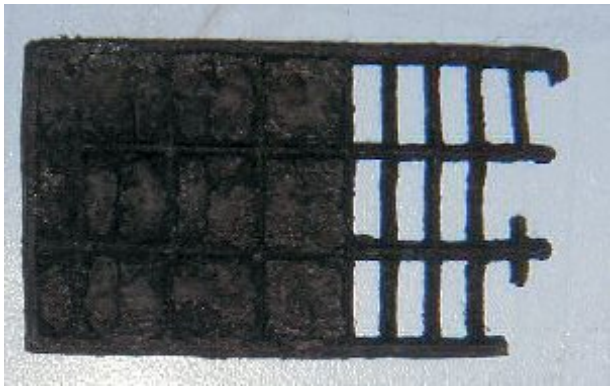


Bild 1.11.: Durch Korrosion zerstörte Gitterplatte [19]

Abschlammung im Akkumulator ist ein weiterer wichtiger Grund für den Kapazitätsverlust eines Bleiakkumulators. Abschlammung wird auch Shedding genannt. Durch den Lade- und Entladevorgang löst sich das Schwammblei an den negative Platten und das Bleidioxid an den positiven Platten aus dem Gefüge. Dieser Vorgang ist nicht umkehrbar und hat den direkten Kapazitätsverlust zur Folge. Abhilfe schaffen dabei Gel-Batterien. Gel-Batterien sind aber im Kfz-Bereich unüblich und finden eher Gebrauch im Hobbybereich. [19]

Die aufgeführten Phänomene stellen die häufigsten Gründe für eine frühe Alterung von Akkumulatoren dar. Als Grundlage für die vorliegende Arbeit ist besonders die Gitterkorrosion und die Sulfatierung zu nennen. Diese „hausgemachten“ Probleme sind durch eine Zellspannungsüberwachung sofort zu erkennen und geeignete Gegenmaßnahmen können eingeleitet werden.

1.3. Funkdatenübertragung

Funkdatenübertragungen und **R**adio **F**requency **ID**entification sind heutzutage wirtschaftlich unverzichtbar. Ende der 1960er Jahre revolutionierte eine RFID-Lösungen den Markt, die sogenannte SICARID. Eingesetzt wurden diese Identifikationsmarken in Großlackierereien zur Markierung von Karosserieteilen. Es wurde schnell erkannt, was für ein potential RFID besitzt, zum Beispiel bei Warensicherungssystemen, Tierkennzeichnungen und Zugangskontrollen. Die Funkdatenübertragung geriet vorerst in den Hintergrund. Es gab Industrielle oder vom Militär genutzte Funk-Anwendungen, kommerziell vertriebene Systeme aber nicht. Dies änderte sich grundlegend mit voranschreiten der Computertechnik.

In den 1980er Jahren kamen Systeme auf den Markt, die einen Datenaustausch von Computer zu Computer über CB-Funk erlaubten. Mit Einführung der mobilen Kommunikation kam der Durchbruch. Funkdatenübertragungen waren allgegenwärtig. Um weiterhin Störungsfreiheit gewährleisten zu können wurden Verordnungen erlassen, die die Benutzung der verschiedenen Frequenzbereiche regeln. Neben kommerziell vertriebenen Frequenzbereichen (Mobilfunk) wurden auch kostenfreie Frequenzbereiche geschaffen. Dies gewährleistet, dass auch kleine Anwendungen Funkdatenübertragungen nutzen können. Diese freien Frequenzbereiche nennen sich ISM-Bereiche.

1.3.1. Frequenzbereiche

ISM steht für **I**ndustrial, **S**cientific and **M**edical Band und wird von der **I**nternational **T**elecommunication **U**nion (ITU), Sektor **R**adiocommunication (ITU-R) in den Regelungen 5.138 und 5.150 weitestgehend festgelegt.

Tabelle 1.1 zeigt alle ausgewiesenen ISM-Bereiche und einige typische Anwendungen. [3]

Frequenzbereich		**Typische Anwendung**
6,765 MHz	6,795 MHz	
13,553 MHz	13,567 MHz	Funketiketten, RFID
26,957 MHz	27,283 MHz	Modelbau-Fernsteuerung
40,66 MHz	40,70 MHz	Modelbau-Fernsteuerung
433,05 MHz	434,79 MHz	Babyphone, Funk-Schalter, RFID
868 MHz	870 MHz	Zulassungsfreie Handfunkgeräte (Europa), RFID
902 MHz	928 MHz	Zulassungsfreie Handfunkgeräte (USA)
2,4 GHz	2,5 GHz	Mikrowellenherde, Bluetooth, ZigBee, WLAN
5,725 GHz	5,875 GHz	WLAN
24 GHz	24,25 GHz	Einparkhilfe beim Pkw
61 GHz	61,5 GHz	Radar-Bewegungsmelder
122 GHz	123 GHz	
244 GHz	246 GHz	

Tabelle 1.1.: Ausgewiesene Frequenzbereiche durch die ITU-R [4]

Jeder Frequenzbereich bietet Vor- und Nachteile. Langwellen beispielsweise haben eine sehr hohe Eindringtiefe in nichtmetallische Stoffe. Somit ist es möglich, Transponder ins Fettgewebe von Tieren zu injizieren, um diese zu markieren. Die Datenrate ist bei diesen Transpondern allerdings sehr gering. Bei der Frequenzauswahl eines RFID- oder Funksystems sind daher die Eigenheiten der zur Verfügung stehenden Frequenzbereiche zu berücksichtigen.

Die ausgewiesenen Frequenzbereiche unterliegen festgelegten Regularien. Diese werden in Europa durch die **E**uropeane **C**onference of **P**ostal and **T**elecommunications Administrations (CEPT) zugeteilt. Die CEPT/ERC REC 70-03 definiert 13 verschiedene Anwendungen (Annexe) von *Short Range Devices* (Tabelle 1.2).

Für jede Anwendung ist der Frequenzbereich, der Leistungspegel, das Kanalraster und die Dauer (duty-cycle) der Aussendung definiert. RFID-Anwendungen im Langwellen-

Annex	Anwendung
Annex 1	Non-specific short range devices
Annex 2	Devices for detecting avalanche victims
Annex 3	Wideband data transmission systems
Annex 4	Railway applications
Annex 5	Road transport & traffic telematics
Annex 6	Equipment for detecting movement and alert
Annex 7	Alarms
Annex 8	Mode control
Annex 9	Inductive application
Annex 10	Radio microphones
Annex 11	RFID
Annex 12	Wireless application in healthcare
Annex 13	Wireless audio application

Tabelle 1.2.: Anwendungen aus der CEPT/ERC REC 70-03 [14]

bereich fallen unter Annex 9, da es sich um eine induktive Funkanlage handelt. Zulassungsfreie Handfunkgeräte (Tabelle 1.1) entsprechen Annex 11. Alle anderen häuslichen Anwendungen sind in Annex 1 definiert.

Die im Laufe der Arbeit entwickelte Anwendung kann keiner der schon definierten Annexen wahrheitsgetreu zugeordnet werden. Da es sich um ein Funkdatenübertragungssystem handelt, wird es derzeit Annex 1 zugeordnet. Sollten Systeme dieser Art weitreichend eingesetzt werden, ist es nach Meinung des Autors notwendig, eine eigene Kategorie (Annex) dafür zu verfassen.

Die vollständige Regulierung kann auf der Homepage des **E**uropean **R**adio **O**ffice (ERO) eingesehen werden: http://www.ero.dk/. [1, 14]

1.3.2. Digitale Modulation

Unter Modulation versteht man die Beeinflussung eines Signalparameters einer elektromagnetischen Welle durch ein Modulationssignal. Aus der Analogtechnik sind drei Modulationsarten bekannt. Je nachdem welcher Parameter geändert wird spricht man von Phasen-, Amplituden- oder Frequenzmodulation. Alle anderen Modulationsverfahren leiten sich aus diesen drei Modulationsarten ab. Auch die bei RFID-Systemen üblichen digitalen Modulationsverfahren **P**hase-, **F**requency- oder **A**mplitude **S**hift **K**eying sind davon abgeleitet.

Bei jeder Modulationsart spricht man von einem Träger und zwei Seitenbändern. Bei den Seitenbändern unterscheidet man zwischen dem *Unteren (LSB)* und *Oberen (USB) Seitenband*. In beiden Seitenbändern ist die Nutzinformation gespeichert und die Verschiebung auf der Frequenzachse entspricht der Frequenz des Nutzsignals.

Anders als bei der analogen Modulation wird bei der digitalen Modulation in der Regel nur zwischen zwei Zuständen umgeschaltet. Bei der PSK-Modulation werden die binären Zustände in einer Phasenverschiebung codiert, bei der FSK-Modulation in verschiedene

Frequenzen des Trägersignals und bei der ASK-Modulation in verschiedene Amplituden der Trägerschwingung.

Amplitude Shift Keying

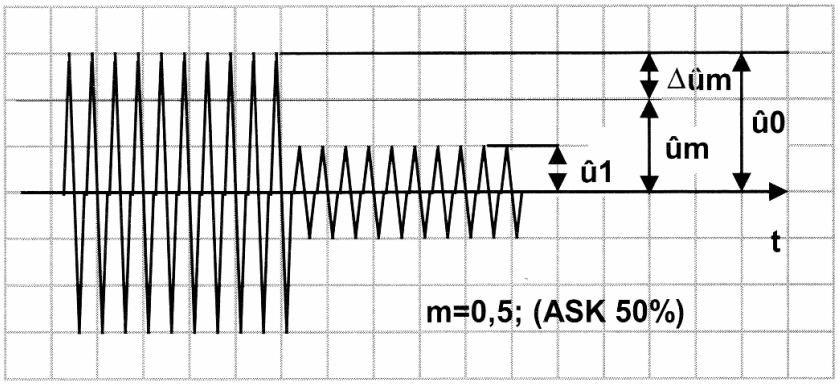


Bild 1.12.: ASK-Modulation [14]

Bild 1.12 zeigt ein moduliertes Trägersignal. Das verwendete Modulationsverfahren ist die ASK-Modulation. Hier stellen die binären Zustände 0 und 1 die Spannungen u_1 und u_0 dar. Je nach Tastgrad m, liegt u_1 zwischen 0 und u_0. Der Tastgrad beschreibt hier nicht die Länge eines Tastimpulses sondern das Verhältniss der Amplitudenänderung. [14]

Um den Tastgrad zu bestimmen wird der Mittelwert $\widehat{u}_m$ berechnet. [14]

$$\widehat{u}_m = \frac{\widehat{u}_0 + \widehat{u}_1}{2} \tag{1.9}$$

Das Verhältnis der Amplitudenänderung $\widehat{u}_0 - \widehat{u}_m$ zum Mittelwert $\widehat{u}_m$ [14]

$$m = \frac{\Delta\widehat{u}_m}{\widehat{u}_m} = \frac{\widehat{u}_0 - \widehat{u}_1}{\widehat{u}_0 + \widehat{u}_1} \tag{1.10}$$

Setzt man die Werte aus Bild 1.12 in Formel 1.9 und Formel 1.10 ein, erhält man als Ergebnis:

$$m = \frac{\widehat{u}_0 - \widehat{u}_1}{\widehat{u}_0 + \widehat{u}_1} = \frac{1 - 1/3}{1 + 1/3} = \frac{2/3}{4/3} = 0,5$$

Die Berechnung ist hier mit den Scheitelwerten durchgeführt worden. In anderer Fachliteratur werden die Berechnungen mit den Effektivwerten durchgeführt. Die Relationen sind bei beiden Berechnungen die selben, es handelt sich nur um eine andere Art der Darstellung.

Die am häufigsten eingesetzte Form der ASK-Modulation ist das **O**n-**O**ff-**K**eying. Hier wird der binäre Zustand direkt umgesetzt. Eine 0 entspricht somit einer 0 der Amplitude und eine 1 dem Wert u_0. Werden diese Werte in Formel 1.10 eingesetzt, ergibt sich wie erwartet $m = 1$.

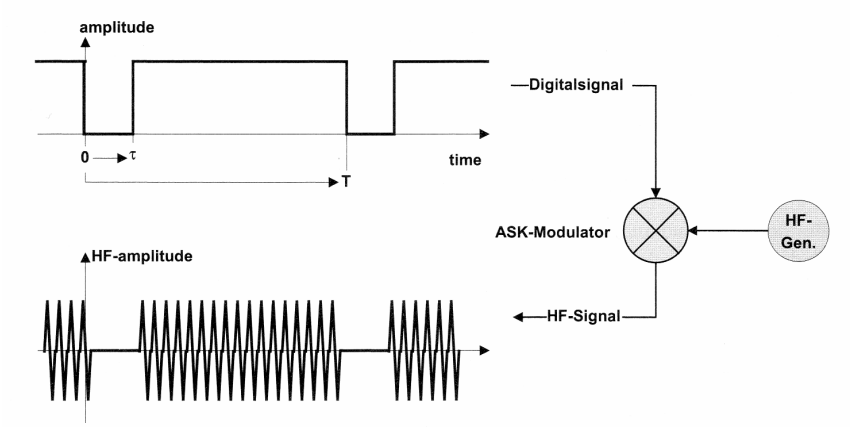


Bild 1.13.: Darstellung einer OOK-Modulation [14]

Mathematisch betrachtet entspricht die ASK einer Multiplikation des binären Datensignals mit dem Trägersignal. Bild 1.14 verdeutlicht noch einmal den Signalverlauf des binären Datensignals und des modulierten Trägers.

Frequency Shift Keying

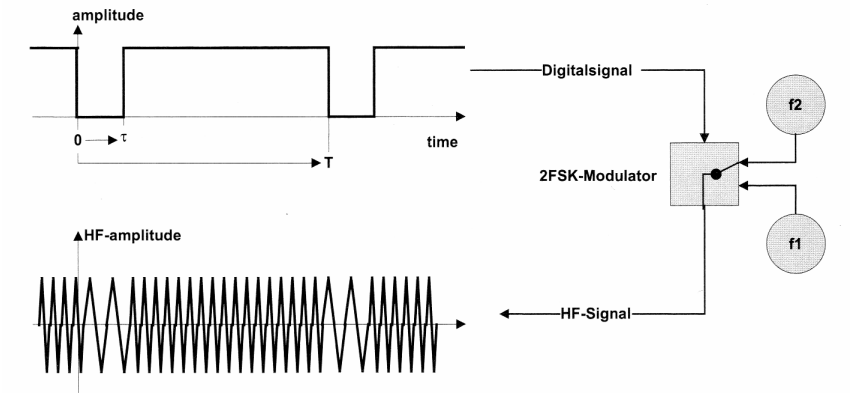


Bild 1.14.: Darstellung einer BFSK-Modulation [14]

Bei der FSK wird zwischen unterschiedlichen Frequenzen umgeschaltet. Die einfachste Form ist die 2-FSK oder BFSK. Hier wird zwischen der Frequenz f_1 und f_2 umgeschaltet. Das binäre Datensignal gibt den Schaltzustand an. Eine 0 entspricht somit der Frequenz f_1 und eine 1 der Frequenz f_2 (Bild 1.14). Als Trägerfrequenz wird der Mittelwert der beiden Frequenzen f_1 und f_2 definiert. Diese Form der FSK ist die einzige Form in der die Bitrate gleich der Symbolrate ist. Werden mehrere Frequenzen verwendet, spricht man von einer M-FSK. Hier ist die Bitrate nicht gleich der Symbolrate. [14]

Phase Shift Keying

Bei der PSK wird zwischen unterschiedlichen Phasenlagen umgeschaltet. Die einfachste Form ist die 2-PSK oder BPSK. Hier wird zwischen zwei Phasenlagen umgeschaltet. Typischerweise entspricht die Phasenlage 0° einer binären 0 und die Phasenlage 180° einer binären 1 (Bild 1.15). Mathematisch gesehen entspricht die BPSK einer Multiplizierung des Trägersignals mit „1“ oder „-1“. [14]

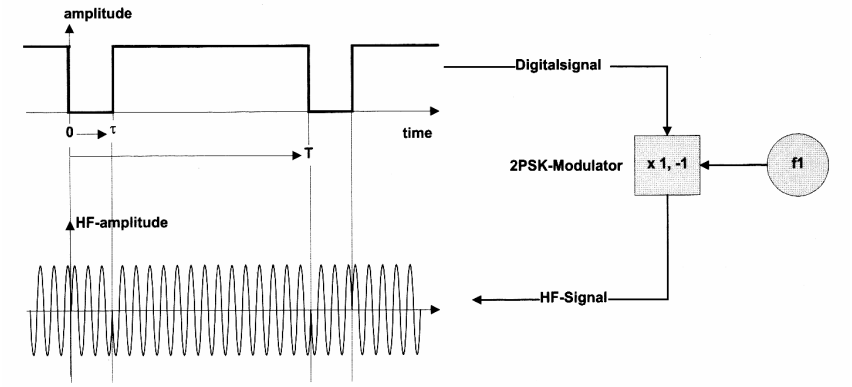


Bild 1.15.: Darstellung einer BPSK-Modulation [14]

1.3.3. Manchestercodierung

Bei einer Funkübertragung ist es nicht möglich, die Rohdaten über den Äther zu transportieren. Genau genommen wäre es zwar möglich, der Empfänger hätte dann aber keine Möglichkeit, bei einer gleichen Bitfolge (z.B. 1111 0000) Anfang und Ende eines einzelnen Bits zu erkennen. Die Begründung liegt nahe. Funkübertragungen sind auf eine Leitung beschränkt, die Luft. Über diese „Leitung“ werden alle Daten gesendet. Somit steht kein seperater Takt für die Übertragung zur Verfügung. Da ein Takt aber benötigt wird, um bei einer seriellen Datenübertragung Anfang und Ende eines Bits zu erkennen, müssen die Rohdaten codiert werden.

Als Codierungsverfahren stehen mehrere Arten zur Verfügung. Je nach Anwendung ist ein geeignetes Verfahren auszuwählen. Bei passiven Transpondern ist besonders darauf zu achten, dass die Energieversorgung gewährleistet ist. Die einfachste und am häufigsten verwendete Codierung ist die Manchestercodierung.

Bei der Manchestercodierung handelt es sich um ein recht einfaches Codierungsverfahren. Um bei der Übertragung eine gleichbleibende Bitfolge zu unterbinden, müssen die Daten codiert werden. Die Codierung beschränkt sich auf das Zerspalten eines Bits in zwei Teilbits (Tabelle 1.3).

Ein HIGH-Pegel entspricht nach der Codierung einem HIGH-Pegel auf den ein LOW-Pegel folgt. Bei einem LOW-Pegel entspricht es einem LOW-Pegel auf den ein HIGH-Pegel folgt. Wendet man dieses Verfahren auf ein Datenwort an, so kann höchstens zweimal der gleiche Pegel nacheinander auftreten. Anhand eines Beispiels kann dies verdeutlicht werden.

Bitweise Darstellung	Manchestercodierung
0	01
1	10

Tabelle 1.3.: Manchestercodierung

Beispiel: Es soll ein Datenwort mit dem Bezeichner X übertragen werden. X soll 255 betragen. In bitweiser Darstellung entspricht das: $X_{bit} = 1011100$. Ohne eine Codierung kann dieses Datenwort nicht empfangen werden, da die Folge von 1 Bits und 0 Bits im Empfänger nicht erkannt wird. Wird die Bitfolge in Manchestercodierung gebildet, ist es sehr einfach jedes Bit zu erkennen, da bei 50% eines Bits ein Wechsel des logischen Zustands stattfindet. In Manchestercodierung entspricht das Datenwort $X_{Man} = 10011010100101$. Bild 1.16 zeigt die jeweiligen Bitfolgen.

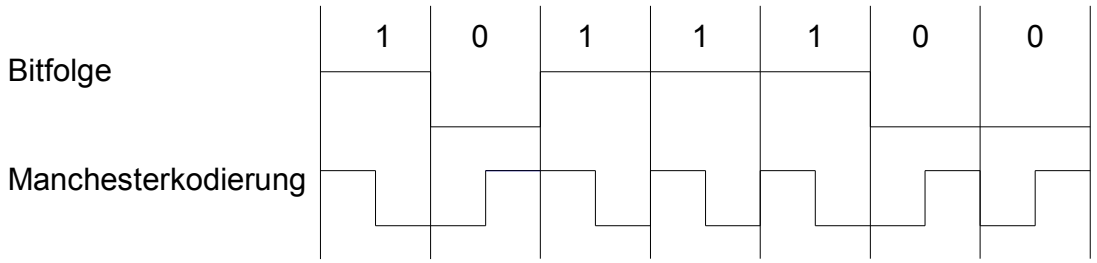


Bild 1.16.: Darstellung einer uncodierten Bitfolge und der in Manchestercodierung codierten Bitfolge

Der einzige nennenswerte Nachteil der Manchestercodierung ist die Verdopplung der Daten und eine daraus resultierende Halbierung der Datenrate. Diesen Nachteil besitzen allerdings alle Codierungsverfahren.

1.3.4. Übertragungsprotokoll

Als Übertragungsprotokoll bezeichnet man eine Vereinbarung, die den Austausch von Daten in einem Netz regelt. Ein Netz beschreibt die Verbindung mit der die kommunizierenden Computer oder Prozesse verbunden sind. In komplexeren Netzen arbeiten verschieden Protokolle zusammen. Um die daraus resultierende Komplexität beherrschen zu können, werden diese Protokolle in verschiedene Schichten eingestuft. Protokolle höherer Schichten greifen auf die Daten niedrigerer Schichten zurück. Diesen Zusammenschluss mehrerer Protokolle nennt man einen Protokollstapel. Betrachtet man das TCP/IP Protokoll erkennt man etwa 500 verschiedenen Protokolle. [14]

Die Hardware erlaubt nur einen unidirektionalen Datenaustausch. Daher ist es nicht nötig, verschiedene Protokolle zu vereinbaren. Um zu vermeiden, dass Daten gleichzeitig

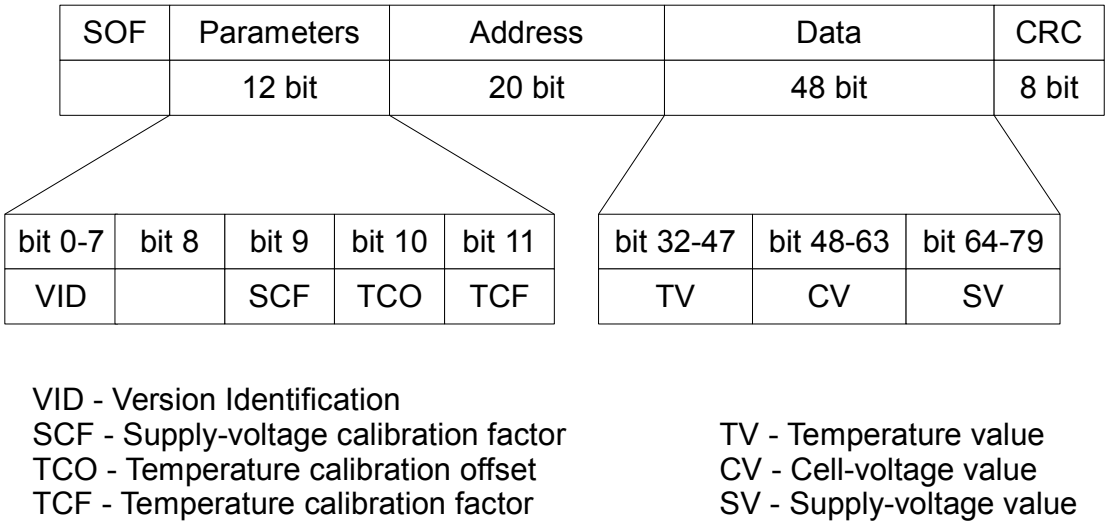


Bild 1.17.: Aufbau des verwendeten Übertragungsprotokolls

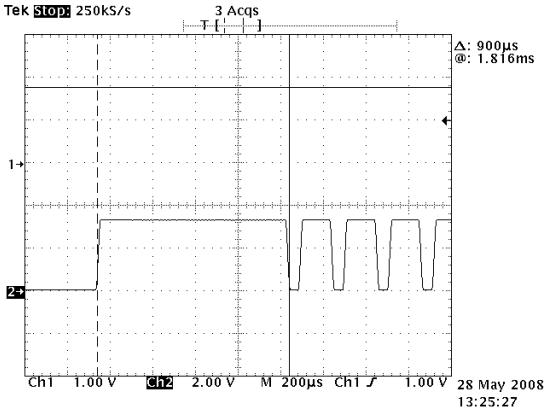
an den Empfänger gesendet werden, ist ein Antikollisionsverfahren implementiert. Angewendet wird ein asynchrones Verfahren, das ALOHA-Verfahren. Dabei handelt es sich um ein Zeitmultiplex Verfahren (TDMA). Hierbei werden die Daten in zufälligen Abständen an den Empfänger gesendet. Bild 1.17 zeigt den Aufbau des entworfenen Übertragungsprotokolls.

Das gesamte Protokoll umfasst 88 Bits. Die Daten werden unverschlüsselt gesendet. Um den Anfang einer Übertragung festzulegen, ist eine Run-In-Sequenz (SOF) vorangestellt. Diese umfaßt einen Run-In-Block und eine definierte Bitfolge von 16 Bits. In Bild 1.18(a) ist der Run-In-Block dargestellt. Die Länge des Run-In-Blocks ist auf 900 μs festgelegt.

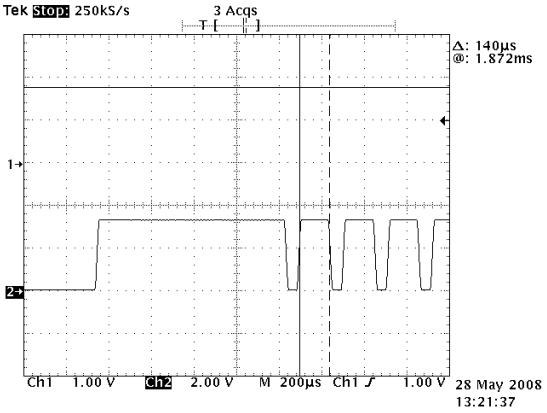
Nach dem Run-In-Block folgt eine 16 Bit breite Bitfolge, die dem Wert 1 entspricht. Diese Bitfolge dient dem Empfänger zur Kalibrierung der Übertragungsgeschwindigkeit. In Bild 1.18(b) ist der HIGH-Zustand und in Bild 1.18(c) ist der LOW-Zustand markiert. In der verwendeten Softwareversion sind die Zustände für LOW- und HIGH-Pegel unterschiedlich lang. Diese Abweichung von der Norm ist zu Testzwecken implementiert. Die Länge der Zustände beträgt 140 μs für den HIGH-Zustand und 60 μs für den LOW-Zustand. Diese Zeiten stellen die absoluten Randbedingungen dar. Bei weiterer Verwendung der Software muss dies beachtet und geändert werden. Die gesamte Zeit für ein codiertes Bit beträgt 200 μs. Daraus ergibt sich eine Datenrate des decodierten Signals von 5 kBit/s.

Die folgenden 12 Bits dienen der Parameterübergabe. Da es sich um eine unidirektionale Übertragung handelt, ist es nicht möglich, diese Parameter im Sensor zu verändern. Diese Parameter werden nur zu Informationszwecken übergeben. Die ersten acht Bit (VID) beschreiben die Software Version des Sensors. Das neunte Bit ist noch nicht zugewiesen und kann frei verwendet werden. Das zehnte Bit (SCF), elfte Bit (TCO) und zwölfte Bit (TCF) geben an, ob der Kalibrierungsfaktor der Versorgungsspannungsmessung, der

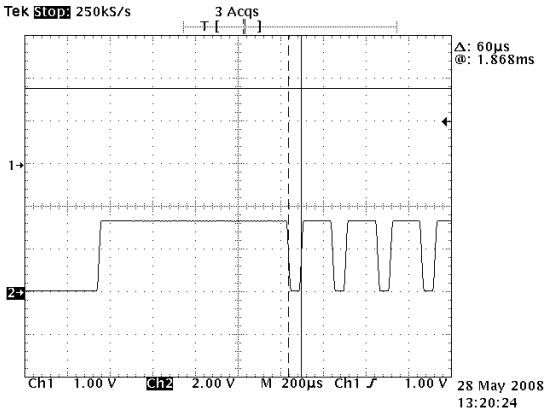
Offset der Temperaturmessung oder der Kalibrierungsfaktor der Temperaturmessung verändert wurde.



(a) Auszug aus der Run-In-Sequenz mit Markierung des Run-In-Blocks durch die Cursor



(b) Auszug aus der Run-In-Sequenz mit Markierung des HIGH-Zustands durch die Cursor



(c) Auszug aus der Run-In-Sequenz mit Markierung des LOW-Zustands durch die Cursor

Bild 1.18.: Verschiedene Auszüge aus dem Übertragungsprotokoll

Die folgenden 20 Bit enthalten die Adressinformation des Sensors. Der Bereich umfasst somit 2^{20} Adressen. Die erste frei zu vergebene Adresse ist 1. Dies liegt daran, dass die Adresse 0 dem Zustand nach der Programmierung entspricht.

Der dritte Block umfasst die eigentlichen Daten, die Messwerte. Er umfasst 48 Bit. Dabei sind jedem Messwert 16 Bit zugeordnet. Die Messwerte werden in der Reihenfolge: Temperatur (TV), Zellenspannung (CV) und Versorgungsspannung (SV) übermittelt.

Der letzte Block beinhaltet die Prüfsumme und ist 8 Bit breit. Als Prüfsummenverfahren kommt hier das CRC-Verfahren zum Einsatz. Dieses erlaubt eine einfache Erkennung von Übertragungsfehlern ohne eine Fehlerkorrektur. Um die Prüfsumme zu generieren, wird das Generatorpolynom, hier 0x00, über eine XOR-Operation mit dem ersten zu übertragenen Byte verknüpft. Der daraus resultierende Wert dient als Startwert für die nächste XOR-Operation mit dem zweiten zu übertragenden Byte. Diese Operation findet

mit allen zu übertragenden Bytes statt. Am Ende ergibt sich die CRC-Prüfsumme.

Im Empfänger wird die CRC-Prüfsumme als Startwert für die Überprüfung verwendet. Hier werden ebenfalls alle empfangenen Bytes mit Hilfe der XOR-Operation verknüpft. Am Ende der Überprüfung muss der Wert dem des Generatorpolynoms entsprechen. Ist dies nicht der Fall, ist in der Übertragung ein Fehler aufgetreten.

In Anhang C.2 ist eine Aufzeichnung des vollständigen Protokolls. Die Aufzeichnung erfolgte mit einem Oszilloskop. Die Rohdaten der Oszilloskopmessung wurden gespeichert und mit Hilfe von MATLAB dargestellt. Der MATLAB Code ist in Anhang B.2.3 zu finden.

2. Realisierung

Bei der Auswahl der Hardware mussten verschiedene Kriterien für den Sensor und für das Steuergerät berücksichtigt werden. Das wichtigste Kriterium ist ein besonders niedriger Stromverbrauch. Beide Geräte sind im Fahrzeug untergebracht und beziehen ihre Versorgungsspannung aus einer Batterie. Je nach Anwendung ist das die Starterbatterie im Automobil oder die Antriebsbatterie in Flurförderzeugen. Besonders im Automobil ist daher eine stromsparende Lösung wichtig. Dies liegt daran, dass bei längerem Stillstand die Batterie nicht entladen werden darf. Um dies zu verhindern ist ein maximaler Ruhestrom von 36 mA für alle Verbraucher im Kraftfahrzeug vorgeschrieben. Somit darf die Stromaufnahme, auch bei Betrieb von Sensor und Steuergerät, nicht mehr als einige Milliampere betragen.

Bei der Datenübertragung ist eine ausreichende Geschwindigkeit und eine genügende Reichweite wichtig. Ebenfalls muss berücksichtigt werden, dass das Überwachungssystem auf schon vorhandene Systeme aufsetzbar ist. Weiterhin sind sicherheitsrelevante Aspekte und der Kostenfaktor zu beachten. Nach Berücksichtigung dieser Aspekte ist die Entscheidung getroffen worden, die Datenübertragung per Funk zu realisieren. Hierbei ist keine Verkabelung notwendig. Diese senkt die Kosten und der Einsatz in schn vorhandenen Systemen wird erleichtert. Zudem sind Sensor und Steuergerät galvanisch getrennt ohne großen schaltungstechnischen Aufwand zu betreiben.

Um eine ausreichende Geschwindigkeit der Funkdatenübertragung bereitzustellen, sind vorab konventionelle Systeme untersucht worden. Bei dem entwickelten System kommen je nach Anwendung 6 bis 40 Sensoren zum Einsatz. Jeder Sensor muss mindestens einmal pro Minute seine Daten zum Steuergerät übertragen können. Die Daten bestehen, wie in Kapitel 1.3.4 beschrieben, aus 88 Bit plus der Startsequenz. Bei einer für 125 kHz Transponder üblichen Modulationsfrequenz von $f_{mod} = 1\ kHz$ ergibt sich für einen Sensor eine Übertragungszeit von:

$$t_{trans} = n_{bit} * \frac{1}{f_{mod}} + t_{start} = 88 * \frac{1}{1\ kHz} + 4,1\ ms = 92,1\ ms \tag{2.1}$$

Bei 40 Sensoren beträgt die reine Übertragungszeit: $t_{trans40} = t_{trans} * 40 = 3,684\ s$. Da die Daten in zufälligen Intervallen übertragen werden, ist die Bandbreite, ohne Übertragungsfehler zu berücksichtigen, nicht ausreichend. Somit kommt eine Übertragung im 125 kHz Bereich nicht in Frage.

Die vorliegende Arbeit beruht auf Ergebnissen die von Prof. Dr.-Ing. Karl-Ragmar Riemschneider zur Verfügung gestellt wurden. Daher sind Überlegungen, die den Sensor betreffen, schon im Vorfeld geschehen und nicht vom Autor dieser Arbeit durchgeführt worden (siehe Kapitel 2.1).

2.1. Sensor-System

Im folgenden Kapitel wird das Sensor-System beschrieben. Da dieser Teil der Hardware zur Verfügung gestellt und nicht selbst entwickelt wurde, wird nur eine Übersicht über das gesammte System gegeben. Bild 2.1 beschreibt schematisch den Aufbau des Sensor-Systems. Der Mikrocontroller übernimmt die zentrale Rolle. Die Messwerte werden mit den vorhandenen ADC aufgenommen und an den Sender übertragen. Dieser sendet die Daten an das Steuergerät.

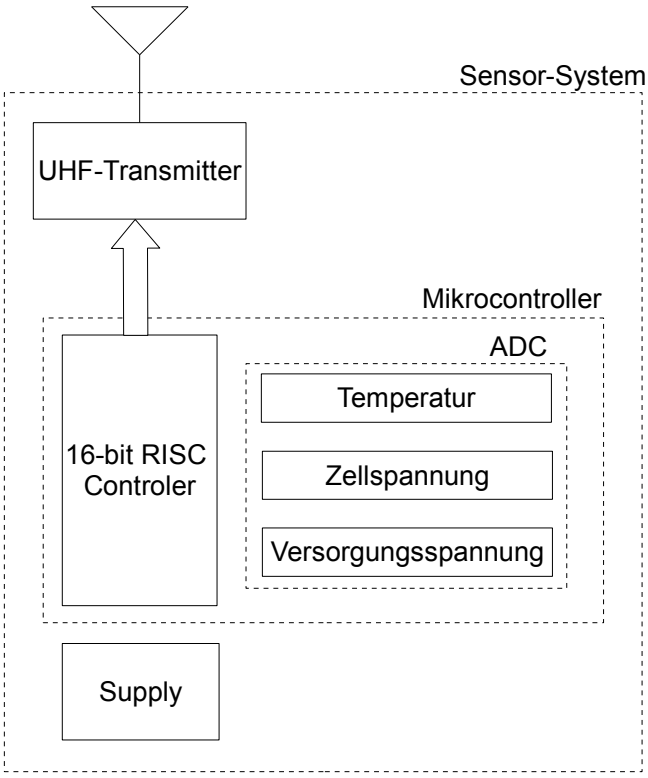
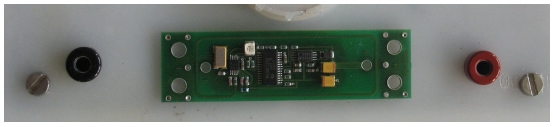


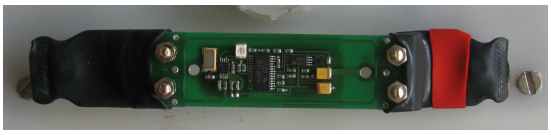
Bild 2.1.: Blockschaltbild des Sensor-Systems

Um den geringen Stromverbrauch zu garantieren, sind geeignete Komponenten gewählt worden. Als Mikrocontroller wird ein Controller der MSP430er Serie von Texas Instruments verwendet. Mikrocontroller aus dieser Serie sind derzeit die stromsparensten Mikrocontroller auf den Markt. Um die Messaufgaben zu erledigen, ist ein 10 Bit Analog zu Digital Wandler und eine interne Diode zur Temperaturbestimmung vorhanden.

Die Versorgung stellt ein Problem dar. Die Spannung einer Zelle schwankt von 1,5 V bei starker Beanspruchung bis 2,4 V beim Ladevorgang. Bei schweren Tiefentladungen oder längeren Standzeiten kann die Spannung noch weiter sinken. Das Problem ist hierbei nicht die maximal zu erwartende Spannung, sondern die minimale Spannung. Der ausgewählte Mikrocontroller benötigt laut Spezifikation eine Spannung von mindestens 1,8 V, der Senderchip sogar 2,1 V. Als Lösung dieses Problems ist ein Step-Up-Converter verbaut.



(a) Sensorplatine ohne Anschlußfahnen und Schutzschlauch



(b) Sensorplatine mit geschützten Anschlußfahnen

(c) Sensorplatine mit Anschlußfahnen und Schutzschlauch

Bild 2.2.: Mechanischer Aufbau des Sensor-Systems

Um den Sensor vor äußeren Einflüssen zu schützen, wurde das gesamte Sensor-System in einem Schutzschlauch annähernd luftdicht eingeschweißt. Dies ist notwendig, da austretende agressive Gase beim Ladevorgang die Bauteile des Sensor-Systems angreifen und zerstören könnten. Bild 2.2 zeigt das Sensor-System vor, während und nach dem Einschweißvorgang. Um Kontakt mit der Batteriezelle herzustellen, sind 4mm Stecker an die Anschlussfahnen montiert. Diese werden in die auf der Batterie vorhandenen Klemmschrauben aufgesteckt.

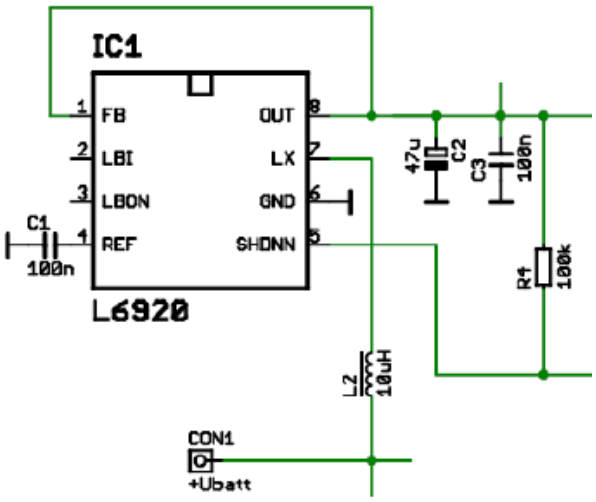


Bild 2.3.: Beschaltung des L6920 [21]

2.1.1. Step-Up-Converter

Als Spannungsversorgung kommt ein Step-Up-Converter der L6920 Serie zum Einsatz. Eingesetzt werden Step-Up-Converter in Kleinstgeräten, die mit einer niedrigen Versorgungsspannung auskommen müssen. Zum Einschalten werden nur 0,8 V benötigt, im

Betrieb sogar nur 0,5 V. Damit ist es möglich, eine Ausgangsspannung von 2 V bis 5,2 V zu erzeugen. Je nach Eingangsspannung und Laststrom ist eine Effizienz von bis zu 95% gewährleistet. Der niedrige Stromverbrauch der Schaltung bewirkt das Herabsetzen der Effizienz auf ungefähr 90%. [24]

Bild 2.3 stellt die Minimalbeschaltung dar wie sie auch im Sensor verwendet wird. Durch Beschaltung des FB Eingangs mit der Ausgangsspannung werden 3,3 V bereitgestellt. [24]

2.1.2. UHF-Transmitter

Im Vorfeld wurde erläutert, welche Anforderungen an das System gestellt werden. Die Datenübertragungsrate spielt dabei eine große Rolle. In Kapitel 2 wurde bewiesen, dass ein 125 kHz Transpondersystem kaum den Ansprüchen gerecht wird. Um eine ausreichende Bandbreite zur Verfügung zu stellen, ist das System mit einem Transmitter ausgerüstet, der im 434 MHz Bereich überträgt. Mit diesem Baustein ist es möglich, eine maximale Datenübertragungsrate von 20 kbps zu erreichen. Dies entspricht dem 20-fachen eines typischen 125 kHz Systems. Setzt man diese Werte in Formel 2.1 ein, so erhält man:

$$t_{trans} = 88 * \frac{1}{20\ kHz} + 4,1\ ms = 8,5\ ms$$

Bei 40 Sensoren ist eine reine Übertragungszeit von: $t_{trans40} = 340\ ms$ erreicht. Mit dieser Übertragungsgeschwindigkeit ist es möglich auch kurzzeitige Ereignisse zu erfassen. In der vorliegenden Arbeit werden die Möglichkeiten des Systems aber nicht ausgereizt. Es wird eine statistische Auswertung angestrebt, wobei es nicht nötig ist, jede Sekunden einen Wert zu erhalten. Bei anderen Systemen könnte dies eine Rolle spielen. Mit der Auslegung des Systems ist somit eine andere Anwendung nicht grundlegend ausgeschlossen.

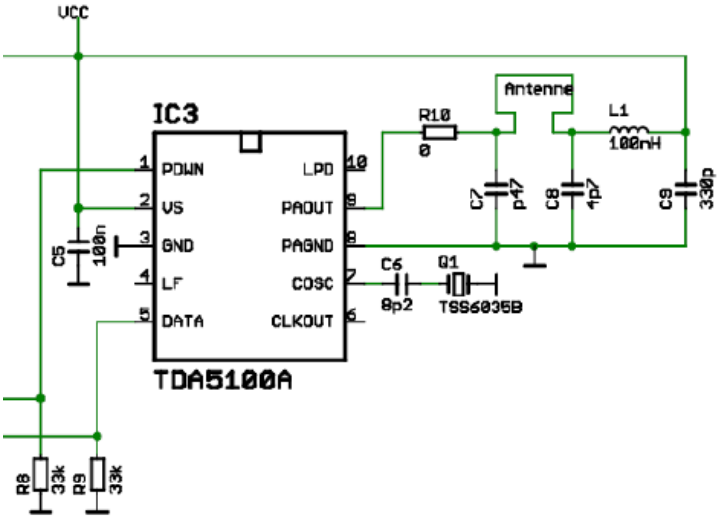


Bild 2.4.: Beschaltung des TDA5100A [21]

Der verwendete Baustein ist aus der TDA5100 Reihe mit der Zusatzbezeichnung A. Der Zusatz A bezieht sich auf den Frequenzbereich. Die TDA5100 Serie kann im 434 MHz

und im 869 MHz Bereich mit ASK oder FSK übertragen. Die Bausteine mit dem Zusatz A sind auf den 434 MHz Bereich und ASK beschränkt. [13]

In Bild 2.4 ist die Beschaltung des TDA5100A dargestellt. Die Komponenten C7-C9, L1 und R10 werden zur Abstimmung der Antenne benötigt. Diese ist als Schleifendipol direkt auf der Platine aufgebracht. Für die Berechnung der Antenne und die benötigten Komponenten sind folgende Wert ausschlaggebend: die Länge der Schleife (l), der Abstand der Windungen (D) und der Durchmesser der Windungen (d_1, d_2) (Bild 2.5).

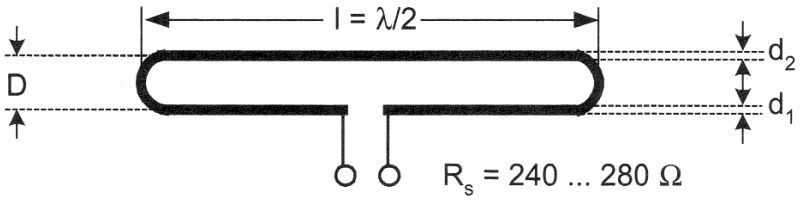


Bild 2.5.: Darstellung eines Schleifendipol mit der Länge $\lambda/2$ [14]

Die Wellenlänge λ beträgt bei 434 MHz: $\lambda = 69,08\ cm$. Die Maße der Antenne auf dem Sensor lauten: $l = 50\ mm,\ D = 16\ mm,\ d_1 = d_2 = 0,9\ mm$. Diese muss, da ihre Länge $l = \lambda/2$ oder $l = \lambda/4$ unterschreitet, elektrisch verlängert werden. Das heißt es erfolgt eine Anpassung mit einem geeigneten Netzwerk am Speisepunkt. Dies muss erfolgen, da sonst die kapazitiven die induktiven Anteile der Antenne überwiegen. Bei Antennen dieser Art spricht man von *kurzen Antennen*. [8]

2.1.3. Mikrocontroller

Die MSP430-Mikrocontrollerfamilie umfasst eine Vielzahl von leistungsstarken Mikrocontrollern. Der besondere Vorteil liegt im extrem niedrigen Stromverbrauch. Dieser kann durch die Auswahl von fünf Betriebsarten noch weiter gesenkt werden. Verschiedene Compiler erlauben es, den Mikrocontroller in *C* oder *Assembler* zu programmieren.

Einsatz im Sensor-System findet der Mikrocontroller mit der Bezeichnung MSP430x1232. Mehrere Gründe sprechen dafür:

- Stromverbrauch im μA Bereich
- 16 Bit RISC Architektur mit einer maximalen Frequenz von 8 MHz
- 8 kB Programmspeicher, 256 Byte FLASH und 256 Byte RAM
- 16 Bit Timer zur Signalgenerierung
- 10 Bit ADC für Spannungsmessungen
- Interne Diode zur Temperaturbestimmung
- ISP - In System Programming

Weitere Information sind dem Handbuch zu entnehmen. [25]

Die Messwertaufnahme beschränkt sich auf drei verschiedene Spannungsmessungen. Um die Temperatur zu bestimmen, wird die Spannung über eine interne Diode gemessen. Für diese Messung ist keine äußere Beschaltung notwendig. Durch Setzen eines Konfigurationsbits wird der ADC-Kanal intern mit der Diode verbunden. Bei der Temperaturmessung ist eine Kalibrierung erforderlich. Dies kann entweder in der Software des Sensor-Systems oder in der Software zur Auswertung der Ergebnisse geschehen. Bei der vorliegenden Arbeit wurden die Temperaturwerte in der MATLAB Software kalibriert. Dazu wurden verschiedene Temperaturen aufgenommen, die Differenz zu den realen Werten berechnet und dann eine Differenzgleichung abgeleitet. Die Kalibrierung ist in Kapitel 3.1.1 genauer beschrieben.

Die Versorgungsspannung wird ebenfalls intern gemessen und durch das Setzen eines Konfigurationsbits gestartet. Bei der Messung der Zellenspannung ist etwas Schaltungsaufwand erforderlich. Hier ist ein Spannungsteiler und ein Kondensator zur Spannungsglättung vorgesehen. Der Eingang des Spannungsteilers ist direkt an den positiven Pol der Zelle angeschlossen. Der Spannungsteiler besteht aus drei in Reihe geschalteten Widerständen mit jeweils 6,8kΩ. Der Abgriff für den Eingang des ADC ist direkt nach dem ersten Widerstand. Somit wird eine Spannung gemessen, die zwei Drittel der realen Spannung entspricht. Der gesamte Schaltplan des Sensor-Systems ist in Bild D.4 dargestellt.

2.1.4. Leistungsmessung

Um den tatsächlichen Verbrauch des Sensors zu messen, wurde ein Messwiderstand in Reihe zur Spannungsversorgung geschaltet. An diesen ist mit Hilfe eines Oszilloskops die Spannung gemessen worden. Da der Messwiderstand 1 Ω beträgt, ist die Spannung gleichbleibend mit dem Strom. Die Versorgungsspannung wurde im Vorfeld gemessen und als Referenz (R1 Bild 2.6) im Oszilloskopausdruck eingefügt. Durch Multiplikation der beiden Kurven ist die Leistungsaufnahme berechnet und als Kurve dargestellt worden (Math1 Bild 2.6).

Bild 2.6 stellt den gemessenen Leistungs-/Stromverlauf dar. Der Bereich 2 enspricht der Datenaufnahme und -übertragung. Die Übertragung sollte nach den in Kapitel 1.3.4 vorgestellten Zeiten zirka 20 ms betragen. Durch Temperaturschwankungen werden Differenzen von ±15% erwartet. Die hier ermittelte Zeit von 24,4 ms entspricht voll und ganz den Erwartungen. Der maximale Strom beträgt dabei 47,2 mA, die daraus resultierende maximale Leistung 62,4 mW. Der folgende Bereich 3 entspricht der restlichen Einschaltdauer der Kontroll-LED auf dem Sensor. Diese beträgt laut Berechnung 50 ms (Anhang B.1.1). Die hier gemessene Zeit von 58ms ist auf Temperaturschwankungen zurückzuführen. Der Minimalstrom beträgt 4,4 mA welcher in Bereich 1 gemessen ist. Der Verbrauch im Ruhezustand ist somit 5,63 mW.

Je nach Anwendung ist der Stromverbrauch schon ausreichend gering. Es gibt aber noch Möglichkeiten diesen zu senken. Die Software könnte verändert werden, indem der Ruhezustand in einen Sleep-Modus des Mikrocontrollers resultiert. Weiterhin ist es möglich die Kontroll-LED zu entfernen und dadurch den Strombedarf zu senken. Je nach Anwendung sollten solche Maßnahmen in Betracht gezogen werden. Für den Einsatz im Gabelstapler

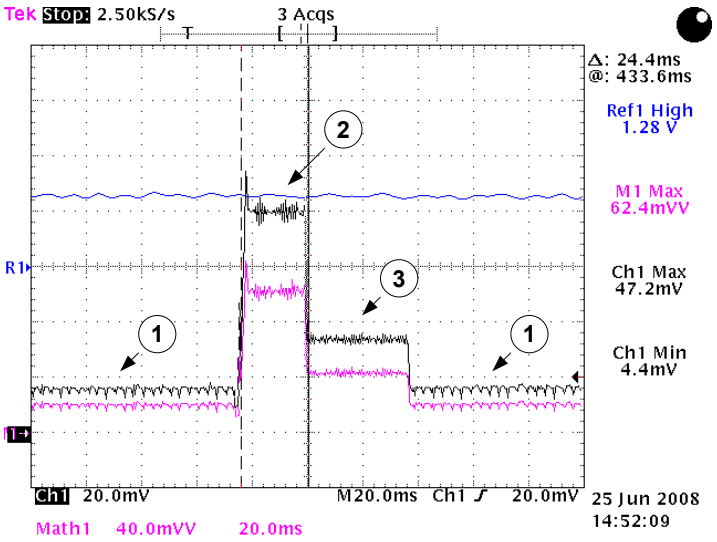


Bild 2.6.: Leistungsmessung: blau = Versorgungsspannung, schwarz = Stromaufnahme, pink = Leistungsaufnahme; 1 = Ruhezustand, 2 = Übertragungsfall, 3 = Kontroll-LED eingeschaltet

ist der Stromverbrauch ausreichend und es werden keine weiteren Bestrebungen diesen zu senken in vorliegender Arbeit behandelt.

2.1.5. Software

Die Software für das Sensor-System ist von Dipl.-Ing. Bernd Fischer geschrieben und getestet worden. Die Anforderungen an das Sensor-System sind überschaubar. Die Messwerte sind mit den internen ADC aufzunehmen und seriell an den UHF-Transmitter zu übergeben. Der gesamte Quellcode ist in Anhang B eingefügt.

In Bild 2.7 ist der Ablauf der Sensor-Softare dargestellt. Zu Beginn erfolgt die Initialisierung. Hier wird die interne Peripherie des Mikrocontrollers konfiguriert. Dazu gehören der verwendete Timer, die IO-Ports und der ADC mit seinen verschiedenen Kanälen. Ebenfalls wird der Datenflash ausgelesen. In ihm befinden sich Informationen, die das Sensor-System betreffen, wie die Addresse und verschiedene Korrekturwerte für die Spannungsmessungen. Um Daten in den FLASH des Mikrocontrollers schreiben zu können, muss ein Zeiger auf den Datenbereich vereinbart werden. Die genaue Vereinbarung ist in Listing 2.1 dargestellt. Der Datenflash beginnt bei den hier verwendeten MSPx1232 ab der Adresse 0x1000. Nach dieser Adresse ist die Struktur mit dem Namen *InfoBlock* abgelegt. Um Elemente aus dieser Struktur zu lesen, ist die gleiche Syntax wie bei herkömmlichen Pointern anzuwenden (Listing 2.2).

Zur Speicherung von Werten ist etwas mehr Aufwand notwendig. Da bei einem FLASH die einzelnen Bits nicht einzelnd gesetzt werden können, muss der jeweilige Bereich des FLASH gelöscht werden bevor er neu beschrieben werden kann. Da bei dem Sensor-System nur wenige Daten im FLASH abgelegt werden, wurde diese Funktion nicht implementiert.

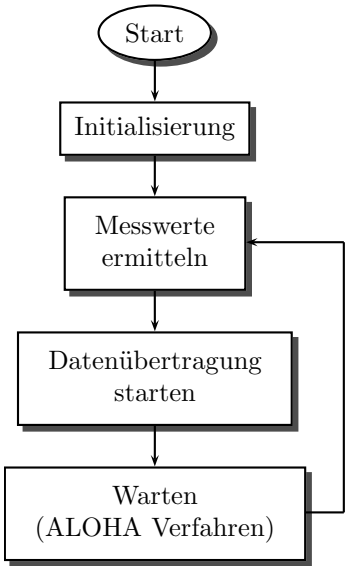


Bild 2.7.: Ablaufdiagramm der Sensorsoftware

Statt dessen werden die Daten mit Hilfe des Debuggers in den FLASH geschrieben. Weitere Informationen über die Verwendung des Datenflash sind dem Datenblatt [25] zu entnehmen.

Listing 2.1: Vereinbarung der Systemdaten im Datenflash

```
....
struct InfoBlock
{
        unsigned long magicID;
        unsigned long sonsorID;
        // Daten für Kalibrierung
        unsigned short vFactor;
        unsigned short tOffset;
        unsigned short tFactor;
};
// Daten im FLASH des Mikrocontrollers ablegen!!!
const struct InfoBlock *const infoBlock = (const struct InfoBlock *)0x1000;

const unsigned long magicID = 0x64490F01;
const unsigned char version = 0x01;
....
```

In Listing 2.1 sind zwei weitere Konstanten definiert: *magicID* und *version*. Die Versionsnummer ist jeder Datenübertragung, im Parameterblock unter VID, angehängt (Kapitel 1.3.4). Die Konstante *magicID* ist, wenn der Sensor konfiguriert ist, in der Struktur *InfoBlock* enthalten. Entspricht der Wert in der Struktur dem der Konstanten *magicID*, ist der Sensor konfiguriert. Entspricht er nicht dem der Konstanten, ist der Sensor nicht konfiguriert und die Adresse 0 wird dem Sensor vergeben (Listing 2.2). Des Weiteren werden die Korrekturfaktoren aus dem FLASH gelesen und abgespeichert.

Listing 2.2: Auslesen der Systemdaten aus dem Datenflash des Mikrocontrollers

```
1  int main(void)
2  {
3          if (infoBlock->magicID != magicID)
4                  initTransmit(0x00000000);
5          else
6          {
7                  unsigned long sonsorID = infoBlock->sonsorID;
8                  sonsorID &= 0x000FFFFF;
9                  unsigned long versionID = version;
10                 versionID <<= 24;
11                 sonsorID |= versionID;
12
13                 // Auslesen der Korrekturfaktoren
14                 if (infoBlock->vFactor != 0xFFFF)
15                 {
16                         vFactor = infoBlock->vFactor;
17                         sonsorID |= 0x00400000;
18                 }....
```

Nachdem die internen Komponenten initialisiert sind, werden die beiden externen Hardwarebausteine aktiviert. Dies geschieht durch Setzen eines bestimmten Ausgangspegels (HIGH oder LOW), eines IO-Pins. Im Falle des Step-Up-Converters entspricht das einem HIGH-Pegel am Pin 2.0 und im Falle des UHF-Transmitters einem HIGH-Pegel am Pin 1.1 (Bild D.4).

Nach der Initialisierung werden die Messwerte ermittelt. Dazu wird der Step-Up-Converter deaktiviert und die gesamte Schaltung durch einen Kondensator versorgt. Diese Maßnahme ist nötig, da durch den Step-Up-Converter starke Schwankungen in der Versorgungsspannung auftreten. Die Temperatur wird über die interne Diode gemessen. Dazu wird der ADC-Kanal intern mit der Diode verbunden und eine Referenzspannung angelegt. Die genauen Einstellung sind im Anhang B.1.1 ab Zeile 290 dargestellt. Zur genauen Aufschlüsselung sind sie mit dem Datenblatt [25] zu vergleichen, in welchem auch die linearisierte Funktion zur Errechnung der Temperatur zu finden ist. Diese lautet:

$$Temperatur/°C = \frac{(\frac{A10}{1024} * 1500\ mV) - 986\ mV}{3.55\ mV} \tag{2.2}$$

Um den Rechenaufwand so gering wie möglich zu halten, ist die Berechnung vereinfacht. Diese vereinfachte, skalierte Berechnung der Temperatur lautet:

$$Temperatur_{skaliert} = \frac{(A10 - 673) * 423}{128} \tag{2.3}$$

Die Versorgungsspannung wird über einen internen Spannungsteiler gemessen. Die ermittelte Spannung beträgt dabei $1/2$ der gesamten Spannung. Die Einstellungen des ADC sind im Anhang B.1.1 ab Zeile 309 dargestellt.

Die Zellenspannung wird über einen externen Spannungsteiler gemessen, welcher wie in Kapitel 2.1.3 beschrieben aufgebaut ist. Die ermittelte Spannung beträgt $2/3$ der realen Spannung. Die Einstellungen des ADC sind dem Anhang B.1.1 ab Zeile 329 zu entnehmen. Die Formel zur Berechnung der Zellenspannung lautet:

$$U_{Zelle}/V = \frac{3}{2} * \frac{A10 * 2500\ mV}{1023} \tag{2.4}$$

Auch hier wird die Formel vereinfacht und ein Skalierungsfaktor eingefügt. Die Formel lautet dann:

$$U_{Zskaliert} = \frac{A10 * 3750 + 125}{250} \tag{2.5}$$

Wie beschrieben sind alle drei Messwerte skaliert. Tabelle 2.1 stellt die Skalierungsfaktoren dar, die bei der Analyse berücksichtigt werden müssen.

	Skalierungsfaktor
Temperaturmessung	x/8
Zellspannungsmessung	x/4096
Versorgungsspannungsmessung	x/4096

Tabelle 2.1.: Skalierungsfaktoren der Messungen

Nachdem alle Messwerte ermittelt sind, wird die Übertragung gestartet. Diese läuft vollkommen im Interrupt des Timers ab. Bild 2.8 beschreibt den genauen Ablauf im Timerinterrupt. Er ist in Intervalle aufgeteilt, die nur einem Teil einer Bitlänge entsprechen. Somit wird der Interrupt mehrmals aufgerufen, um dem Zustand von einem Bit zu entsprechen. Ist die Übertragung abgeschlossen, wird ein Bit gesetzt. Daraufhin wird im Interrupt erkannt, dass keine Daten zur Übertragung bereitstehen und die Interruptroutine wird verlassen.

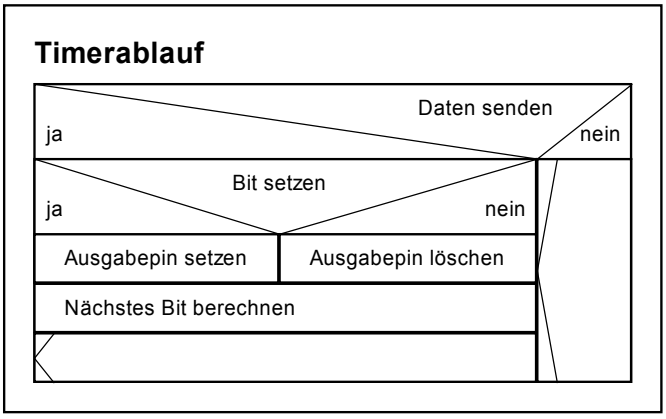


Bild 2.8.: Struktogramm des Timerinterruptes

Nachdem die Übertragung gestartet wurde, wird das Sensor-System in einen Ruhezustand versetzt. Die Zeit, die gewartet wird, ist ein zufälliges Intervall zwischen 1,8 - 3,6 Sekunden. Der Wartevorgang ist ein Teil des Antikollisionsverfahrens. Da bei jedem Sender die Wartezeit mit Hilfe eines linearen Schieberegisters berechnet wird, ist die Wahrscheinlichkeit sehr gering, dass zwei Sensoren zur gleichen Zeit Daten übertragen. Zum Thema lineares Schieberegister und Zufallszahlen kann in einschlägiger Literatur nachgelesen werden.

2.2. Steuergerät

Das Steuergerät beinhaltet einen UHF-Receiver, einen Mikrocontroller zur Verarbeitung der Daten, einen FLASH-Speicher zur Datenspeicherung, ein LCD zur Zustandsdarstellung und eine serielle Schnittstelle. Bei der Auswahl der Komponennten sind die Kriterien aus Kapitel 2 wieder zu berücksichtigen. Auch hier ist der Stromverbrauch eines der wichtigsten Auswahlkriterien. Bild 2.9 beschreibt den schematischen Aufbau des Steuergerätes.

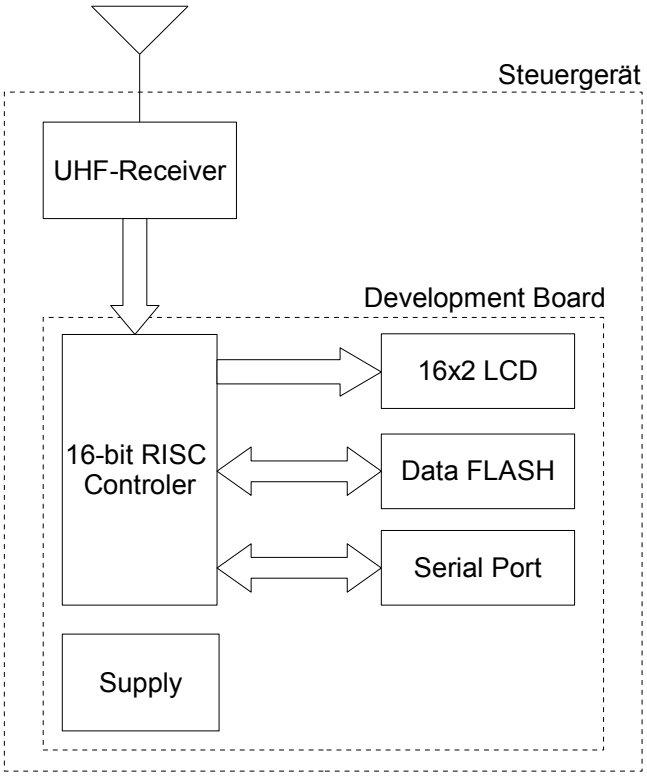


Bild 2.9.: Blockschaltbild des Steuergerätes

Um den Aufbau möglichst einfach zu gestalten, ist ein Entwicklungsboard eingesetzt. Dieses übernimmt die Datenverarbeitung und -ausgabe und ist von der Firma Olimex Ltd. entwickelt worden. Den UHF-Empfang erledigt ein Hybrid-Chip der Firma RF Monolithics Incorporated. Somit sind nur zwei grundlegende Komponenten erforderlich, die keinen oder nur einen geringen Schaltungsaufwand erfordern (Bild 2.9).

2.2.1. Entwicklungsboard

Das verwendete Development Board trägt die Bezeichnung *MSP430-169STK*. Hergestellt wird es von der Firma *Olimex Ltd.*. Auf dem Board kommt ebenfalls ein Mikrocontroller der MSP430er Serie zum Einsatz. Hierbei handelt es sich um den MSP430x169 mit einer

16 Bit RISC CPU. Im Gegensatz zum MSPx1232 ist dieser Mikrocontroller mit einer sehr großen Auswahl an Peripherie ausgestattet:

- 12 Bit ADC mit 10 Kanälen
- 12 Bit DAC
- Zwei 16 Bit Timer
- Zwei serielle Schnittstellen, asynchron oder synchron
- 60 kB Programmspeicher, 256 Byte FLASH und 2 kB RAM

Hier nur ein kurzer Auszug aus dem Datenblatt [25], weitere Informationen können eben diesem entnommen werden. Diese umfangreiche Peripherie ist ebenfalls notwendig, um die zur Verfügung gestellten Aufgaben des Development Boards zu bewältigen.

Entwickelt ist das Development Board zur Verarbeitung von analogen Signalen, explizit zur Verstärkung von Audio Signalen. Es stehen jeweils zwei Stereo Klinkenbuchsen für Eingangs- und zwei für Ausgangssignale zur Verfügung. Alle vier Buchsen besitzen jeweils einen Verstärker. Diese sind so konfiguriert wie es die Eingangs- oder Ausgangssignalpegel erfordern. Um die Lautstärke steuern zu können, ist jeweils ein Potentiometer an die Ausgangsverstärkerschaltung angeschlossen.

Neben der Ein- und Ausgabe von analogen Signalen sind weitere Funktionen vorgesehen. Um die Signale zu speichern, ist ein 8 MB großer FLASH-Speicher installiert. Um diesen nutzen zu können, ist die Ansteuerung per Software zu realisieren. Zur Anzeige stehen ein zwei-reihiges 16 Character LCD und zwei Leuchtdioden zur Verfügung. Die Ansteuerung erfolgt ebenfalls per Software. Zur Bedienung sind drei frei programmierbare Taster vorgesehen. Zum Datenaustasch steht eine serielle Schnittstelle zur Verfügung. Diese nutzt eine der Hardwareschnittstellen des Controllers und kann mit einer Übertragungsgeschwindigkeit von bis zu 119200 kbps betrieben werden.

Zur Programmierung des Mikrocontrollers ist eine JTAG-Anschlussleiste auf dem Board angebracht. Die übrigen IO-Pins sind über fünf Anschlussleisten nach außen geführt. Die Anschlussleiste mit der Bezeichnung *EXT* (Bild D.2) wird zum Anschluss des UHF-Receivers verwendet. Hierzu wurde die senkrechte Stiftleiste durch eine um 90° abgewinkelte Stiftleiste ersetzt. So ist es möglich, die Platine mit dem UHF-Receiver direkt aufzustecken (Bild 2.11 unten rechts). Der UHF-Receiver benötigt drei Leitungen: Versorgungsspannung, Masse und eine Datenleitung. Das Einlesen erfolgt über einen IO-Pin (Bild D.2 Pin 1.1), der als externe Interruptquelle genutzt werden kann und intern einen Timerinterrupt auslöst.

Weitere Informationen sind auf der Herstellerhomepage (*http://www.olimex.com/*) zu finden.

2.2.2. UHF-Receiver

Der UHF-Receiver besteht aus zwei Komponenten. Die erste Komponente umfasst den UHF-Receiver, die zweite eine Verstärkerstufe (Bild 2.11). Der UHF-Receiver besteht aus

einer kleinen Platine auf der der Hybrid-IC und einige externe Komponenten untergebracht sind. Der Hybrid-IC ist von der Firma RF Monolithics Incorporated entwickelt und trägt die Bezeichnung *RX5000*. Kurz die wichtigsten Daten:

- 434 MHz Frequenzbereich
- Datenübertragungsrate bis 115,2 kbps
- Stromverbrauch < 4 mA
- Betriebsspannung: 2,2 V - 3,7 V

Der Empfängerbaustein wird in der Grundschaltung für OOK-Modulation betrieben (Bild 2.10). Die Werte für die externen Komponenten und weitere Informationen können dem Datenblatt entnommen werden [20].

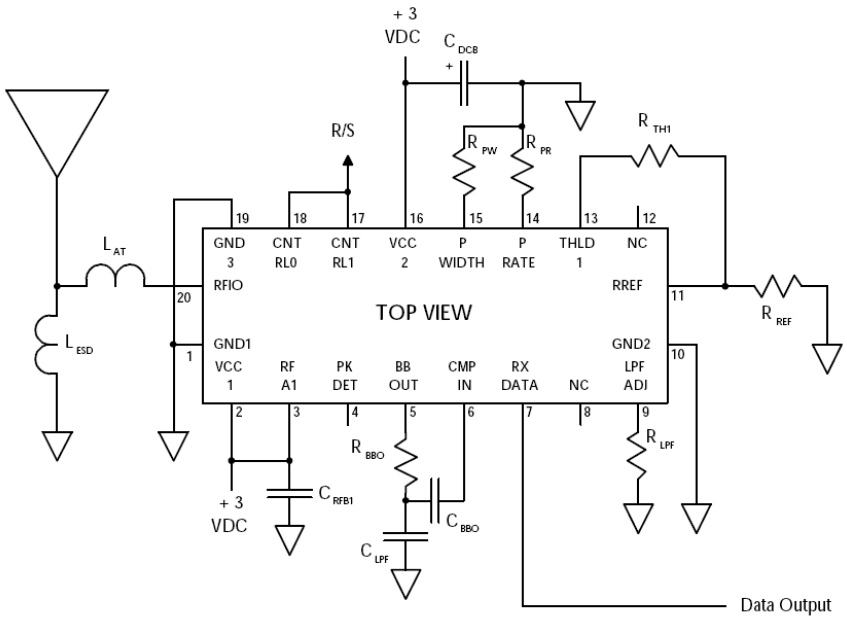


Bild 2.10.: Grundschaltung des RX5000 für OOK [20]

Der digitale Ausgang des RX5000 ist in der Lage eine parallele Last von 500 $k\Omega$ und 10 pF zu treiben. Um einen fehlerfreien Betrieb zu gewährleisten, ist eine zweite Platine mit einer Verstärkerstufe entwickelt worden. Die Verstärkerstufe besteht aus einem XOR-Gatter mit der Bezeichnung 4070. In CMOS-Bauweise besitzt dieses eine Eingangskapazität von 5 pF. Mit dieser Anordnung ist ein fehlerfreier Betrieb gewährleistet. Der Ausgang des Gatters treibt den Eingang des Mikrocontrollers. Zur visuellen Empfangskontrolle ist eine LED parallel zum Datensignal geschaltet. Der Schaltplan ist im Anhang D.1.1 dargestellt.

2.2.3. Aufbau

Alle Komponenten sind in einem bruchfesten Metallgehäuse untergebracht. Um die einzelnen Komponenten im Gehäuse fixieren zu können, wurde eine Plexiglasscheibe im Gehäuse verschraubt. Auf dieser sind Abstandshalter befestigt, die zur Montage der Platinen dienen. Unterhalb der Platinen ist eine Batterie angebracht. Dieses ist notwendig, um eine autonome Versorgung zu garantieren. Die Anschlüsse für die serielle Schnittstelle, die Antenne und die Stromversorgung sind über Buchsen nach außen geführt. Die Stromversorgung schaltet sich automatisch um wenn von außen eine Versorgung eingesteckt wird (Bild 2.11).

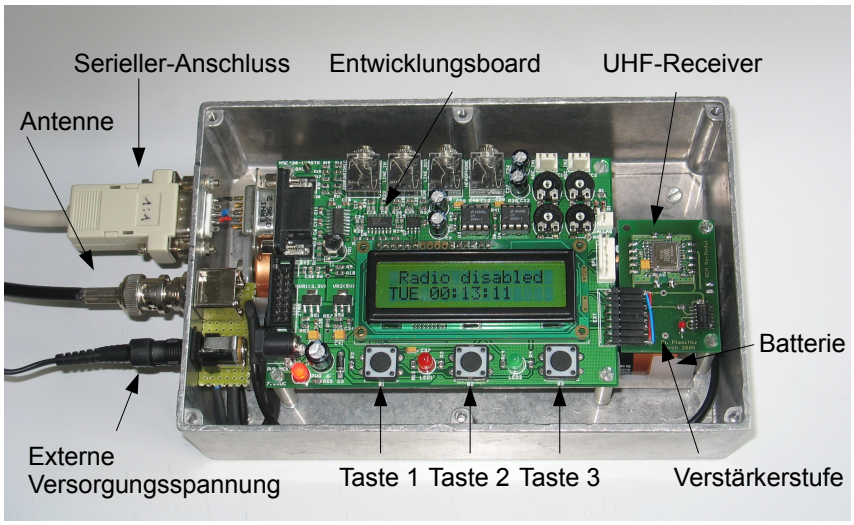


Bild 2.11.: Aufbau des Steuergerätes

Als Abdeckung ist ebenfalls eine Plexiglasscheibe verwendet worden. Um das Steuergerät weiterhin bedienen zu können, sind Löcher an die Position der Taster gebohrt worden. Zur Bedienung werden Stifte benötigt, die gleichzeitig einen guten Schutz vor unbefugtem Zugriff darstellen. Zur Erstellung des Fotos (Bild 2.11) ist die Plexiglasabdeckung entfernt worden und daher auf dem Bild nicht zu erkennen.

2.2.4. Software

Die Software für das Steuergerät ist in der Programmiersprache *C* modular geschrieben. Die Modulinhalte sind so gewählt, dass sie einfach in andere Projekte implementiert werden können. Jedes Modul besteht aus einer Header- und einer Quellcode Datei. Alle Dateien des gesamten Projektes sind im Anhang B.1.2 zu finden.

Die Software gliedert sich in drei Gruppen in denen die verschiedenen Module untergebracht sind. Die erste Gruppe beschreibt den Empfang der Daten. Hier werden die Daten vom UHF-Receiver eingelesen, überprüft und gegebenfalls zwischengespeichert. Alle Prozesse finden im Interrupt des Timers statt.

Die zweite Gruppe beinhaltet die Benutzeroberfläche. Dazu gehört die Ausgabe auf dem LCD Display und über die serielle Schnittstelle an einen PC, sowie die Bedienung

per Tasten oder serieller Schnittstelle. Die Prozesse finden in der Hauptschleife und im Interrupt der seriellen Schnittstelle statt.

Die dritte und letzte Gruppe umfasst das Thema Datenspeicherung. Hier werden die geprüften Daten mit einer Zeitmarke versehen und in den Speicher des Entwicklungsboards geschrieben. Die Prozesse finden aus der Hauptschleife heraus und im Interrupt des Watchdog Timers statt.

Datenempfang

Die Daten werden seriell vom UHF-Receiver eingelesen. Dazu wird Pin 1.1, der als externe Interruptquelle für den TimerA vorgesehen ist, mit dem Ausgang der Verstärkerschaltung verbunden. Tritt auf der Datenleitung ein Wechsel von 0 auf 1 auf, so springt der Mikrocontroller in den Interrupt und überprüft den empfangenen Code. Es werden nur die Wechsel von 0 auf 1 betrachtet, da so direkt das decodierte Bit erkannt werden kann. Das vollständige Ablaufdiagramm ist in Bild 2.12 dargestellt.

Zu Beginn wird eine Run-In-Sequenz übertragen. Eine genaue Beschreibung des verwendeten Übertragungsprotokolls ist in Kapitel 1.3.4 nachzulesen. Die Zeiten der einzelnen Datenbits sind variabel gehalten, die des Run-In-Blocks nicht. Dieser ist bei Raumtemperatur auf 900 μs festgelegt, schwankt aber je nach Temperatur um $\pm 15\%$.

Bei Betrachtung der Hardware des Sensors wird der Grund der Temperaturschwankung deutlich. Der Takt im Mikrocontroller wird durch ein R-C-Glied geregelt. Bei Veränderung der Temperatur verändert sich der Widerstand des R-C-Gliedes. Steigt die Temperatur, steigt der Widerstand und die Ladezeit des Kondensators verlängert sich. Somit sinkt die Frequenz des Mikrocontrollers und die Breite des Run-In-Blocks verlängert sich. Im Gegensatz dazu verringert sich bei fallender Temperatur der Widerstand und die Ladezeit verkürzt sich. Daraus resultiert eine höhere Frequenz und eine Verkürzung der Übertragungszeit.

Um daraus resultierenden Fehlern entgegenzuwirken, ist der Temperaturbereich auf -15°C bis +60°C festgelegt. Um die maximale und minimale Dauer des Run-In-Blocks zu ermitteln, wurde ein Sensor im besagten Temperaturbereich ausgelesen. Die daraus resultierenden Werte sind in Tabelle 2.2 abzulesen. Eine graphische Darstellung der Messungen ist in Anhang C.1 zu finden. Aus diesen wird die prozentuale Differenz zur Referenzmessung bei 23°C berechnet. Diese wird vom eingestellten CPU-Takt addiert oder subtrahiert um den aktuellen CPU-Takt zu erhalten. Dieser Wert soll nur als Anhaltspunkt dienen.

Temperatur	**Blocklänge**	**Differenz**	**Errechneter CPU-Takt**
-15°C	788 μs	-12,44%	2,25 MHz
23°C	900 μs	0%	2 MHz
60°C	1024 μs	13,78%	1,72 MHz

Tabelle 2.2.: Zeitmessung des Run-In-Blocks

Ist die Run-In-Sequenz korrekt empfangen, wird der eigentliche Datenempfang fortgesetzt. Hier wird ebenfalls die Zeit zwischen zwei aufeinander folgenden Ereignissen gemessen und mit den aus der Run-In-Sequenz ermittelten Werten verglichen. Entspricht die

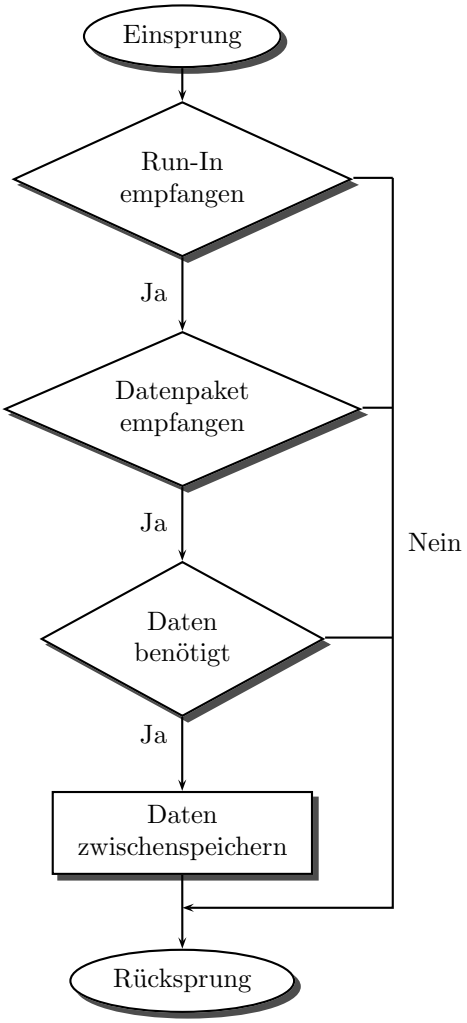
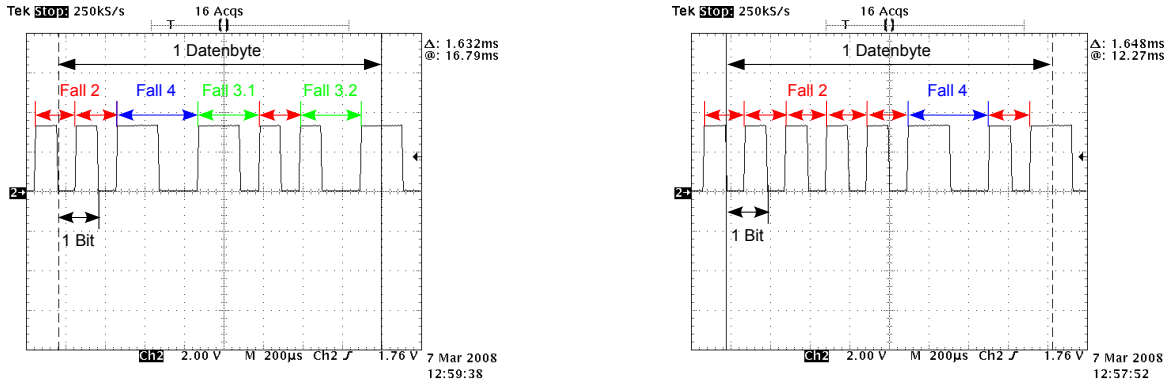


Bild 2.12.: Ablaufdiagramm des Timerinterrupthandlers

Dauer nicht der maximalen oder minimalen Zeit, wird der Interrupt verlassen und der Datenempfang startet wieder mit der Run-In-Sequenz. Entspricht die Zeit der erwarteten, sind drei Fälle zu betrachten. Diese insgesammt fünf Fälle werden anhand von zwei Codeausschnitten erläutert (Bild 2.13).

Die fünf Fälle beschreiben fünf verschiedene Bitlängen oder eher fünf verschiedene Zeiten zwischen zwei Ereignissen. Da bei Manchestercodierung (Kapitel 1.3.3) ein Bit in eine Bitfolge von zwei Bit zerspalten wird, sind drei verschiedene Zeiten für eine gültige Bitfolge möglich (Bild 2.13). Wird eine dieser Bitfolgen erkannt, wird das Bit im Zwischenspeicher abgelegt. Die zwei übrigen Fälle beschreiben einen Fehler und führen zum Abbruch des Empfangsprozesses.

Die Referenzzeit wird in der Run-In-Sequenz berechnet. Daraus werden die einzelnen Zeiten abgeleitet (Tabelle 2.3). Die Zeiten und die Bezeichnungen beziehen sich auf ein decodiertes Bit.



(a) Datenbyte mit der Wertigkeit 46

(b) Datenbyte mit der Wertigkeit 4

Bild 2.13.: Auszüge aus einem Datenpaket mit Markierung der unterschiedlichen Übertragungsfälle

Bezeichnung	**Länge**
Referenzzeit	x
minimale Bitlänge	$x - 50\ \mu s$
maximale Bitlänge	$x + 50\ \mu s$
minimale doppelte Bitlänge	$2 * x - 50\ \mu s$
maximale doppelte Bitlänge	$2 * x + 50\ \mu s$

Tabelle 2.3.: Berechnung der Referenzzeiten für den Datenempfang

Fall 1: Hierbei ist die aktuell gemessene Zeit kürzer als die für die minimale Bitlänge errechnete Zeit. Tritt dieser Fall ein, wird der Empfangsprozess abgebrochen und der Datenempfang beginnt erneut mit dem Empfang der Run-In-Sequenz.

Fall 2: Unterschreitet die gemessene Zeit die maximale Bitlänge, entspricht das decodierte Bit dem Wert des vorigen decodierten Bit. Dieser Fall ist mit Fall 2 in Bild 2.13.a und 2.13.b markiert.

Fall 3: Unterschreitet die gemessene Zeit die minimale doppelte Bitlänge, ist das vorige Bit zu betrachten. Entspricht das vorige Bit einer 0 ist eine „0“ erkannt (Fall 3.1). Entspricht es einer 1 ist eine Bitfolge von „10“ erkannt worden (Fall 3.2). Beide Fälle sind in Bild 2.13.a markiert.

Fall 4: Unterschreitet die gemessene Zeit die maximale doppelte Bitlänge, enspricht die Zeit zwischen zwei Interrupts der Zeit von zwei decodierten oder vier Bit in Manchestercodierung. Tritt dieser Fall ein, ist eine Bitfolge von „10“ erkannt. Dieser Fall ist mit Fall 4 in Bild 2.13.a und 2.13.b markiert.

Fall 5: Hierbei ist die aktuell gemessene Zeit länger als die maximale doppelte Bitlänge. Tritt dieser Fall ein, wird ebenfalls der Empfangsprozess abgebrochen und es wird

mit dem Empfang der Run-In-Sequenz fortgesetzt.

Das einzige Problem bei dieser Abarbeitung ist, das kein definiertes Ende vorgesehen ist. Das Datenpaket endet mit dem CRC-Byte und nicht mit einer definierten Bitfolge. Entspricht das CRC-Byte einem ungeraden Wert, kann dieses so nicht empfangen werden. Die Begründung liegt bei der Erkennung der einzelnen Bits. Da bei einem ungeraden Byte das letzte Bit die Wertigkeit 1 besitzt, endet das Datenpaket mit einer negativen Flanke. Da diese vom Mikrocontroller nicht ausgewertet werden, wird das gesamte Datenpaket verworfen. Um dieses Problem zu beheben Bedarf es eines kleinen Tricks. Wird bei der Übertragung eine Zeit ermittelt, die Fall 5 entspricht, wird überprüft ob die Erkennung des letzten Bits ansteht. Ist dies der Fall, entspricht das letzte Bit einer 1. Im normalen Betrieb konnten mit dieser Ausführung keine Fehler festgestellt werden.

Ist das Datenpaket vollständig empfangen (88 Bit), wird überprüft ob, die Daten korrekt empfangen wurden. Dazu wird eine CRC-Überprüfung auf alle 11 Byte angewendet (Kapitel 1.3.4). Des Weiteren wird überprüft, ob die empfangenen Sensordaten in diesem Speicherungsintervall schon einmal empfangen wurden. Ist dies der Fall oder die CRC-Überprüfung fehlerhaft, werden die Daten verworfen und der Empfangsprozess beginnt mit der Run-In-Sequenz. Werden die Daten benötigt und die CRC-Überprüfung ist fehlerfrei, werden die Daten zwischengespeichert, ein Statusbit gesetzt und der Empfangsprozess wird mit dem Empfang der Run-In-Sequenz fortgesetzt.

Benutzeroberfläche

Der Benutzer kann das Steuergerät über die serielle Schnittstelle oder über die Tasten auf dem Entwicklungsboard steuern. Die Steuerung über die Tasten erlaubt keine Konfiguration des Steuergerätes. Diese muss vorab über die serielle Schnittstelle erfolgen, kann aber im FLASH-Speicher gesichert werden.

Bild 2.14 zeigt das gesamte Ablaufdiagramm der Hauptschleife. In dieser werden hauptsächlich Statusbits abgefragt. Diese werden in anderen Prozessen, wie zum Beispiel dem Interrupt der seriellen Schnittstelle, gesetzt. Bei dieser Methode wird nur bei einem wirklich auftretenden Fall der jeweilige Prozess durchlaufen. So ist es möglich, weitere Funktionen zu implementieren ohne die gesamte Struktur des Hauptprogramms zu ändern.

Bei jedem Durchlauf der Hautpschleife wird die Uhrzeit aktualisiert. Diese wird normalerweise in der untersten Reihe des LCD Displays angezeigt. Befindet sich der User in einem Untermenü, wird die Uhrzeit nur intern weitergesetzt und nach Verlassen des Menüs aktualisiert. Die Uhrfunktion ist in der dritten Gruppe implementiert.

Befindet sich der User in einem Untermenü und verläßt dieses, ist es notwendig, die Darstellung auf dem LCD zurückzusetzen. Dazu wird in den jeweiligen Untermenüfunktionen ein Statusbit gesetzt. Dieses wird in der Hauptschleife abgefragt und die Darstellung des jeweiligen Untermenüs wird gelöscht und durch die Standarddarstellung ersetzt.

Nachdem die Darstellung auf dem LCD angepasst wurde, werden die Steuerbefehle ausgewertet. Zuerst werden die Tastenzustände abgefragt. Wurde eine Tasten gedrückt, wird die jeweilige Aktion ausgeführt. Je nach Menü kann eine Taste verschiedene Aktionen hervorrufen. Die Erkennung erfolgt in einem Timerinterrupt. Hier werden die Zustände der Tasten in Intervallen von 60 μs abgefragt. Entspricht der Zustand einer Taste in fünf

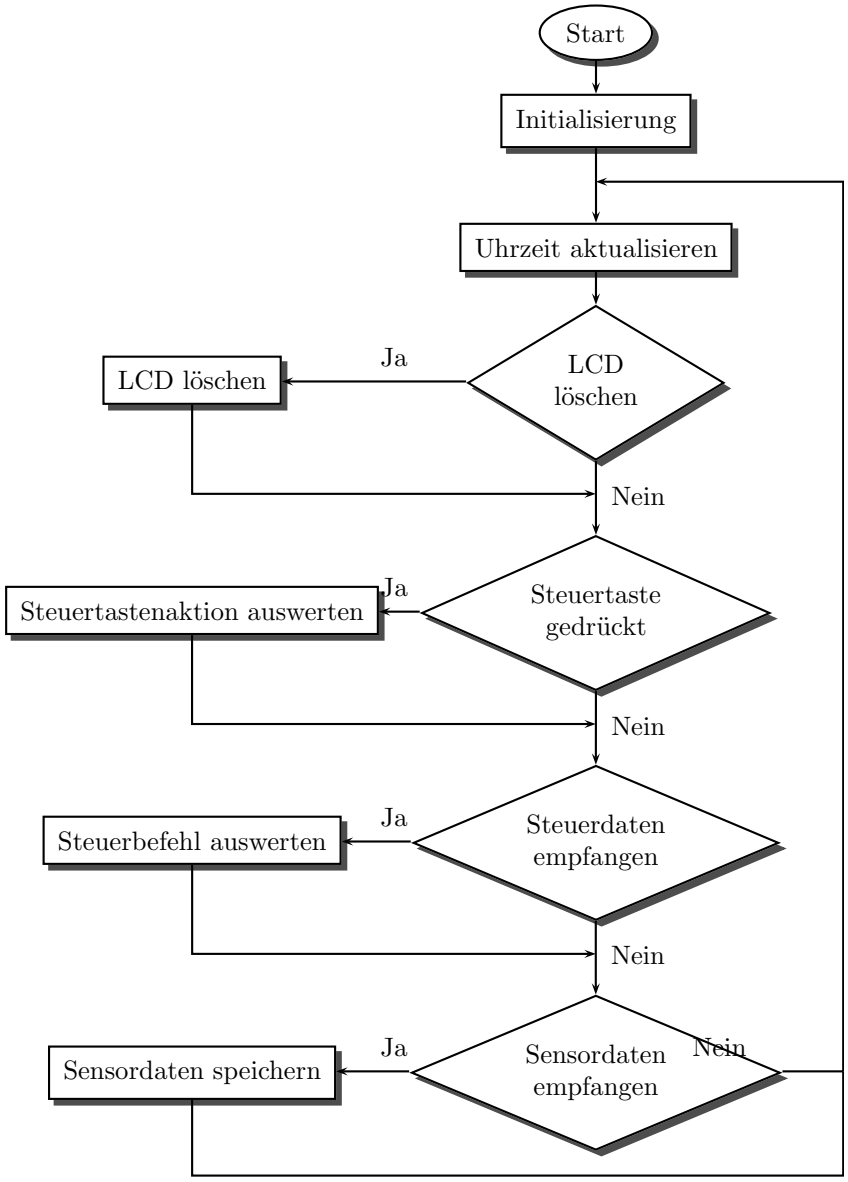


Bild 2.14.: Ablaufdiagramm der Hauptschleife

folgenden Intervallen einem *LOW* Zustand (Taste gedrückt), wird das Statusbit gesetzt. Wird die Taste daraufhin wieder losgelassen, wird das Statusbit nicht zurückgesetzt. Dies geschieht nur in der Hauptschleife nachdem die Tastenaktion ausgewertet wurde. Nötig ist das Einlesen über den Interrupt um die einzelnen Tasten zu entprellen.

Nach der Auswertung der Tastenaktion erfolgt die Auswertung der Steuerbefehle die über die serielle Schnittstelle empfangen wurden. Der Ablauf des Interrupts ist in Bild 2.15 dargestellt.

Im Interrupt werden die empfangen Zeichen ausgewertet. Bei einem *NEWLINE* (Enter), werden die Daten an die Hauptschleife übergeben und das Statusbit gesetzt. Wird

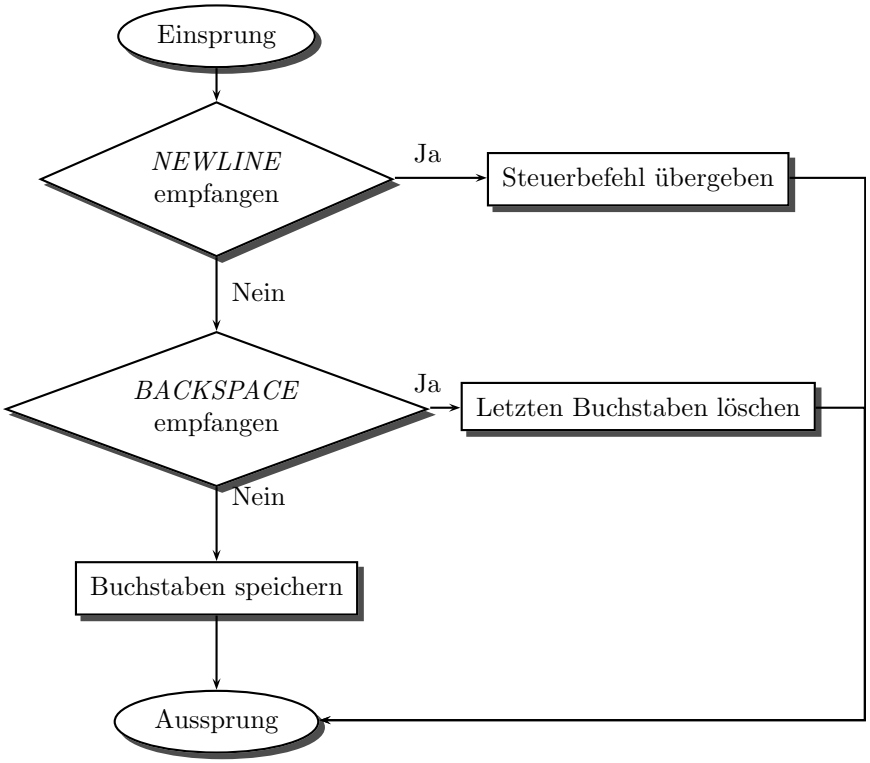


Bild 2.15.: Ablaufdiagramm des Schnittstelleninterrupts

ein *BACKSPACE* (löschen) empfangen, wird das letzte empfangene Zeichen gelöscht. Tritt keiner der beiden Fälle ein, wird das Zeichen zwischengespeichert und der Interrupt verlassen.

Sensordatenspeicherung

Der letzte Punkt der Hauptschleife betrifft die Datenspeicherung. Ist das Statusbit nach erfolgreichem Empfang eines Datenpakets gesetzt, werden die Daten im FLASH-Speicher gespeichert. Die Lese- und Schreibsoftware ist von Tobias Krannich im Laufe seiner Diplomarbeit entwickelt und getestet worden. Manche Funktionen wurden verändert oder sind neu hinzugefügt worden. Im Quellcode (Kapitel B.1.2) des Moduls sind die Funktionen von Herrn Krannich kenntlich gemacht worden.

Um die maximale Betriebszeit zu ermitteln, wurde vorab eine Berechnung durchgeführt. Hier wurde errechnet, wie lange die Speicherkapazität bei leerem Speicher und verschiedener Anzahl von Sensoren ausreicht.

Die Speicherkapazität beträgt $(8\ k + 256)\ Byte * 1024 \approx 8\ MB$. Für die Steuerdaten ist eine *half page* im Speicher reserviert. Eine *half page* entspricht einer Kapazität von 256 Byte [22]. Berechnet man daraus die zur Verfügung stehenden Byte, erhält man:

$$(8\ k + 256)\ Byte * 1024 - 256\ Byte = 8.453.888\ Byte \tag{2.6}$$

Die Daten werden mit einer Zeitmarke versehen und im folgenden Format abgespeichert:

TUE 01-Mar-2008 00:00:00 01 00 00 00 00 00 00 00

Die Zeitmarke ist aus dem englischen Sprachraum übernommen und beinhaltet den Wochentag (3 Byte), das vollständige Datum (11 Byte) und die Uhrzeit (8 Byte). Darauf folgen die eigentlichen Daten mit jeweils 2 Byte in hexadezimaler Darstellung. Beginnend mit der Sensoradresse (2 Byte), dem Temperaturwert (4 Byte), dem Versorgungsspannungswert (4 Byte), dem Zellenspannungswert (4 Byte) und dem CRC-Byte (2 Byte). Zwischen jedem Wert wird ein Leerzeichen (10 Byte) und am Ende ein *NEWLINE* und *CARRIAGE RETURN* (2 Byte) gesetzt. Addiert man alles zusammen erhält man:

$$(3 + 11 + 8 + 2 + 4 + 4 + 4 + 2 + 10 + 2)\ Byte = 50\ Byte \tag{2.7}$$

Die Daten werden in einem Intervall von 60 Sekunden gespeichert. Das heißt jede Minute wird ein Datenstring pro Sensor gespeichert. Als Beispiel mit 12 Sensoren sind das 12*50 Byte pro Minute. Bei Flurförderzeugen ist die Arbeitszeit in Betriebsstunden oder -tagen anzugeben. Daher wird der Wert in Byte pro Stunde errechnet:

$$12\ Sensoren * 50\ Byte * 60\ Minuten = 36000\ Byte/Stunde \tag{2.8}$$

Teilt man die zur Verfügung stehende Speicherkapazität mit den benötigten Byte pro Stunde, erhält man die Zeit, in der das System Daten speichern kann ohne alte Werte zu verlieren.

$$\frac{8453888\ Byte}{36000\ Byte/Stunde} = 234\ Stunden\ 49\ Minuten \approx 9\frac{1}{2}\ Tage \tag{2.9}$$

Das Steuergerät kann so konfiguriert werden, dass entweder 6, 12, 24 oder 40 Sensoren abgefragt werden. In Tabelle 2.4 ist für alle vier Fälle die maximale Arbeitszeit angegeben.

Anzahl der Sensoren	**Arbeitszeit in Stunden**
6	469
12	234
24	117
40	70

Tabelle 2.4.: Maximale Arbeitszeit bei verschiedener Anzahl von Sensoren

Die Uhrzeit wird im Interrupthandler des Watchdog Timers berechnet. Somit stehen die restlichen Interrupts zur freien Verfügung. Das Intervall ist auf eine Sekunde eingestellt. Bild 2.16 zeigt den genauen Ablauf des Interrupts.

Bei jedem Aufruf werden als erstes die Uhrzeit und das Datum berechnet. Bei der Datumsberechnung werden Schaltjahre und die verschiedenen Tage eines Monats berücksichtigt. Ist eine Minute abgelaufen, wird überprüft, ob Daten von allen Sensoren gespeichert wurden. Ist dies nicht der Fall, wird der letzte empfangene Wert des fehlenden Sensors aus dem Zwischenspeicher in den FLASH geschrieben. Dadurch soll garantiert werden,

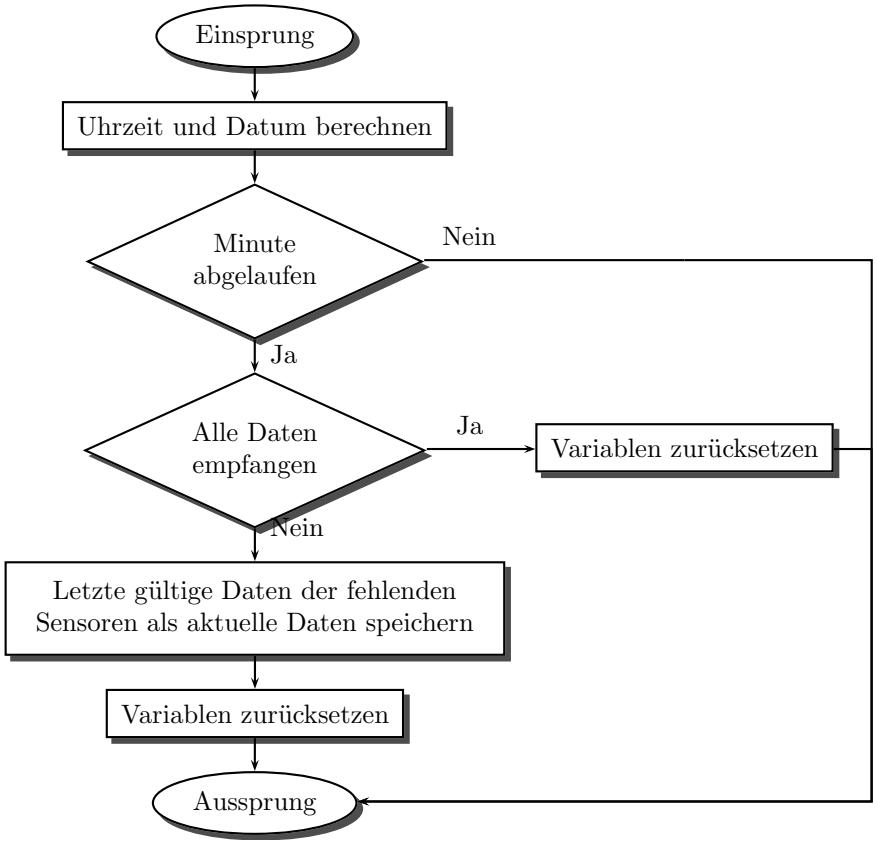


Bild 2.16.: Ablaufdiagramm des Watchdog Interrupthandlers

dass für jeden Sensor die gleiche Anzahl von Messwerten zur Verfügung steht. Dies ist nur für die Auswertung wichtig, eine genaue Aussage über den Zustand der Zelle kann damit nicht getroffen werden. Zu erkennen ist so ein Fehler in der Auswertung sehr einfach: Da immer der gleiche Messwert gespeichert wird, ist die Funktion in dem Bereich eine Gerade. Tritt dieser Fall ein, sind die Sensoren und die dazugehörigen Zellen zu überprüfen und gegebenfalls auszutauschen.

Sind alle Daten gespeichert oder durch Werte aus dem Zwischenspeicher ersetzt, werden die Variablen, die zur Kontrolle der gespeicherten Messwerte dienen, zurückgesetzt und der Interrupt wird verlassen.

2.2.5. Bedienung des Steuergerätes

Wie im vorigen Kapitel beschrieben ist die Bedienung des Steuergerätes über die serielle Schnittstelle und die drei auf den Entwicklungsboard aufgebrachten Tasten möglich.

Bedienung per Tasten

Die Tasten erlauben nur eine eingeschränkte Bedienung. Genau genommen ist es nur möglich, die aktuelle Messung zu starten/stoppen, Konfigurationen zu laden/speichern, empfangene Sensoren anzuzeigen und Uhrzeit/Datum einzustellen. In Bild 2.17 sind die verschiedenen Tastenaktionen aus unterschiedlichen Menüs dargestellt.

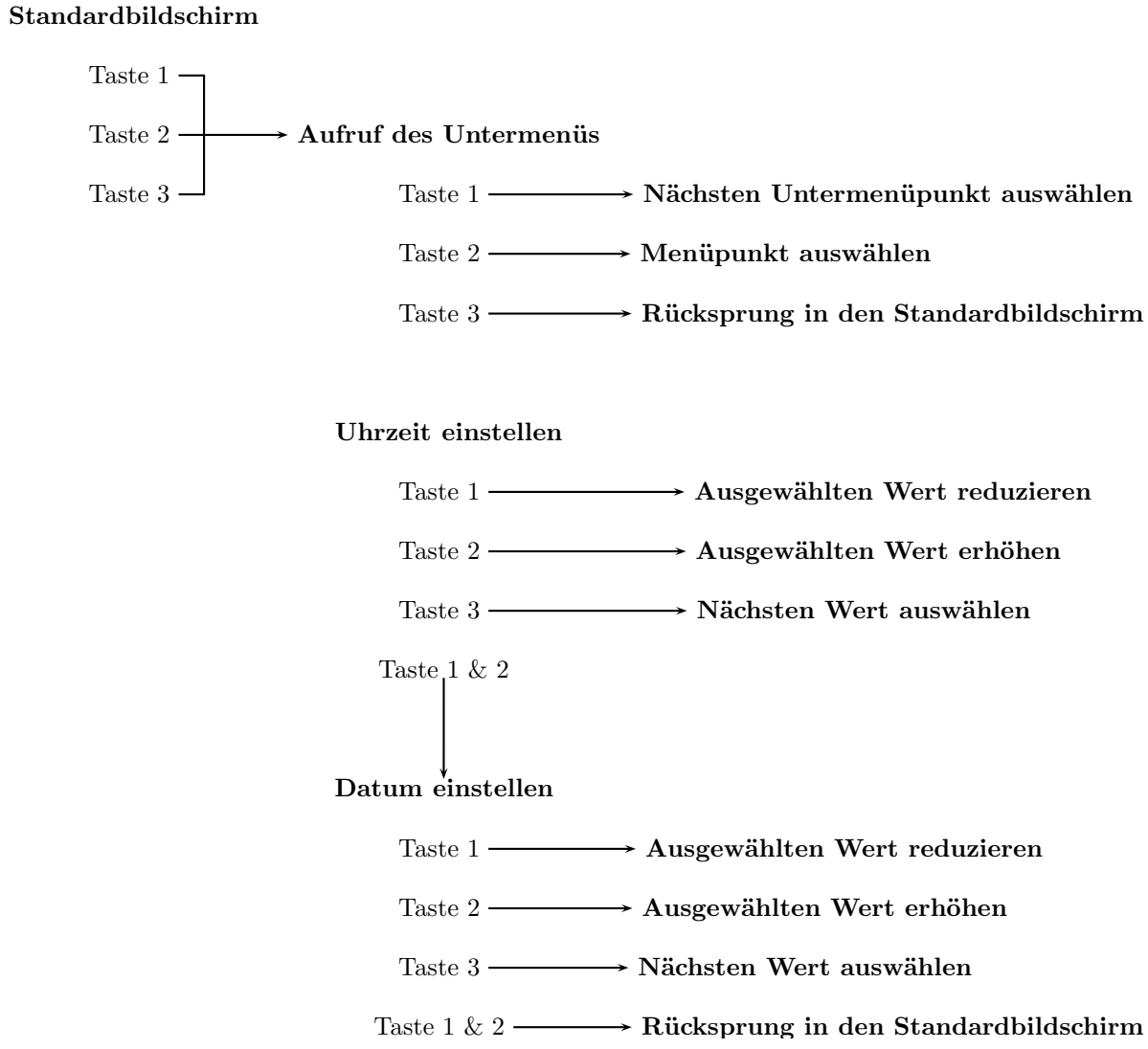


Bild 2.17.: Bedienung des Steuergeräts über die Steuertasten

Die Tasten des Steuergerätes sind in Bild 2.11 markiert. Der Standardbildschirm entspricht der Darstellung wie sie in Bild 2.11 zu sehen ist. Aus diesem Bildschirm gibt es zwei verschiedene Untermenüs die aufgerufen werden können. Bei Betätigung nur einer Taste wird das Menü zur Steuerung der Messungen aufgerufen. Hier gibt es sechs verschiedene Menüpunkte. Die Tastenaktionen sind im ersten Baum von Bild 2.17 dargestellt.

Punkt 1: Darstellung des aktuellen Sensorstatus Sind die Daten eines Sensors im aktuellen Speicherungsintervall bereits empfangen, enspricht die Darstellung einem „ * “, sind sie noch nicht empfangen einem „ . “. Der Bildschirm erscheint fünf Sekunden lang, bevor automatisch in den Standardbildschirm zurückgekehrt wird.

Punkt 2: Starten der Messung Aus diesem Menüpunkt heraus ist es möglich, die Messwertaufnahme zu starten. Zur Sicherheit wird bei Punkt 2 bis 5 eine Bestätigung der gewählten Aktion angefordert.

Punkt 3: Stoppen der Messung Aus diesem Menüpunkt heraus ist es möglich, die Messwertaufnahme zu stoppen.

Punkt 4: Konfiguration laden Hiermit ist es möglich, die gespeicherte Konfiguration zu laden.

Punkt 5: Konfiguration speichern Die Einstellungen, die über die serielle Schnittstelle erfolgten, können aus diesem Punkt heraus gespeichert werden. Dazu gehören die Anzahl und Adressen der Sensoren und die Position der Daten im FLASH-Speicher. Wird im laufenden Betrieb eine neue Messung gestartet, ist es sinnvoll die Konfiguration zu speichern um die aktuelle Position im FLASH-Speicher zu erhalten. Wird dies nicht durchgeführt, werden alle Daten der vorigen Messung ebenfalls abgerufen.

Punkt 6: Uhrzeit/Datum einstellen Um korrekte Messergebnisse zu erhalten, ist es notwendig die richtige Uhrzeit beziehungsweise das richtige Datum einzustellen. Bei Auswahl dieses Menüpunktes wird die Einstellung von Uhrzeit/Datum aufgerufen. Die Funktionen der Tasten sind in Bild 2.17 im zweiten Baum gezeigt. Nach Einsprung wird das Datum eingestellt. Dazu wird der Tag, der Monat und das Jahr einzeln verändert. Die zu verändernde Stelle blinkt im Sekundentakt. Betätigt man Taste 2, wird der aktuelle Wert erhöht, mit Taste 1 wird er reduziert. Taste 3 wählt die nächste Stelle aus. Um die Einstellung zu beenden, ist es notwendig, Taste 1 und 2 gleichzeitig zu drücken. Die Datumseinstellung läuft nach dem gleichen Schema ab. Die Aktionen jeder Taste bleiben die selben. Bei gleichzeitigem Drücken der Tasten 1 und 2, wird in den Standardbildschirm zurückgesprungen und Uhrzeit/Datum gespeichert.

Bedienung über die serielle Schnittstelle

Die Bedienung über die serielle Schnittstelle ist sehr komfortabel. Aus dem Terminal heraus können die Befehle eingegeben werden. Die Einstellungen für die Schnittstelle lauten:

Übertragungsgeschwindigkeit: 115200 bps
Datenbits: 8
Stoppbits: 1
Parität: keine
Flowcontrol: keine

Jede Eingabe kann in Groß- oder Kleinschreibung erfolgen. Bei einem korrekt erkannten Befehl wird eine Hilfe, die die genaue Syntax für den jeweiligen Befehl enthält, gezeigt. Bild 2.5 zeigt alle Steuerbefehle. Zu jedem Steuerbefehl gibt es noch eine Kurzform, die ein schnelleres Arbeiten ermöglicht.

Steuerbefehl	**Kurzbefehl**	**Aktion**
help	he	Darstellen der Hilfe
realtime	rtc	Einstellen der Uhr/Datum
sensors	se	Einstellungen der Sensoren
memory	mem	Gespeicherte Daten abrufen
clear	clr	FLASH komplett löschen
start	st	Messung starten
stop	stp	Messung stoppen
read	rd	Steuergerätedaten speichern
write	wr	Steuergerätedaten laden
exit	ex	Verlassen der Funktion

Tabelle 2.5.: Auflistung der Steuerbefehle und deren Aktionen

Im vorigen Kapitel wurden schon die meisten Erklärungen geliefert. Daher wird hier nur auf neue Menüpunkte eingegangen.

help Wird dieser Befehl an das Steuergerät gesendet, wird die Hilfe, die alle Steuerbefehle und Anweisungen enthält, eingeblendet.

sensors Mit diesem Steuerbefehl wird die Konfiguration der einzelnen Sensoren vorgenommen. Zuerst muss die Anzahl der Sensoren, daraufhin jede Sensoradresse eingegeben werden. Die Anzahl und jede Adresse muss mit Enter bestätigt werden.

memory Hierbei wird die Übertragung der gespeicherten Messwerte gestartet. Um die Messwerte zu speichern gibt es zwei Möglichkeiten. Zum einen ist es möglich die Daten mit der MATLAB Software zu empfangen. Zum anderen kann die Speicherung aller empfangenen Daten im Terminal aktiviert werden. Diese Möglichkeit wird empfohlen da die Fehlerquote in MATLAB höher ist. Nachdem alle Daten empfangen wurden ist am Steuergerät der Empfang über die Steuertasten zu bestätigen.

clear Sollen Messungen mit anderen Sensoren durchgeführt werden ist es ratsam vorher den FLASH-Speicher komplett zu löschen. Hierbei werden auch die Steuerdaten gelöscht, was einem Neustart entspricht.

exit Sollte bei der Eingabe ein Fehler gemacht worden sein oder ist es doch nicht nötig eine Änderung vorzunehmen, kann die jeweilige Funktion jederzeit mit *exit* verlassen werden.

3. Versuchsbetrieb

Die Messungen wurden an einem handelsüblichen Gabelstapler der Firma Still mit der Typenbezichnung EFG 1,0/5001 durchgeführt. Die installierte Antriebsbatterie ist von der Firma Hoppecke und trägt die Bezeichnung 5PzS55L mit einer Gesamtspannung von 24 V und einer Kapazität von 550 Ah. Die Versuche wurden im Fahrzeuglabor der HAW Hamburg im Department Fahrzeugtechnik mit freundlicher Unterstützung von Herrn Bernd Schröder durchgeführt. Bevor die Messungen durchgeführt werden konnten mussten Wartungsarbeiten erfolgen. Es wurde der Stecker und die Kupplung, die zur Stromversorgung des Gabelstaplers dienen, ausgewechselt.

Bevor der Versuchsaufbau erläutert wird, wird die MATLAB Software erläutert. Dazu gehört die Software zur Darstellung der Messwerte und die zur Kalibrierung der Temperaturwerte jedes einzelnen Sensors.

3.1. Verarbeitung der Messwerte in MATLAB

Die Verarbeitung der Messwerte erfogt in MATLAB. Die verwendete MATLAB Version und weitere Informationen über MATLAB sind in Kapitel A.2 nachzulesen. Bevor die Messwerte dargestellt werden können, ist es notwendig die Temperaturmesswerte zu kalibrieren.

3.1.1. Kalibrierung der Temperaturmesswerte

Eine Kalibrierung ist notwendig da der Offset Fehler bei jedem Mikrocontroller der MSP Familie anders ausgeprägt ist. Um den Fehler zu bestimmen, ist eine Messreihe aufgenommen worden. Die Messreihe umfasst die Aufnahme von Messwerten im Temperaturbereich von 10°C bis 60°C in 5°C Schritten. Dazu werden alle Sensoren an eine Batterie angeschlossen und in einen Thermoschrank der Firma Heraeus Vötsch mit der Typenbezeichnung VMT 04/35 eingeschlossen. Die komplette Messreihe ist in Anhang E.1 dargestellt.

In Bild 3.1 ist die Kalibrierungsmessung eines Sensors abgebildet. Im oberen Fenster sind drei Temperaturkurven dargestellt. Die rote Kurve entspricht den vom Sensor empfangenen, unkalibrierten Werten, die grüne Kurve entspricht den am Thermoschrank eingestellten, realen Werten und die blaue Kurve den kalibrierten Werten.

Um die Kalibrierte Kurve zu berechnen wird zuerst eine Differenzfunktion errechnet. Dazu wird die reale Temperatur auf der X-Achse und die Differenz zwischen der realen und der gemessenen Temperatur auf der Y-Achse abgebildet. Die daraus entstandene Kurve wird mit einer Näherungsgleichung zweiten Grades beschrieben. Bei jedem gemessenen Wert wird die Differenz mit der Näherungsgleichung berechnet und vom selbigen

abgezogen. Die daraus entstandene Kurve entspricht der kalibrierten Temperaturkurve. Die Koeffizienten der Näherungsgleichung wie auch die Anzahl und Adresse der Sensoren werden in der Konfigurationsdatei (*sensorsystem.txt*) abgespeichert.

In Bild 3.1 ist ein Auszug der Darstellung der Kalibrierungsmessung zu sehen. Im mittleren Fenster ist die Versorgungsspannung der gesamten Schaltung dargestellt. Die Erhöhung von zirka 0,1 V ist auf die Erhöhung der Temperatur im Laufe der Messwertaufnahme zurückzuführen. Das letzte Fenster bildet die Versorgungsspannung des Sensor ab.

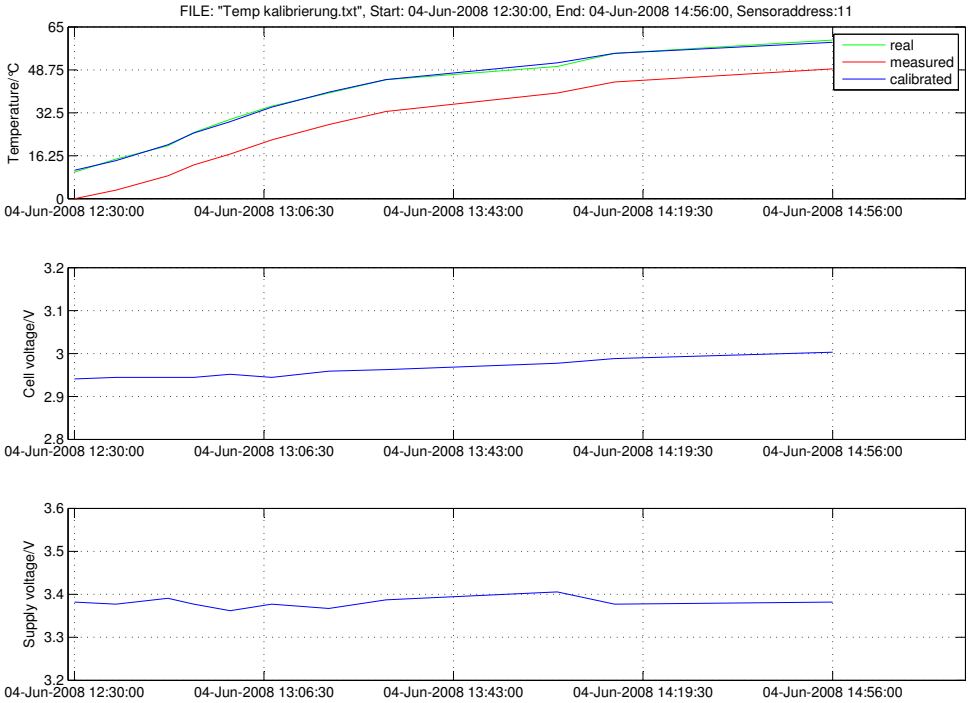


Bild 3.1.: Kalibrierungsmessung, Darstellung der realen (grün), der gemessenen (rot) und der kalibrierten Temperaturkurve (blau)

3.1.2. Darstellung der Messwerte

Die Software zur Darstellung der Messwerte benötigt eine Konfigurationsdatei, welche automatisch bei der Temperaturkalibrierung erstellt wird. Anhand dieser werden die Daten jedes Sensors sortiert und die Temperaturmesswerte kalibriert. Die Kalibrierung erfolgt erfolgt nach dem im vorigen Kapitel beschriebenen Schema. Sind in der Messung Sensoren verwendet worden, die nicht in der Konfigurationsdatei angegeben sind, bricht die Software automatisch ab und gibt eine Fehlermeldung aus. Alle benötigten M-Files sind im Anhang B.2.2 zu finden.

Gestartet wird die Software mit dem M-File *Diplom.m*. Abbidung 3.2 zeigt den Anfangsbildschirm. Als Datenquelle steht eine Textdatei zur Verfügung (Button mit der Beschriftung *File*). Diese kann mit Hilfe des Terminals erstellt werden. Ist kein Terminalprogramm zur Hand, ist es möglich, die Daten direkt vom Steuergerät in MATLAB zu laden (Button mit der Beschriftung *Comport*). Nach erfolgreicher Übertragung werden die Daten zusätzlich zur Darstellung in einer Textdatei abgespeichert. Diese Methode birgt aber noch kleinere Fehler, da der Umgang mit der seriellen Schnittstelle in MATLAB nicht ausgereift zu sein scheint. Teilweise werden Buchstaben oder auch das Ende der Übertragung nicht erkannt. Da der gesamte Prozess automatisch abläuft führt dies zum Abbruch.

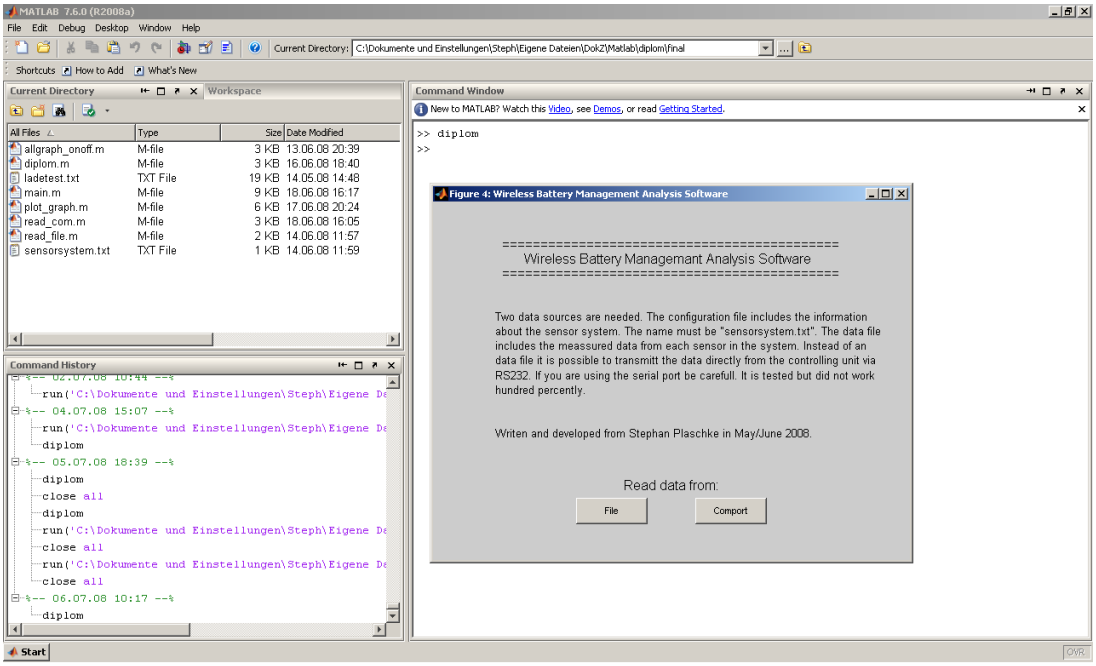


Bild 3.2.: Startbildschirm der MATLAB Software

Wird als Datenquelle eine Textdatei gewählt, erscheint ein Bildschirm mit der Aufforderung, die Textdatei auszuwählen. Hierbei ist der genaue Aufbau der Textdatei zu beachten. Es dürfen keine Leerzeilen enthalten sein. Ebenfalls ist die erste Zeile, in der die Addresse als 0 eingetragen ist, zu löschen. Nach erfolgreicher Auswahl startet die automatische Auswertung. Ist die Auswertung abgeschlossen, werden die Daten jedes Sensors in drei Fenstern abgebildet. Bild 3.3 zeigt alle drei Fenster. Im Vordergrund ist das Fenster mit der Versorgungsspannung dargestellt.

Die einzelnen Button am rechten Rand erlauben es, jede einzelne Kurve jedes Sensors aus- und wieder einzuschalten. Oberhalb der Grafik sind die Daten angegeben, die die Messung betreffen.

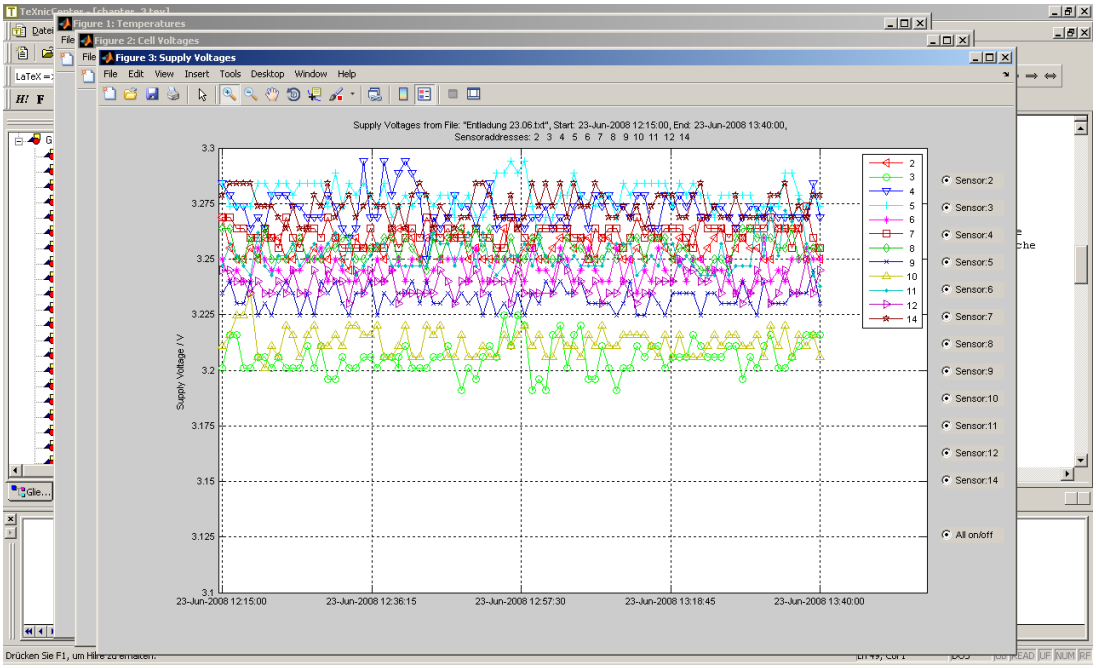


Bild 3.3.: Darstellung der drei Messwerte jeweils in einem eigenen Fenster

3.2. Versuchsdurchführung

Bei dem hier verwendeten Akkumulator handelt es sich um eine 24 V Antriebsbatterie. Das bedeutet es sind 12 Zellen verbaut und somit werden 12 Sensoren zur Erfassung des Zustandes jeder einzelnen Zelle benötigt. Die Sensoren wurden im Vorfeld im Temperaturbereich von 10°C bis 60°C ausgelesen und kalibriert (Kapitel 3.1.1).

Die Anordnung der Batteriezellen im Gehäuse ist in Bild 3.4 schematisch und in Bild 3.5 in Realität zu sehen.

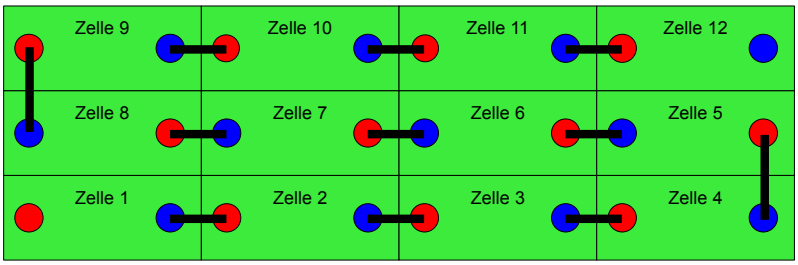


Bild 3.4.: Schematische Anordnung der einzelnen Zellen der Antriebsbatterie

Die Zellenverbinder sind mit Klemmschrauben befestigt. Die Sensoren werden auf den Plus- und Minuspol der einzelnen Zellen aufgesteckt (Bild 3.5). Dazu sind die Klemmschrauben mit einer 4mm Bohrung versehen worden. Diese beeinträchtigt die Festigkeit

der Klemmschraube nicht und es besteht keine Gefahr, dass diese nicht mehr ausreicht.

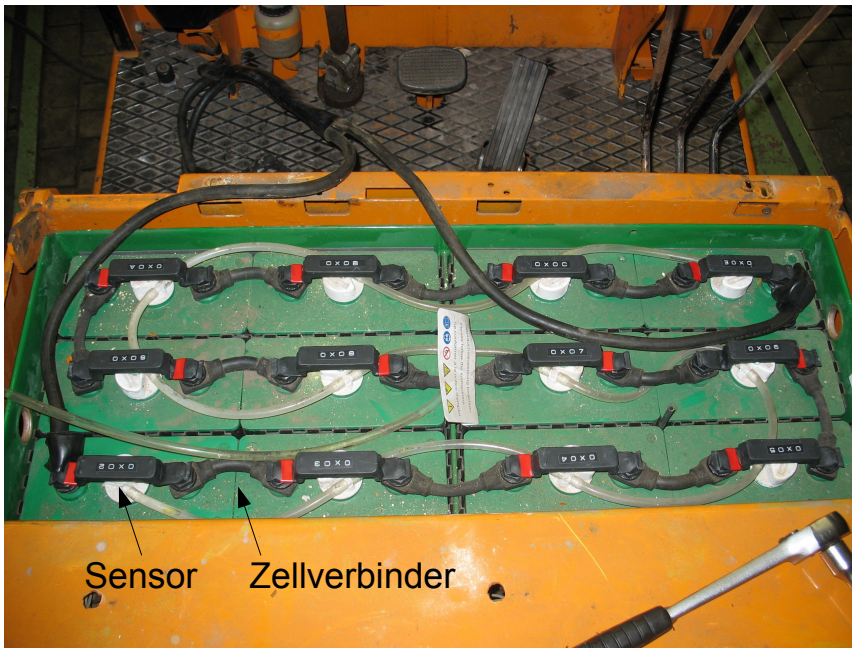


Bild 3.5.: Anordnung der einzelnen Zellen der Antriebsbatterie

Um weitreichende Informationen über die Batterie zu erhalten wurden verschiedene Messungen durchgeführt. Begonnen wurde mit einem Entladevorgang. Hier wird die Batterie über den Antriebsmotor vollständig entladen. Daraufhin wird die Batterie an ein handelsübliches Ladegerät angeschlossen und über mehrere Stunden geladen. Darauf folgte ein weiterer Entladevorgang und eine Messung im Ruhezustand. Mit diesen vier Messungen ist es möglich, eine aussagekräftige Beschreibung des Batteriezustands und der Funktion der Sensoren zu geben.

Bei den Entladungsvorgängen sind verschiedene Anordnungen der Sensoren vorgenommen worden. Dies soll beweisen, dass eventuell auftretende Phänomene nicht vom verwendeten Sensor abhängig sind. Die Anordnung der Sensoren wird vor der Darstellung der jeweiligen Messwerte erläutert. Des Weiteren sind noch andere Randbedingungen aufgenommen worden. Dazu gehört die Leerlaufspannung der Batterie, die Temperatur und die Uhrzeit zu Beginn und zum Ende einer Messung. Diese Werte wurden extern aufgenommen und sollen als direkter Vergleich zu den ermittelten Messwerten der Sensoren gelten.

3.3. Versuch 1: Entladung

Die erste Messung ist eine Entladung. Dazu wurde die Batterie im Vorfeld geladen. Zum entladen wird der Gabelstapler aufgebockt. So ist es über den Antriebsmotor möglich, eine konstante Entladung über den gesamten Verlauf zu erhalten. Die Sensoren sind nach

Tabelle 3.1 auf die Batteriezellen verteilt. Alle anderen Randbedingungen sind in Tabelle 3.2 zu finden.

Zelle	1	2	3	4	5	6	7	8	9	10	11	12
Sensoradresse	2	3	4	5	6	7	8	9	10	11	12	14

Tabelle 3.1.: Anordnung der Sensoren auf den Batteriezellen für die erste Messung

	Uhrzeit	**Temperatur**	**Leerlaufspannung**
Beginn der Messung	11:14 Uhr	21,2°C	23,37 V
Ende der Messung	12:17 Uhr	27,8°C	22,78 V

Tabelle 3.2.: Randbedingungen der ersten Messung

Auf den nachfolgenden Seiten sind die drei Messungen des ersten Versuchs dargestellt. Die erste Grafik beschreibt den Temperaturverlauf, die zweite die Zellenspannung und die dritte die Versorgungsspannung. Eine kurze Beschreibung des Temperatur- und des Zellspannungsverlaufs werden nach den Grafiken gegeben. Die Betrachtung der Versorgungsspannung erfolgt nur in Kapitel 3.7. Bei allen Versuchen wird dies ebenso gehandhabt.

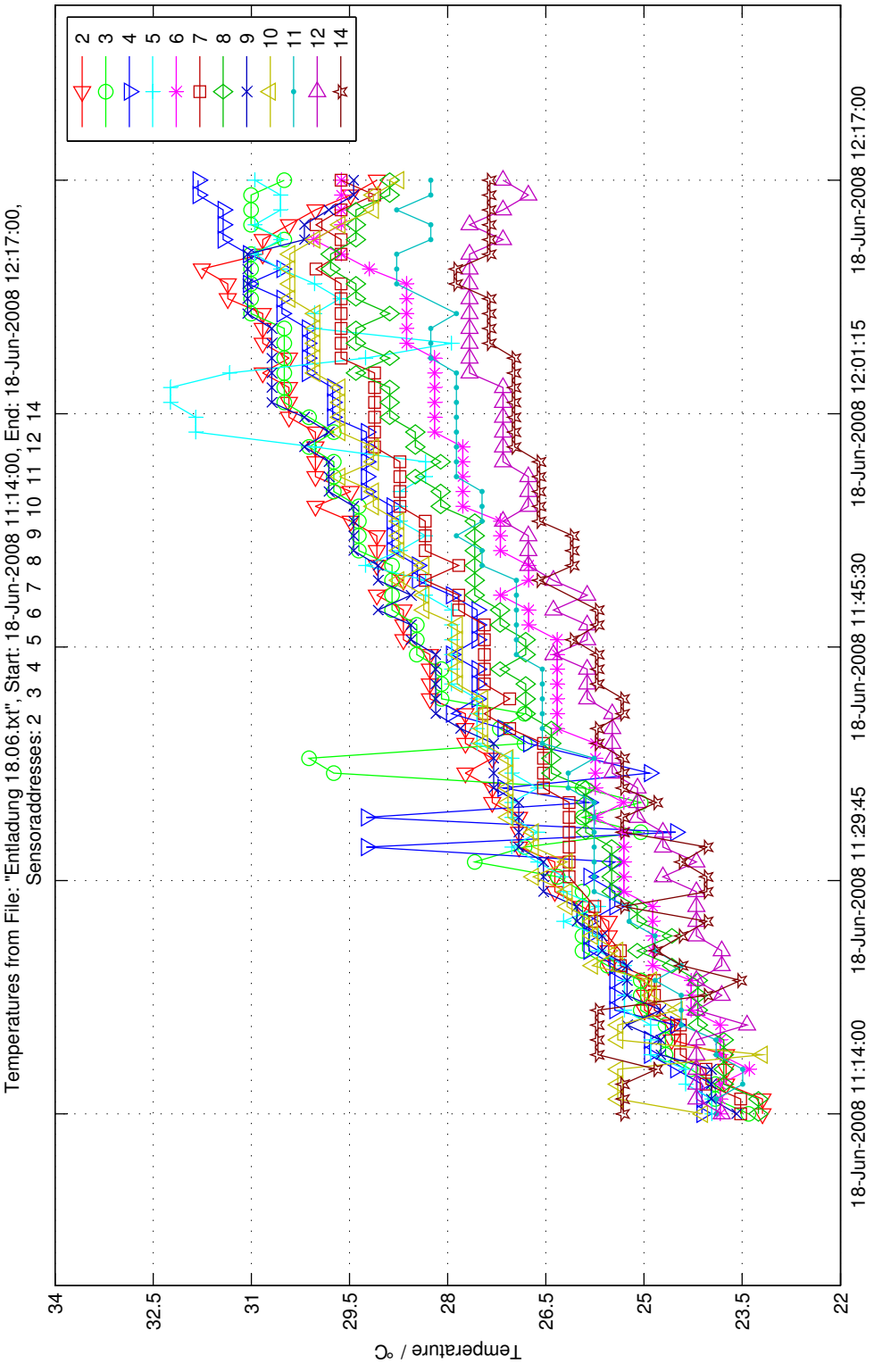


Bild 3.6.: Temperaturverlauf des ersten Entladevorgangs

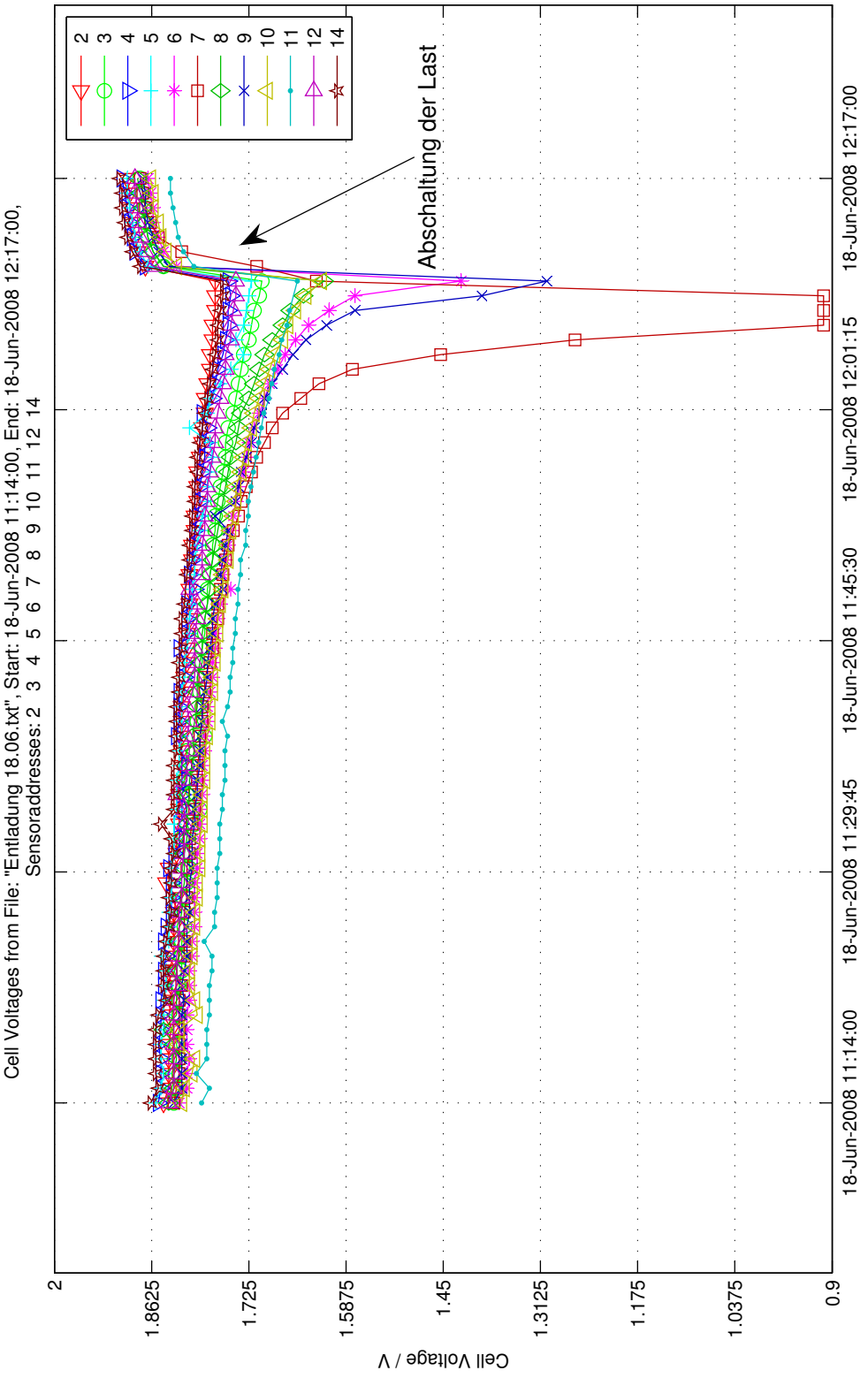


Bild 3.7.: Zellspannungsverlauf des ersten Entladevorgangs

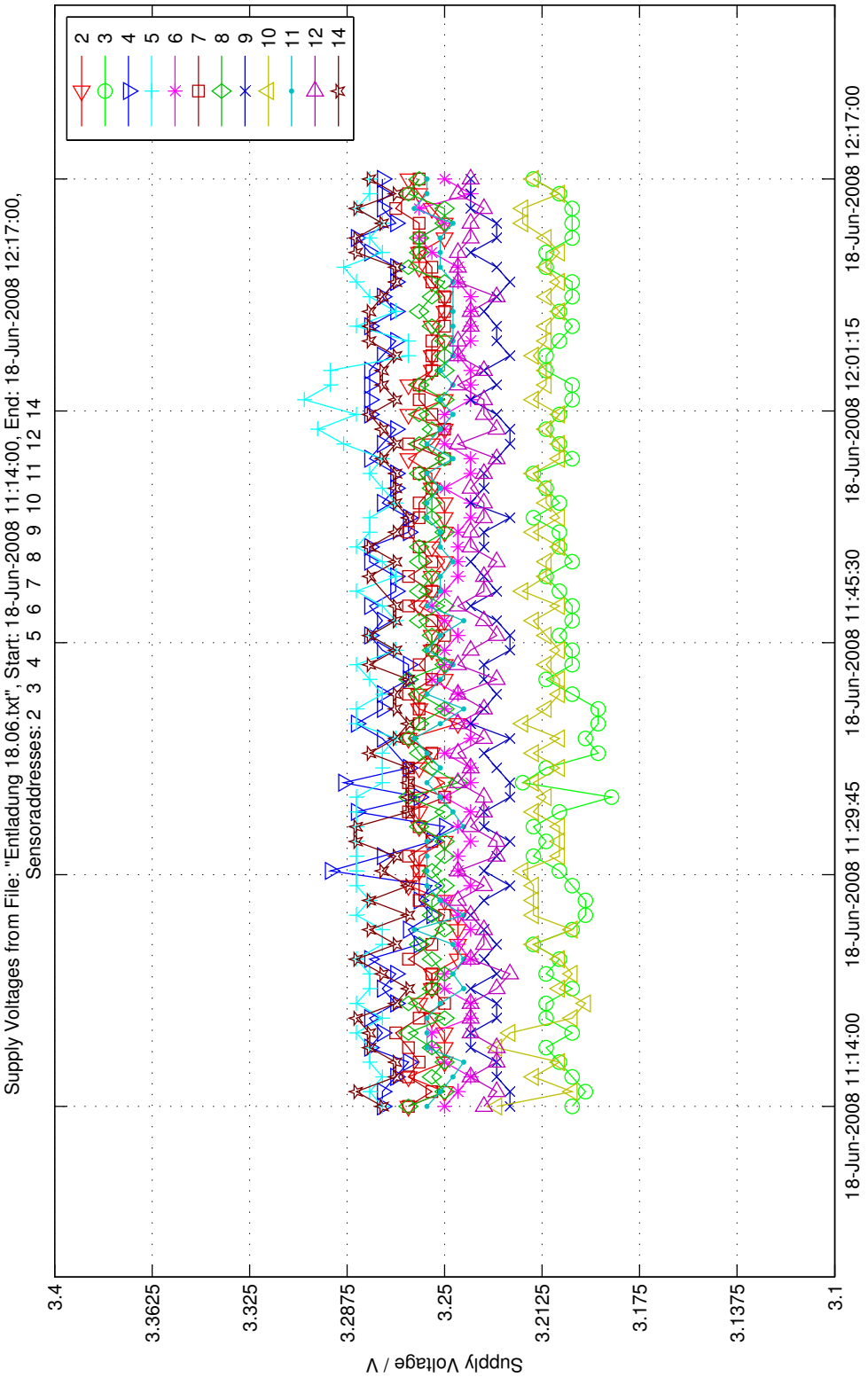


Bild 3.8.: Versorgungsspannungsverlauf des ersten Entladevorgangs

Bei der Temperaturmessung (Bild 3.6) ist an allen Sensoren die steigende Tendenz deutlich. Die Ausreißer der Sensoren mit der Adresse 3, 4 und 5 (Zelle 2, 3 und 4) sind auf Messfehler zurückzuführen. Die maximale Abweichung beträgt 4,2°C die minimale 0,45°C (Tabelle 3.3).

	Beginn	**Ende**
Referenzwert	21,2°C	27,8°C
Min. Abweichung	2°C	0,45°C
Max. Abweichung	4,1°C	4,2°C

Tabelle 3.3.: Temperaturabweichung des ersten Versuchs

Bei der Zellspannungsmessung (Bild 3.7) ist der Abschaltzeitpunkt der Last kenntlich gemacht. Diese Messung wurde im direkten Betrieb gestartet, daher ist kein Einschaltzeitpunkt erkennbar. Der Spannungsabfall an den Sensoren mit der Adresse 6, 7 und 9 (Zelle 5, 6 und 8) lässt einen Rückschluss auf den schlechten Zustand (SOH) der Zellen zu. Die Referenzmessung mit einem Messgerät ergibt annähernd gleiche Werte wie das Aufaddieren der Einzelspannungen (Tabelle 3.4).

	Beginn	**Ende**
Referenzwert	22,37 V	22,78 V
Aufaddierte Einzelspannungen	22,12 V	22,67 V

Tabelle 3.4.: Leerlaufspannung als Referenzwert und durch Aufaddieren der Einzelspannungen des ersten Versuchs

Die Versorgungsspannungsmessung sowie eine detailierte Betrachtung aller Ergebnisse erfolgt in Kapitel 3.7.

3.4. Versuch 2: Ladung

Diese Messung beschreibt eine Ladung nach vollständiger Entleerung der Batterie aus Versuch 1. Hierbei wird die Batterie an das Ladegerät angeschlossen. Das Ladegerät besteht aus einem Transformator und einer Zeituhr. An dieser kann die Dauer der Batterieladezeit eingestellt werden. Der Strom regelt sich über den internen Widerstand der Batterie von selbst. Tabelle 3.5 zeigt die Verteilung der Sensoren auf den Batteriezellen, Tabelle 3.6 die Randbedingungen.

Zelle	1	2	3	4	5	6	7	8	9	10	11	12
Sensoradresse	2	3	4	5	6	7	8	9	10	11	12	14

Tabelle 3.5.: Anordnung der Sensoren auf den Batteriezellen für die zweite Messung

	Uhrzeit	**Temperatur**	**Leerlaufspannung**
Beginn der Messung	12:27 Uhr	25,6°C	23,11 V
Ende der Messung	15:24 Uhr	24,2°C	24,94 V

Tabelle 3.6.: Randbedingungen der zweiten Messung

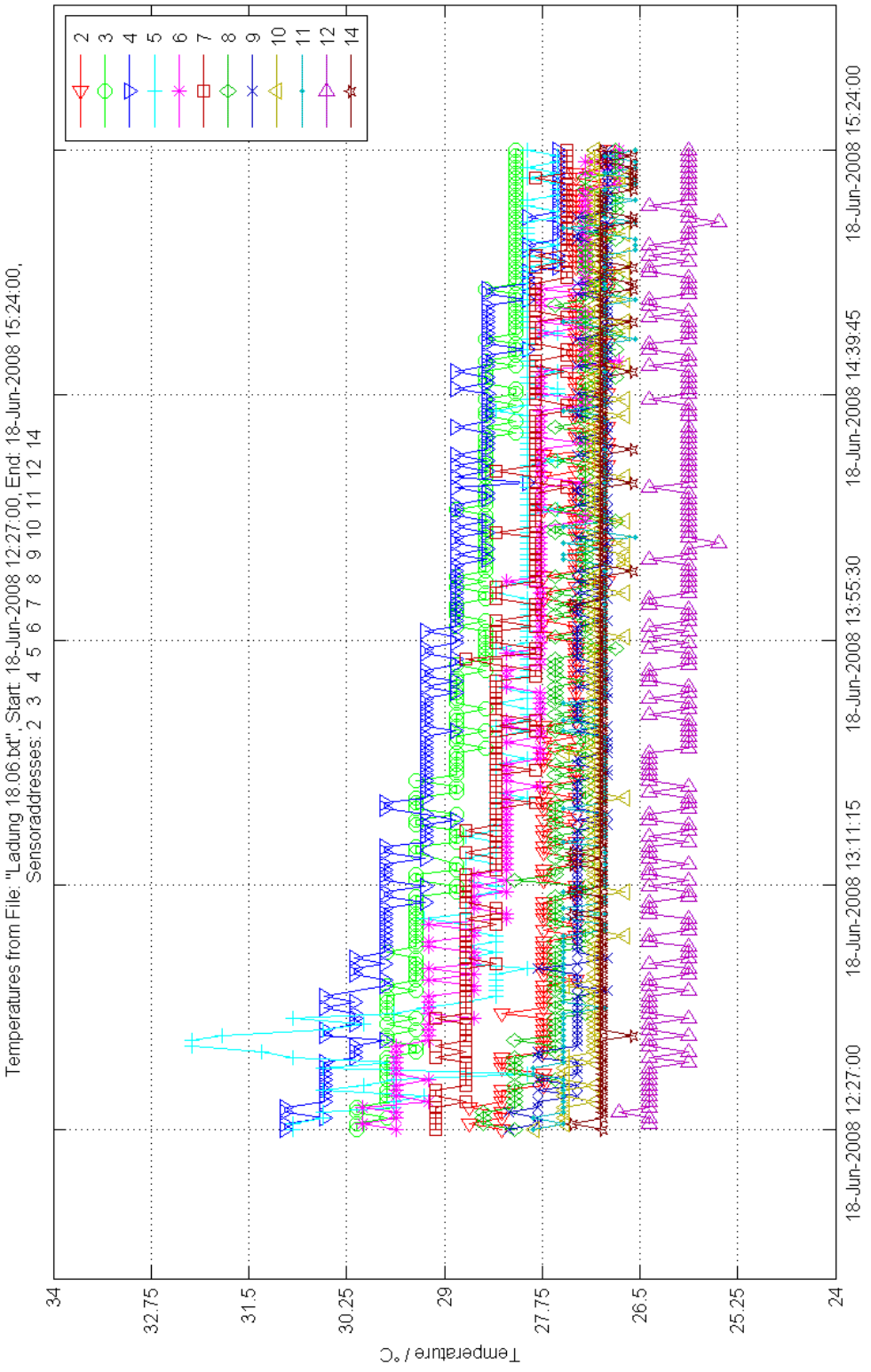


Bild 3.9.: Temperaturverlauf des Ladevorgangs

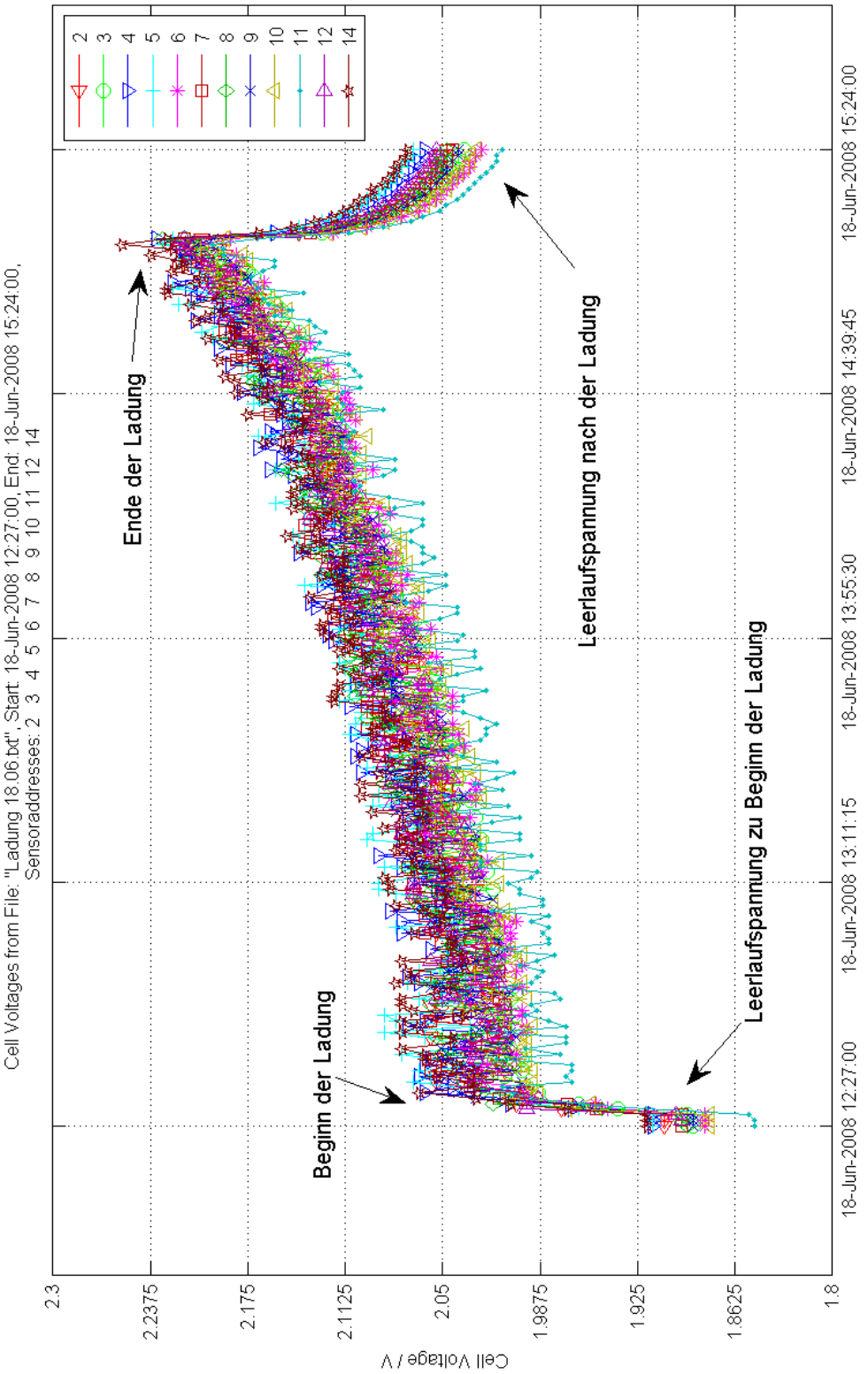


Bild 3.10.: Zellspannungsverlauf des Ladevorgangs

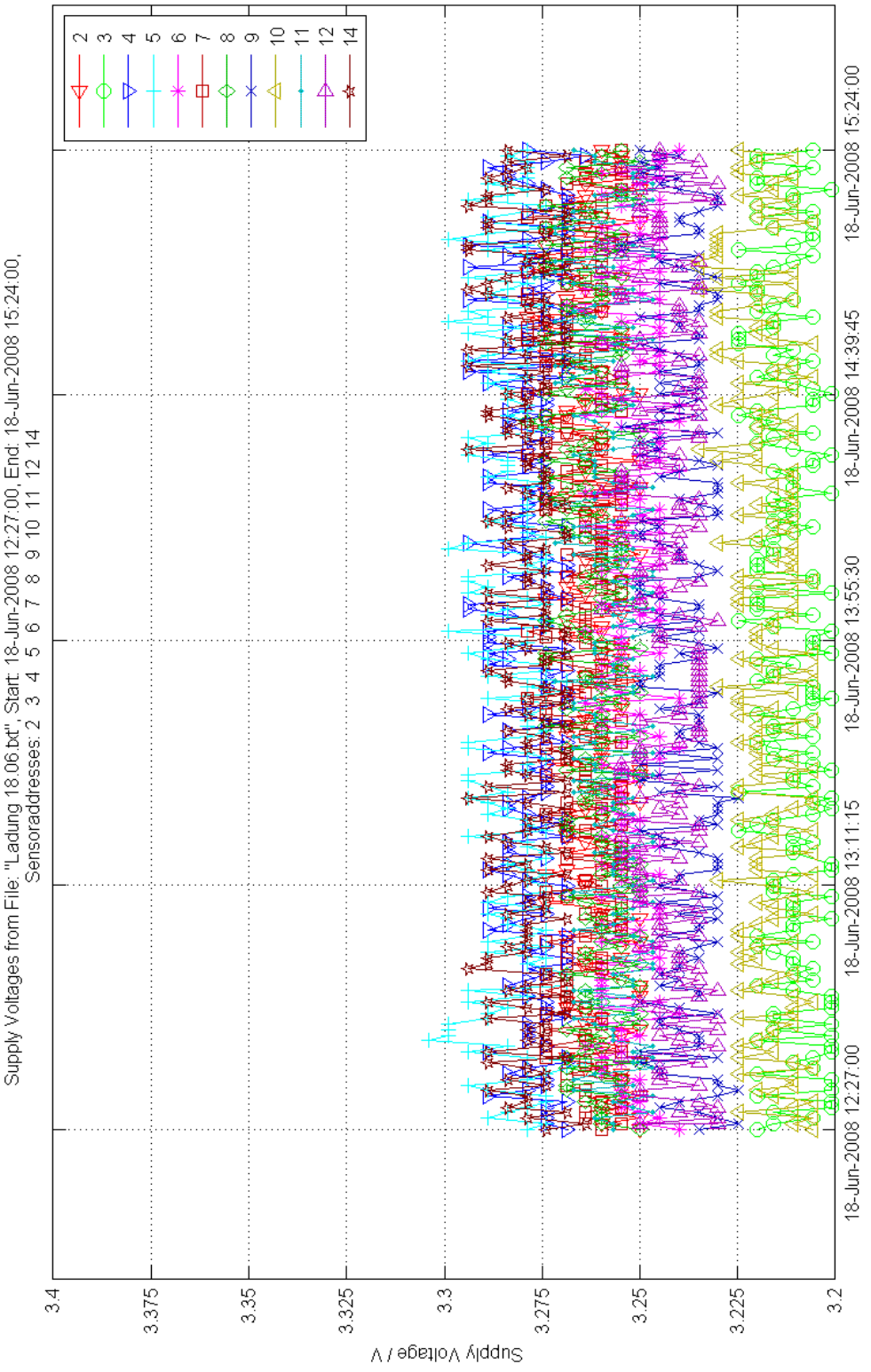


Bild 3.11.: Versorgungsspannungsverlauf des Ladevorgangs

Bei der Temperaturmessung (Bild 3.9) ist die fallende Tendenz an fast allen Sensoren deutlich. Die Ladung erfolgte direkt im Anschluss zur Entladung aus Versuch 1. Darauf begründen sich die unterschiedlichen Startwerte und die daraus resultierenden unterschiedlichen Verläufe. Die maximale Abweichung beträgt 5,4°C die minimale 0,9°C (Tabelle 3.7).

	Beginn	**Ende**
Referenzwert	25,6°C	24,2°C
Min. Abweichung	0,9°C	1,8°C
Max. Abweichung	5,4°C	3,8°C

Tabelle 3.7.: Temperaturabweichung des zweiten Versuchs

Bei der Zellspannungsmessung (Bild 3.10) ist der Einschalt- und Abschaltzeitpunkt der Last kenntlich gemacht. Ebenso wie das Spannungsniveau vor und nach der Ladung. Während der Ladung ist ein starkes Rauschen zu erkennen, welches auf das Ladegerät zurükzuführen ist. Wie im vorangehenden Versuch ergibt die Referenzmessung annähernd gleiche Werte wie das Aufaddieren der Einzelspannungen (Tabelle 3.8).

	Beginn	**Ende**
Referenzwert	23,11 V	24,94 V
Aufaddierte Einzelspannungen	22,78 V	24,63 V

Tabelle 3.8.: Leerlaufspannung als Referenzwert und durch Aufaddieren der Einzelspannungen des zweiten Versuchs

Die Versorgungsspannungsmessung sowie eine detailierte Betrachtung aller Ergebnisse erfolgt in Kapitel 3.7.

3.5. Versuch 3: Entladung

Da die wichtigsten Informationen im Entladebetrieb ermittelt werden, ist eine zweite Messung gemacht worden. Hierbei wurden die Sensoren anders als bei den vorangehenden Versuchen auf der Batterie angeordnet. Die Entladung erfolgt nach dem selben Schema wie bei Versuch 1. Tabelle 3.9 zeigt die veränderte Reihenfolge der Sensoren auf den Batteriezellen. Tabelle 3.10 stellt alle Randbedingungen dar.

Zelle	1	2	3	4	5	6	7	8	9	10	11	12
Sensoradresse	14	12	11	10	9	8	7	6	5	4	3	2

Tabelle 3.9.: Anordnung der Sensoren auf den Batteriezellen für die dritte Messung

	Uhrzeit	**Temperatur**	**Leerlaufspannung**
Beginn der Messung	12:15 Uhr	22,5°C	23,7 V
Ende der Messung	13:40 Uhr	32,2°C	22,7 V

Tabelle 3.10.: Randbedingungen der dritten Messung

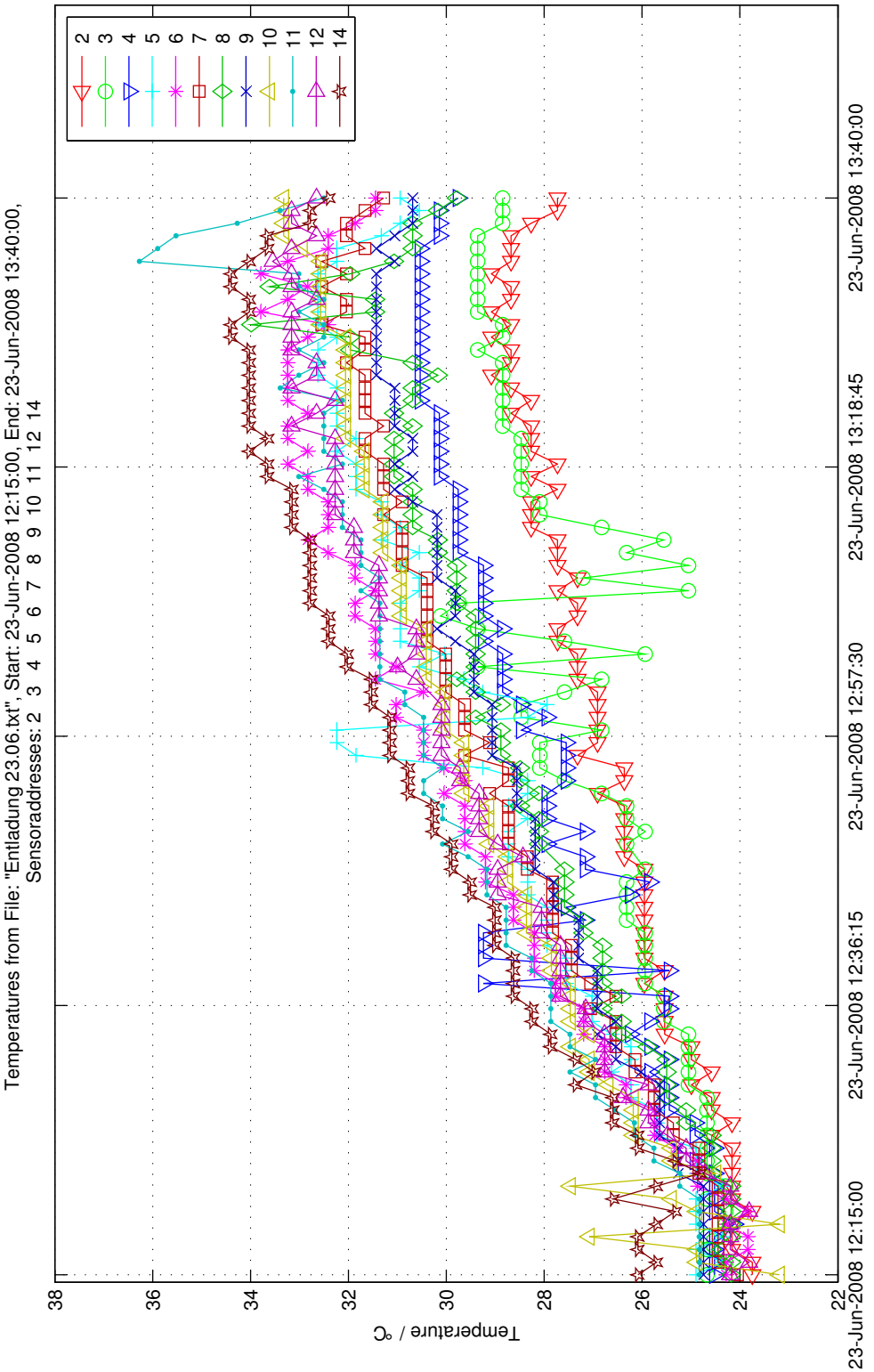


Bild 3.12.: Temperaturverlauf des zweiten Entladevorgangs

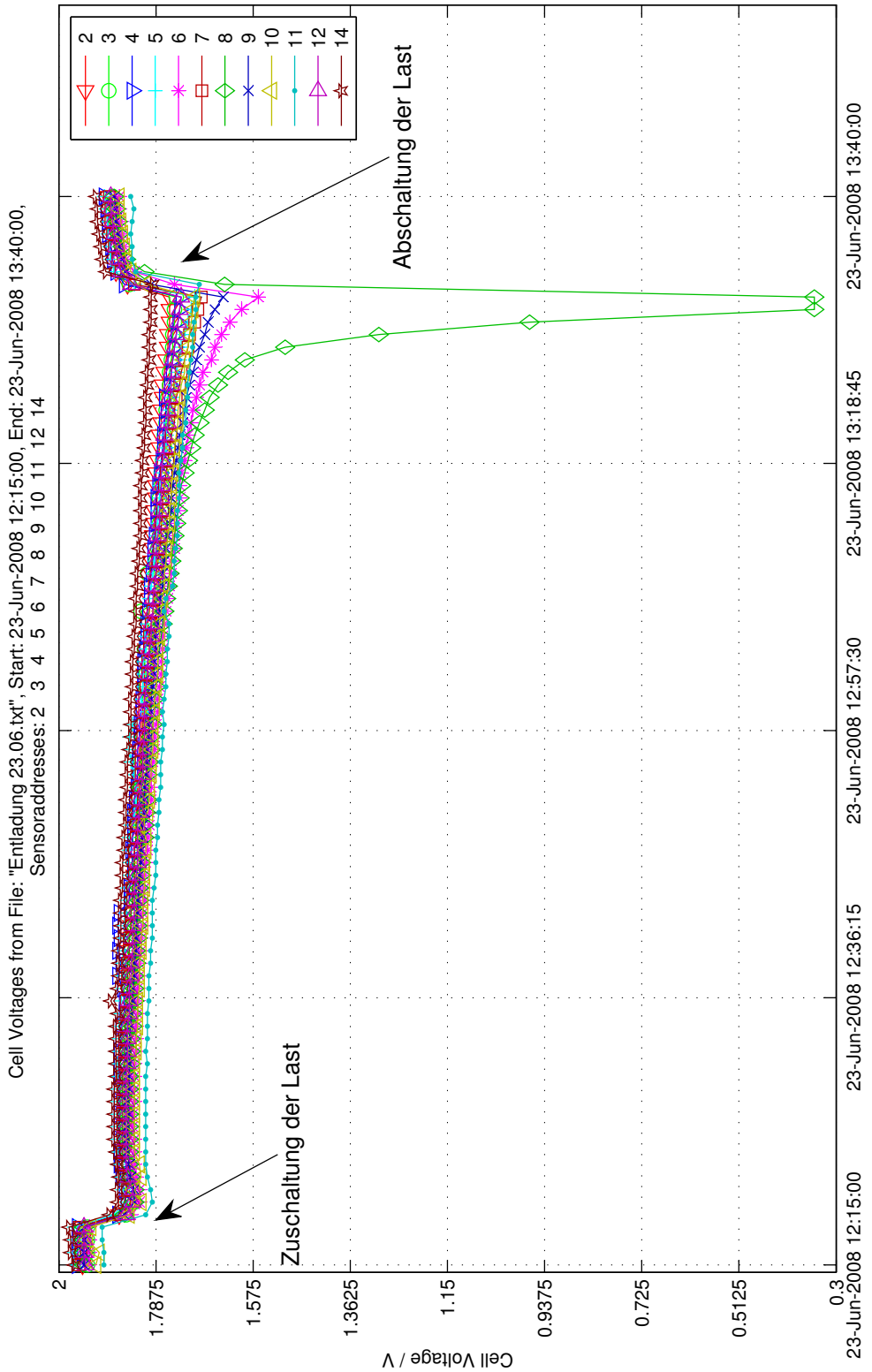


Bild 3.13.: Zellspannungsverlauf des zweiten Entladevorgangs

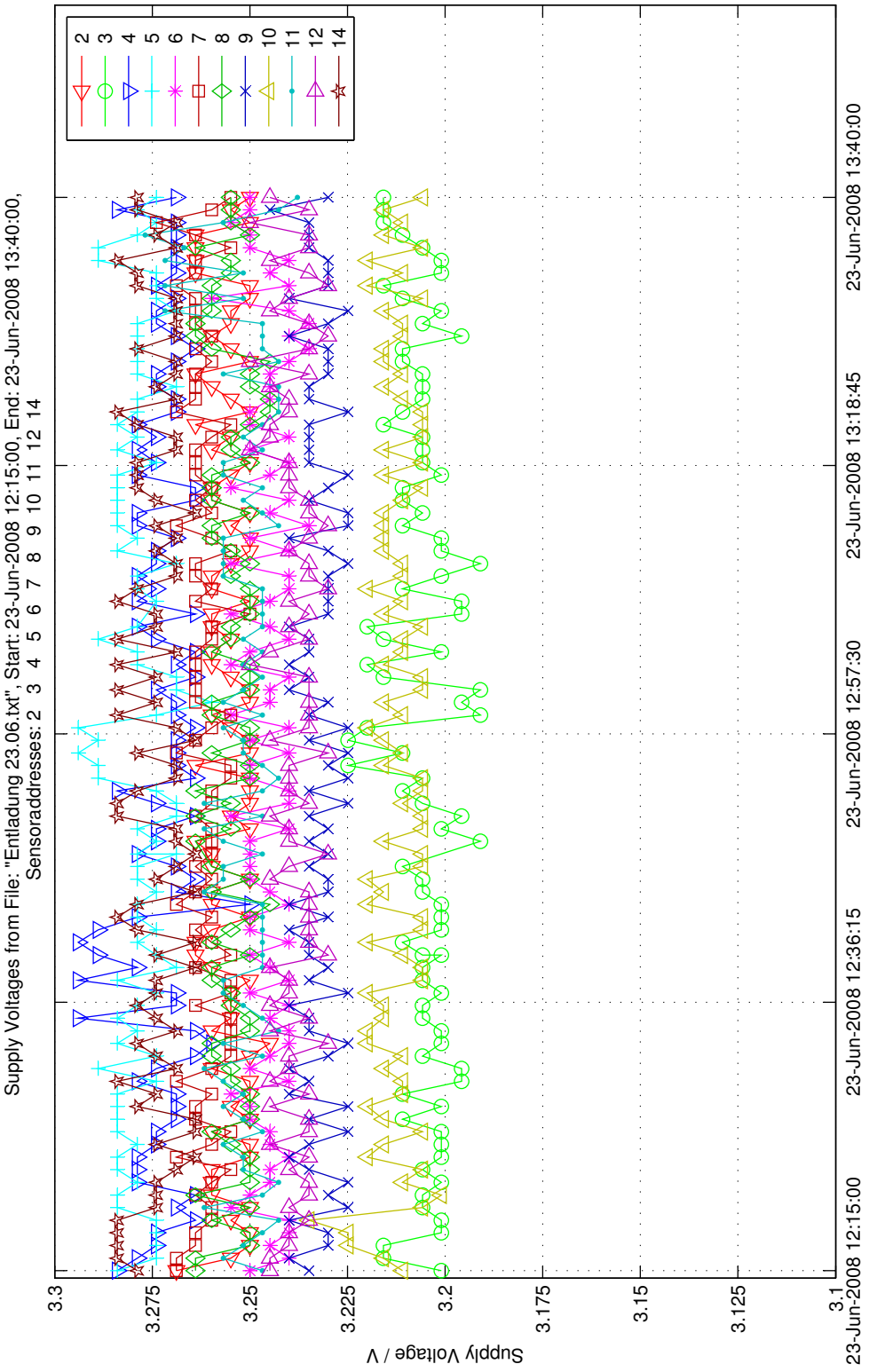


Bild 3.14.: Versorgungsspannungsverlauf des zweiten Entladevorgangs

Bei der Temperaturmessung (Bild 3.12) ist an allen Sensoren die steigende Tendenz deutlich. Die Ausreißer der Sensoren mit der Adresse 3 und 5 (Zelle 11 und 9) sind auf Messfehler zurückzuführen. Da diese Sensoren schon in Versuch 1 Fehler beziehungsweise Meßungenauigkeiten aufwiesen, sind diese zu überprüfen. Die maximale Abweichung beträgt 3,8°C die minimale 1°C (Tabelle 3.11).

	Beginn	**Ende**
Referenzwert	22,5°C	32,2°C
Min. Abweichung	1°C	1,8°C
Max. Abweichung	3,5°C	3,8°C

Tabelle 3.11.: Temperaturabweichung des dritten Versuchs

Dieser Versuch ist im Gegensatz zu Versuch 1 mit anderen Bedingungen gestartet worden. Es handelt sich ebenfalls um eine Entladung, die Aufzeichnung wurde aber schon im Ruhezustand begonnen und die Anordnung der Sensoren ist verändert worden. In Bild 3.13 ist der Einschalt- und Abschaltzeitpunkt der Last kenntlich gemacht. Der Spannungsabfall an dem Sensor mit der Adresse 8 (Zelle 6) beschreibt, wie schon in Versuch 1, den schlechten Zustand (SOH) der Zelle. Die Referenzmessung mit einem Messgerät ergibt annähernd gleiche Werte wie das Aufaddieren der Einzelspannungen (Tabelle 3.12).

	Beginn	**Ende**
Referenzwert	23,77 V	22,71 V
Aufaddierte Einzelspannungen	23,42 V	22,70 V

Tabelle 3.12.: Leerlaufspannung als Referenzwert und durch Aufaddieren der Einzelspannungen des dritten Versuchs

Die Versorgungsspannungsmessung sowie eine detailierte Betrachtung aller Ergebnisse erfolgt in Kapitel 3.7.

3.6. Versuch 4: Ruhemessung

Zuletzt wurde noch eine Ruhemessung durchgeführt. Hierzu wurden die Sensoren wie in Tabelle 3.14 auf der Batterie angeordnet. Diese Messung wird durchgeführt, um zu überprüfen, ob etwaige Störungen, die durch den Lade- oder Entladevorgang hervorgerufen werden, Auswirkungen auf die Messung haben. Die Randbedingungen sind in Tabelle 3.14 dargestellt.

Zelle	1	2	3	4	5	6	7	8	9	10	11	12
Sensoradresse	2	3	4	5	6	7	8	9	10	11	12	14

Tabelle 3.13.: Anordnung der Sensoren auf den Batteriezellen für die vierte Messung

	Uhrzeit	**Temperatur**	**Leerlaufspannung**
Beginn der Messung	10:47 Uhr	22,5°C	23,77 V
Ende der Messung	12:00 Uhr	22,7°C	23,77 V

Tabelle 3.14.: Randbedingungen der vierten Messung

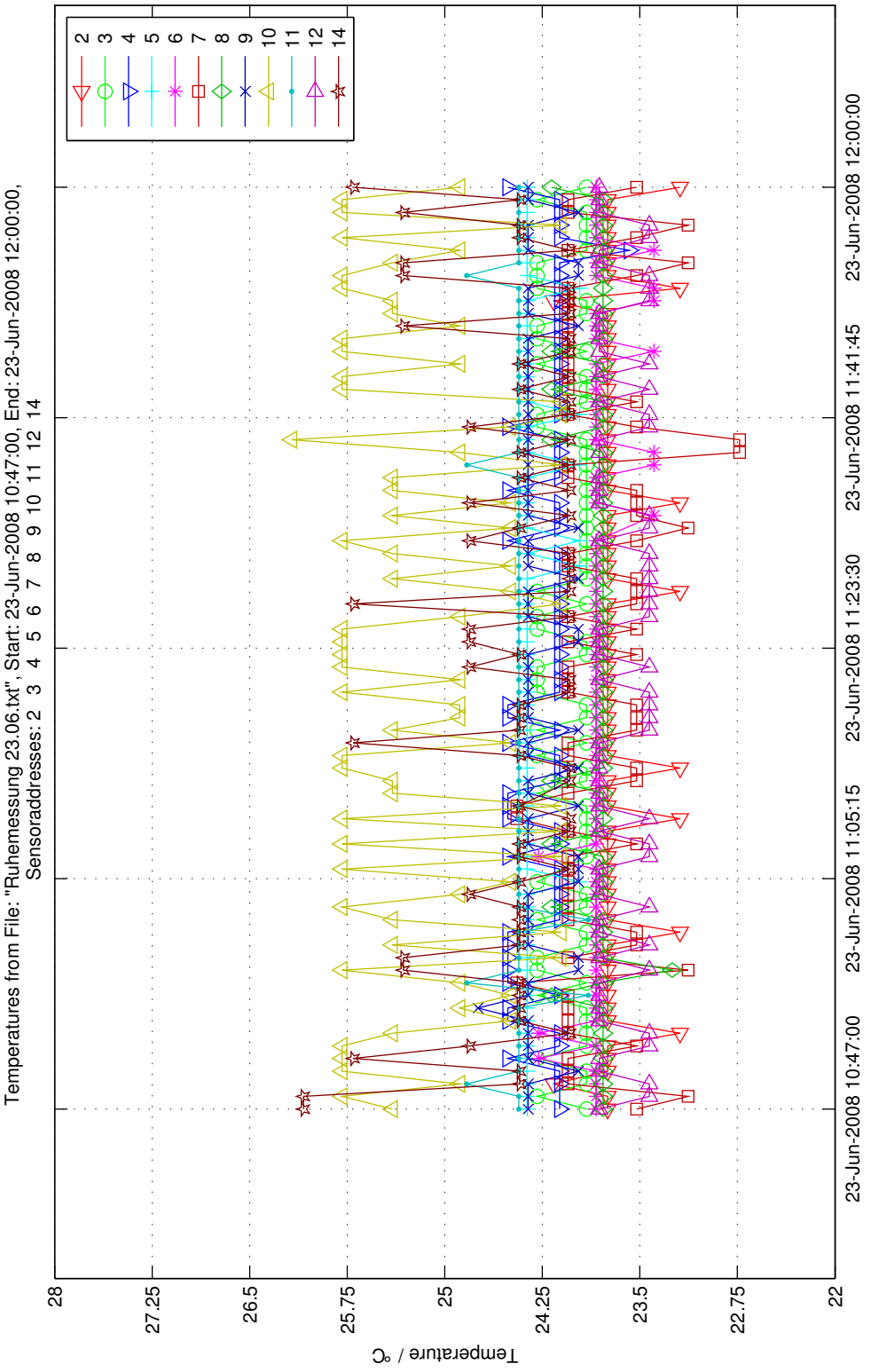


Bild 3.15.: Temperaturverlauf der Ruhemessung

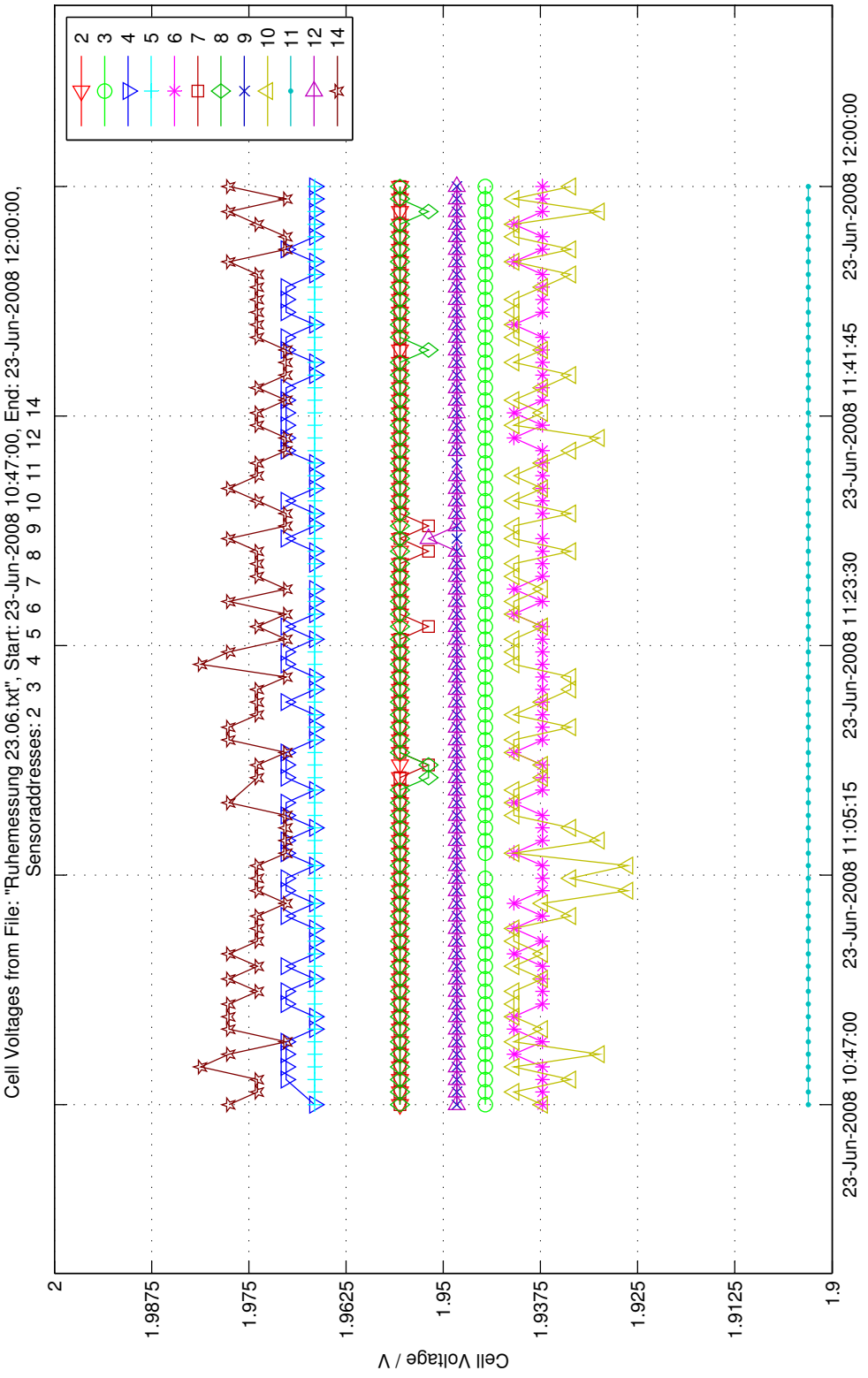


Bild 3.16.: Zellspannungsverlauf der Ruhemessung

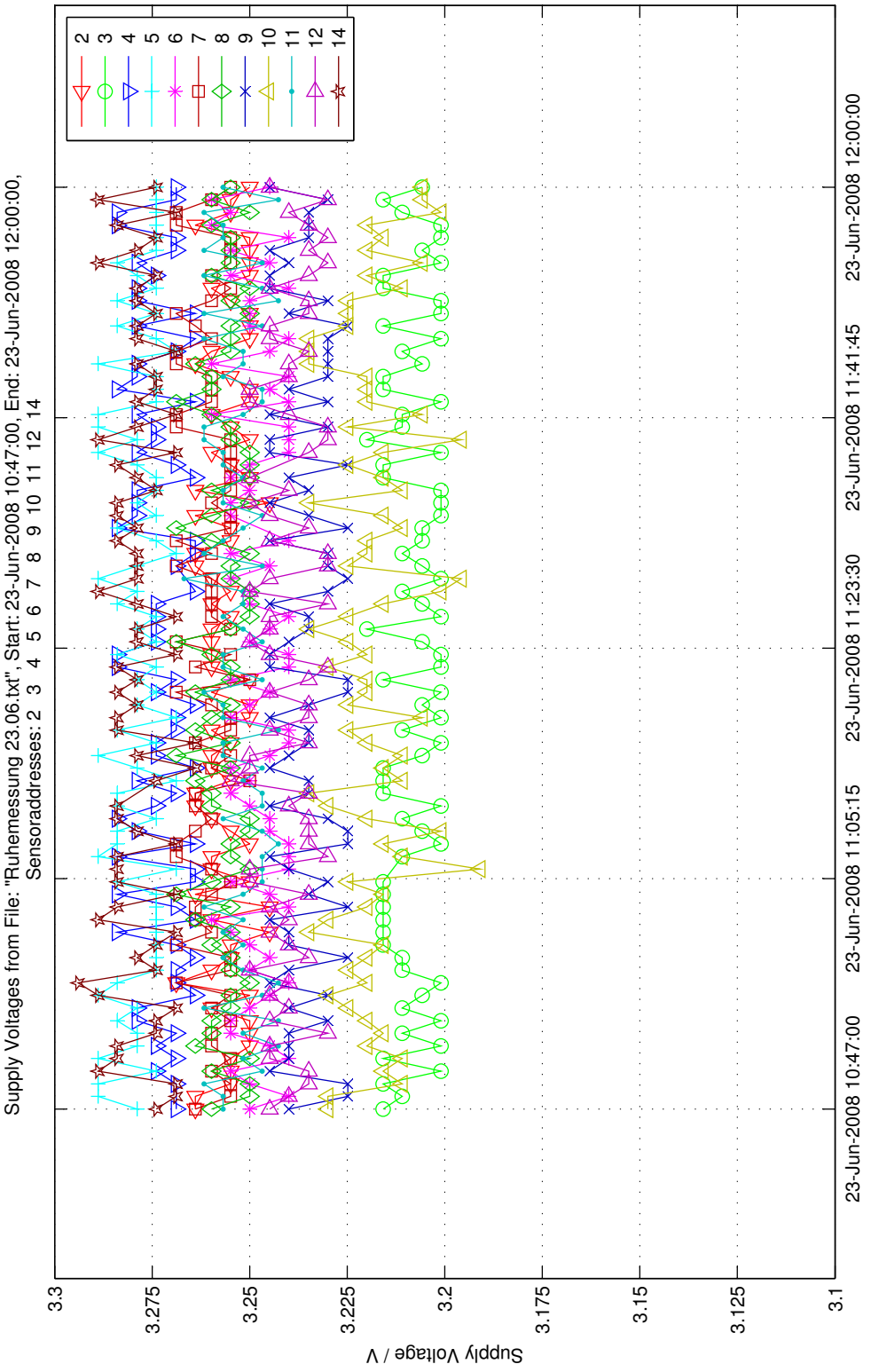


Bild 3.17.: Versorgungsspannungsverlauf der Ruhemessung

Bei der Temperaturmessung (Bild 3.17) sind die Schwankungen, im Vergleich zu den anderen Versuchen, recht klein. Sie umfassen nur einen Messfehler von 1 bis 3 lsb. Die maximale Abweichung beträgt 3,5°C, die minimale 0,3°C (Tabelle 3.3).

	Beginn	**Ende**
Referenzwert	22,5°C	22,7°C
Min. Abweichung	1°C	0,3°C
Max. Abweichung	3,5°C	3,25°C

Tabelle 3.15.: Temperaturabweichung des vierten Versuchs

In Bild 3.13 ist der Spannungsverlauf der Ruhemessung abgebildet. Das auftretende Rauschen ist im Vergleich mit Versuch 2 sehr gering. Dadurch ist bewiesen, dass Entlade- und besonders Ladevorgänge die Messungen beeinflussen. Die Referenzmessung mit einem Messgerät ergibt annähernd gleiche Werte wie das Aufaddieren der Einzelspannungen (Tabelle 3.12).

	Beginn	**Ende**
Referenzwert	23,77 V	23,77 V
Aufaddierte Einzelspannungen	23,50 V	23,49 V

Tabelle 3.16.: Leerlaufspannung als Referenzwert und durch Aufaddieren der Einzelspannungen des vierten Versuchs

Die Versorgungsspannungsmessung sowie eine detailierte Betrachtung aller Ergebnisse erfolgt im nachfolgenden Kapitel.

3.7. Betrachtung der Versuchsergebnisse

Alle Messungen konnten ohne Probleme durchgeführt werden. Es wurde erwartet, dass durch den rein metallischen Körper, eine Datenübertragung nach außen nicht möglich sei. Die Messungen konnten aber durchgeführt werden, ohne die Antenne in den Batterieraum zu montieren. Am Aufschlussreichsten der drei Messungen sind die Temperatur- und die Zellspannungsmessung. Die Messung der Betriebsspannung ist kaum von Belangen für die Zustands- und Ladekontrolle. Daher wird diese nur oberflächlich betrachtet.

3.7.1. Betrachtung der Temperaturmessungen

Bei den Temperaturmessungen kann zum Vergleich die Referenzmessung mit einem externen Thermometer herangezogen werden. Vergleicht man die Referenzwerte mit den aufgenommenen Werten ist bei allen Messungen eine Differenz von bis zu 6°C zu sehen. Ebenfalls ist teilweise ein starkes Schwanken der Sensorwerte zu sehen. Die steigende oder fallende Tendenz ist aber bei allen Messungen im Vergleich zur Referenzmessung gleich. Tabelle 3.17 zeigt die minimalen und maximalen Abweichungen vom Referenzwert jeder Messung.

	Versuch 1	Versuch 2	Versuch 3	Versuch 4
Temperatur bei Beginn	21,2°C	25,6°C	22,5°C	22,5°C
Min. Abweichung	2°C	0,9°C	1,5°C	1°C
Max. Abweichung	4,1°C	5,4°C	3,5°C	3,5°C
Temperatur bei Ende	27,8°C	24,2°C	32,2°C	22,7°C
Min. Abweichung	0,45°C	1,8°C	0,8°C	0,3°C
Max. Abweichung	4,2°C	3,8°C	4,2°C	3,25°C

Tabelle 3.17.: Minimale und maximale Abweichung vom Referenzwert aller Messungen

Trotz der Kalibrierung taucht dieser Fehler auf. Betrachtet man die Formel zur Errechnung des Temperaturwertes (Formel 2.3) wird einer der Gründe deutlich. Bei einer Änderung von einem lsb ändert sich der errechnete Temperaturwert um 0,413°C. Schwankungen von einem lsb sind bei einer Messung mit einem ADC nicht zu vermeiden. Diese Schwankungen werden als Quantisierungsrauschen bezeichnet.

Ein anderer Grund ist, dass die Temperatur intern gemessen wird. Somit spielt die Belastung des Mirkocontrollers eine große Rolle. Ist die Zeit zwischen den Messungen kürzer als die Abkühlphase, wird eine erhöhte Temperatur gemessen. Zur Lösung dieses Problems muss in die Software des Sensors eingegriffen werden. Es wäre möglich, den Wert über einen Zeitraum von ein paar Millisekunden zu mitteln und somit ein besseres Ergebniss zu erhalten.

Die Versuche beweisen, dass eine Temperaturmessung über die interne Diode eines Mikrocontrollers möglich sind. Die Auflösung ist je nach Anwendungsfall zu betrachten. Das auftretende Rauschen ist zum Teil durch den Verbraucher oder das Ladegerät hervorgerufen worden. Zu der schon vorgestellten Softwarelösung ist weiterhin zu überlegen, ob der Einsatz eines Filterelementes notwendig ist.

3.7.2. Betrachtung der Zellspannungsmessungen

Bei den Zellspannungsmessungen kann zum Vergleich die extern gemessene Gesamtspannung als Referenz dienen. Um einen direkten Vergleich herstellen zu können, müssen alle 12 Zellspannungen, zu Beginn und Ende jedes Versuchs, aufaddiert werden. Dies erledigt die MATLAB Software automatisch bei jeder Auswertung. Die Werte werden hierbei in das Befehlsfenster von MATLAB geschrieben. Tabelle 3.18 stellt die Referenzwert und die ermittelten Werte aller Versuche dar.

Die Unterschiede zwischen den beiden Messungen sind auf Messfehler des Multimeters und der Sensoren zurückzuführen. Das verwendete Multimeter der Firma Voltcraft besitzt eine Genauigkeit von 0,5% bei einem Messbereich von 20 V. Die Sensoren haben eine Genauigkeit von ebenfalls 0,5% im Messbereich, welche aber abhängig von der Referenzspannung ist. Zudem sind noch Übergangswiderstände zu berücksichtigen. Addiert man diese Ungenauigkeiten ist es durchaus möglich, eine Maximaldifferenz von 0,3 V zu erhalten, wie es auch bei den dargestellten Messwerten der Fall ist.

Die Zellspannungsmessungen sind ausschlaggebend für die Berechnung des Batteriezu-

Leerlaufspannung	**Versuch 1**	**Versuch 2**	**Versuch 3**	**Versuch 4**
Referenzwert zu Beginn	22,37 V	23,11 V	23,77 V	23,77 V
Aufaddierte Einzelspannungen	22,12 V	22,83 V	23,42 V	23,50 V
Referenzwert zum Ende	22,78 V	24,94 V	22,71 V	23,77 V
Aufaddierte Einzelspannungen	22,67 V	24,63 V	22,70 V	23,49 V

Tabelle 3.18.: Leerlaufspannung als Referenzwert und durch Aufaddieren der Einzelspannungen aller Versuche

stands. Daher wird hier, im Gegensatz zur Temperaturmessung, auf jede einzelne Messung eingegangen.

Im ersten Versuch (Bilden 3.6 bis 3.8) handelt es sich um eine Entladung. In Bild 3.7 ist zu erkennen, dass im letzten Abschnitt drei Spannungen überdurchschnittlich stark zusammenbrechen. Besonders die Zellspannung am Sensor mit der Adresse 7 bricht sehr stark ein. Die dazugehörige Zelle ist Zelle 6 (Bild 3.4). Diese Problematik ist mit einer konventionellen Spannungsmessung der Batteriespannung nicht zu erkennen. Der schlechte Zustand dieser Zellen führt zu einer überdurchschnittlichen Belastung aller restlichen Zellen. Die Vorgaben der vorliegenden Arbeit sind es, ein System zu konstruieren, das solche Fehler erkennen kann. Versuch 3 beschreibt ebenfalls eine Entladung und soll die hier aufgetretenen Probleme belegen.

Im zweiten Versuch (Bilden 3.9 bis 3.11) handelt es sich um einen Ladevorgang. Da sich der Ladestrom auf alle Zellen aufteilt, steigt das Spannungsniveau bei allen Zellen kontinuirlich an (Markierung in Bild 3.10). Das Anfangsniveau von zirka 1,8 V bei allen Zellen, enspricht dem Niveau am Ende der vorigen Messung (Versuch 1). Die Leerlaufspannung von Zelle 6 ist auf gleichem Niveau wie die anderen. Somit ist es nicht möglich, eine defekte Zelle während der Ladung oder im Ruhezustand zu erkennen.

Im Gegensatz zur ersten Messung ist ein starkes Rauschen zu erkennen. Diese Problematik war zu erwarten, da durch das Ladegerät erzeugte Oberwellen die Spannungsmessung beeinflussen. Es sind zwei Lösungsvorschläge für dieses Problem möglich. Bei der ersten Lösung wird ein geeigneter Filter entwickelt und in Hardware realisiert. Die zweite Lösung umfasst die Filterung der Messwerte. Hierzu wird eine Fourier-Transformation durchgeführt und im Frequenzbereich ein Filter angewendet. Da die zu erwartenden Störungen der Nennfrequenz und der doppelten Nennfrequenz des Versorgungsnetzes entsprechen, müssten dazu mindesten 200 Messwerte pro Sekunde aufgezeichnet werden. Eine Änderung der Software im Sensor wäre dazu nötig. Weiterhin ist zu überprüfen, ob diese Probleme im Zusammenhang mit neueren Ladegeräten auftaucht.

Im dritten Versuch (Bilden 3.12 bis 3.14) wurde abermals eine Entladung durchgeführt. Hier wurde die Anordnung der Sensoren verändert (Tabelle 3.10). Wie auch bei der ersten Entladung bricht die Spannung von Zelle 6 stark zusammen (Bild 3.13 Sensor 8). Die Spannung sinkt bei diesem Versuch an Zelle 6 auf unter 0,5 V, im vorigen waren es noch zirka 0,9 V. Wird nun ohne Rücksicht auf die defekte Zelle fortgefahren, wird es schnell zu einem vollständigen Defekt kommen und die Batterie somit unbrauchbar werden. Dies bringt auch die Problematik bei herkömmlichen Systemen zum Vorschein. Bei einer Über-

prüfung der gesamten Spannung (Tabelle 3.18) ist der Zustand der einzelnen Zelle nicht erkennbar. Der Rückschluss daraus wäre, dass die Batterie keinen Defekt aufweist.

Der letzte Versuch (Bild 3.15 bis 3.17) beschreibt eine Ruhemessung. Hier sind über einen Zeitraum von 75 Minuten Messwerte aufgenommen worden, ohne die Batterie zu belasten. Das einzige Rauschen welches hier auftritt, ist Quantisierungsrauschen, wobei das Spannungsniveau konstant bleibt. Somit dient dieser Versuch als Beweis, dass das Rauschen im zweiten Versuch nicht auf die Sensoren zurückzuführen ist.

3.7.3. Betrachtung der Versorgungsspannungsmessung

Zuletzt sind die Versorgungsspannungsmessungen zu betrachten. Diese sind, wie schon beschrieben, für die Bestimmung des Ladezustands nicht nötig. Bei allen vier Versuchen sind hier die gleichen Erscheinungen zu erkennen: Die Spannungswerte sind nicht konstant. Dies liegt an dem verwendeten Step-Up-Converter. Seine Ausgangsspannung schwankt im Betrieb um zirka 0,1 V. Für den Betrieb der Sensoren stellt es aber kein Problem dar. Das einzige Problem an dieser Änderung ist eine eventuelle Änderung der Referenzspannung für die Spannungsmessung. Daraus resultiert ein minimal größerer Messfehler als angenommen. Dieser dürft aber nur im Promillbereich liegen und zu vernachlässigen sein.

Betrachtet man alle aufgenommn Werte, stimmen diese mit den erwarteten Werten überein. Die Temperaturmessungen sind für diesen Fall ausreichend, sollten aber bei weiterer Anwendung noch verbessert werden. Die Spannungsmessungen sind auch für weitere Anwendungen ausreichend. Hierbei sollte nur der Einsatz eines Filters überprüft werden.

4. Fazit

Das Ziel dieser Arbeit war es, ein System zu entwickeln, welches ermöglicht, jede Zelle einer Batterie zu überwachen und deren Ladezustand zu ermitteln. Der Aufbau sollte so kostengünstig wie möglich gestaltet werden. Dennoch sollte das System robust genug sein, um den Einsatz in Kraftfahrzeugen oder Flurförderzeugen zu erlauben.

Die Hardware wird den geforderten Ansprüchen gerecht. Die verwendeten Sensoren sind robust und universell einsetzbar. Die Datenübertragung erfolgt hier über Funk. Der Frequenzbereich, in dem die Daten gesendet werden, ist 433 MHz und entspricht einem frei verfügbaren ISM-Bereich. In diesem Bereich und mit der gewählten Hardware ist es möglich, die Daten mit einer Geschwindigkeit von 20 kbps zu senden. Diese Geschwindigkeit wird derzeit nicht ausgereizt und birgt somit genug Reserven für weitere Anwendungen. Mit dieser Konstellation ist die Übertragung der Daten zuverlässig und kostenlos. Ebenfalls entfallen Kosten für eine Verkabelung, die die bei einem Bus-System notwendig wären und das Steuergerät wird galvanisch von den Sensoren getrennt. Somit entfällt eine kostspielige galvanische Trennung im Steuergerät.

Das entwickelte Steuergerät ist ebenfalls sehr robust aufgebaut. Das verwendete Aluminium Spritzgußgehäuse hält starken Belastungen stand und kann somit bedenkenlos im Kraftfahrzeug oder an einem Flurförderzeug installiert werden. Das verwendete Entwicklungsboard enthält alle notwendigen Elemente. Nur der Empfänger musste extern montiert werden. Aus Kostengründen wäre aber zu überlegen, eine eigene Entwicklungsplattform für weitere Anwendungen zu entwickeln. Die Messaufbauten für Audiosignalverarbeitung, die auf dem Entwicklungsboard zu finden sind, werden nicht verwendet und sind daher überflüssig und kostenintensiv.

Es sind aber auch Unstimmigkeiten aufgetaucht, die es bei weiterer Anwendung noch zu beheben gilt. Besonders der Stromverbrauch des Steuergerätes und der Sensoren ist zu senken, wenn es zu einem anderen Einsatz als der Antriebsbatterie kommen sollte. Dies kann geschehen, indem die Software beider Systeme angepasst wird. Zudem ist die LED zur Übertragungskontrolle von den Sensoren zu entfernen.

Weiterhin ist die Software des Sensors nachzuarbeiten. Wichtig hierbei sind die Parameter der Datenübertragung. Dazu gehören die zu Testzwecken veränderten Zeiten der *LOW* und *HIGH* Pegel. Ebenfalls sind die Wartezeiten zwischen den Übertragungen (ALOHA Verfahren) zu überdenken. Sind diese zu hoch schränken sie die Datenübertragung stark ein. Die hier verwendete Zeit von 1,8 bis 3,6 Sekunden ist für den behandelten Versuchsaufbau ausreichend.

Neben den Parametern der Datenübertragung ist in der Software ebenfalls die Temperaturaufnahme zu verändern. Derzeit wird ein Wert pro Minute aufgenommen und verwendet. Um eine höhere Genauigkeit und einen stabileren Verlauf (Stichwort Quantisierungsrauschen) zu erhalten, können mehrere Werte aufgenommen werden, um daraus

den Mittelwert zu berechnet. Diese Erkenntnisse beruhen auf der aktuellen Anwendung. Wird für weitere Anwendungen jede Sekunde ein Wert gefordert, kann nicht der Mittelwert aus ein paar Millisekunden herangezogen werden. In dem behandelten Aufbau wäre es sinnvoll, mit dem Mittelwert zu arbeiten. Ebenfalls ist zu überlegen, ob die Genauigkeit der Temperaturmessungen ausreicht oder ob externe Sensoren verwendet werden sollten. Für den Fall der Antriebsbatterie ist die Messung der Temperatur allerdings ausreichend. Die Temperaturverläufe geben hier ausreichend Aufschluss.

Ebenso ist die Bedienung durch den Benutzer zu überdenken. Sie entspricht hier einer Mischung aus der Eingabe über die vorhandenen Tasten und der seriellen Schnittstelle. Die Bedienung der Tasten sollte auf ein Minimum beschränkt werden und gegebenenfalls sogar nur die Anzeige der aktuellen Werte erlauben. Zudem ist eine Software zur Konfiguration des Steuergerätes empfehlenswert. Dabei würde die aufwändige Menüführung am Terminal entfallen und alle relevanten Einstellungen könnten direkt am PC eingegeben und visuell dargestellt werden.

Die Ausgabe und Auswertung der Messwerte ist mit der entworfenen MATLAB Software möglich. Die Software stellt in drei Fenstern die verschiedenen Messwerte dar. Hierbei kann jede Kurve aktiviert oder deaktiviert werden. Die Einteilung der Achsen erfolgt dabei automatisch.

Die vier durchgeführten Versuche sind mittels der MATLAB Software anschaulich dargestellt. Die Intention der vorliegenden Arbeit ist es, zu beweisen, dass eine Überwachung der gesamten Batteriespannung nicht ausreicht, um den Zustand der Batterie zu bestimmen. Die Messergebnisse bestätigen diese Annahme. Die vorgenommenen Entladevorgänge beschreiben einen starken Einbruch der Spannung einer Zelle. Dieser ist auf den schlechten Zustand der Zelle zurückzuführen (SOH). Konventionelle Systeme würden in diesem Fall nur eine absinkende Gesamtspannung erkennen. Solange sich diese Spannung noch im erlaubten Rahmen bewegt (Entladeschlussspannung pro Zelle multipliziert mit der Anzahl der Zellen), würde hier kein Fehler erkannt werden. Mit Hilfe des entwickelten Systems hingegen ist ein Absinken der Spannung einer Zelle zu erkennen und eine frühe Einleitung von Maßnahmen dieses zu verhindern können eingeleitet werden.

Um Aussagen über den Ladezustand (SOC) zu treffen, sollten weitere Untersuchungen angestrebt werden. Die Messung der Spannung in Bezug zur Temperatur ist für einige Anwendungen nicht ausreichend. Gegebenfalls ist es notwendig, zusätzlich eine Impedanzmessung an jeder Zelle durchzuführen.

Im Ganzen Betrachtet ist im Laufe dieser Arbeit ein System entwickelt worden, welches den gesetzten Anforderungen gerecht wird. Die Versuche, die durchgeführt wurden, geben einen ersten Aufschluss über weitere Möglichkeiten der Spannungs-Überwachung einer Batterie. Um so ein System in Kraftfahrzeugen oder Flurförderzeugen einsetzen zu können, sind allerdings weitere Versuchsreihen notwendig.

Literaturverzeichnis

[1] MCA-Malta Communications Authority. European Conference of Postal and Telecommunication Administrations. http://www.itu.int/, Abrufdatum: 22.05.2008.

[2] Dietrich Berndt. *Bleiakkumulatoren*. VDI-Verlag, Duesseldorf, 11. neubearb. u. erw. aufl. edition, 1986. ISBN 3-18-400534-8. XV, 346 S. : ZAHLR. ILL., GRAPH. DARST. pp.

[3] ITU-R Web Coordinator. International Telecommunication Union. http://www.itu.int/, Abrufdatum: 22.05.2008.

[4] ITU-R Web Coordinator. International Telecommunication Union - Radiocommunication Sector: FAQ. http://www.itu.int/ITU-R/terrestrial/faq/index.html#i004, Abrufdatum: 22.05.2008, letztes Update am 08.03.2007.

[5] Bill Darden. Batteryfaq. http://www.batteryfaq.org/, Abrufdatum: 10.05.2008, letztes Update am 28.10.2007.

[6] Karl-Joachim Euler. *Batterien und Brennstoffzellen : Aufbau, Verwendung, Chemie*. Springer-Verl., Berlin, 1982. ISBN 3-540-11378-9. XII, 206 S.: Ill., graph.Darst. pp.

[7] Michael Frey. Anorganische Chemie für Schüler: Elektrochemie. http://de.wikibooks.org/wiki/, Abrufdatum: 15.05.2008, letztes Update am 20.01.2008.

[8] Janzen Gerd. *Kurze Antennen*. Frankh., Stuttgart, 1986. ISBN 3-440-05469-1.

[9] Johnson Controls Autobatterie GmbH. Autobatterien *VARTA*. http://www.Varta-automotives.de, Abrufdatum: 07.05.2008.

[10] Heidi Löbert Guenter Gauglitz. Akku, Kurs Nr.51041. http://www.chemgapedia.de/.../akku.vlu.html, Abrufdatum: 15.05.2008.

[11] Peter H.L. Notten Henk Jan Bergveld, Wanda S. Kruijt. *Battery Management Systems - Design by Modelling*. Kluwer Academic Publishers, Netherlands, 2002. ISBN 1-4020-0832-5.

[12] Hella KGaA Hueck&Co. Powermanagement - Der intelligente Batteriesensor (IBS). Abrufdatum 02.07.2008.

[13] Infineon Technologies AG. *Manual TDA5100A - Wireless Components: ASK Transmitter 434 MHz*, 15.02.2001 edition, Abrufdatum: 12.04.2008.

[14] Finkenzeller Klaus. *RFID Handbuch.* Carl Hanser Verl., München, Wien, 4. aufl. edition, 2006. ISBN 3-446-40398-1.

[15] Matthias M. Tabellensammlung Chemie: Elektrochemische Spannungsreihe. http://de.wikibooks.org/wiki/, Abrufdatum: 15.05.2008, letztes Update am 19.03.2008.

[16] Dr. Alfons Graf Infineon Technologies AG Munich. Semiconductors in the 42V PowerNet. September 17-18, 2001.

[17] Pia O. Der Bleiakkumulator. http://referateguru.heim.at/Blei.htm, Abrufdatum: 07.05.2008.

[18] Olimex Ltd. *Schematic MSP430-169STK*, Abrufdatum: 19.06.2008.

[19] Thomas Rücker. Woran sterben Starterbatterien. http://www.microcharge.de/Bleiakku-Interna.html, Abrufdatum: 09.05.2008.

[20] RF Monolithics Incorporated. *Manual RX5000 - 433,92 MHz Hybrid Receiver*, Abrufdatum: 12.04.2008.

[21] K.-R. Riemschneider. Persönliche Unterlagen über Vorarbeiten Batteriesensorik. 2008.

[22] Samsung Electronic CO., Ltd. *K9F6408U0A-TCB0, K9F6408U0A-TIB0 FLASH MEMORY Manual*, Abrufdatum: 30.06.2008.

[23] Nigel Scott. Batteriesensor verlängert Lebensdauer. http://www.elektronikpraxis.vogel.de/.../articles/68941/, 20.03.2007, Abrufdatum: 02.07.2008.

[24] STMicroelectronis. *Manual L6920 - 1V High efficiency synchronous step up converter*, Abrufdatum: 30.05.2008.

[25] Texas Instruments. *MSP430x1xx Family - User's Guide*, 2006 edition, Abrufdatum: 10.01.2008.

[26] Paul Trueb, Lucien F. ; Rüetschi. *Batterien und Akkumulatoren : mobile Energiequellen für heute und morgen.* Springer, Berlin [u.a.], 1998. ISBN 3-540-62997-1. 225 S. : Ill., graph. Darst. pp.

Bildverzeichnis

Tabellenverzeichnis

A. Verwendete Software

A.1. Entwicklungsumgebung (SDK)

Die Auswahl der SDK beschränkte sich auf zwei verschiedene Systeme. Das erste System ist die Studentenversion des IAR Compilers von IAR Systems, das zweite Eclipse unter Verwendung des MSPGCC Compilers.

Beide SDK haben sowohl Vor- als auch Nachteile. Der ausschlaggebene Punkt bei der Auswahl war, dass der Compiler von IAR Systems in der Studentenversion auf 2kB Programmgröße beschränkt ist. Da das Projekt diese Programmgröße in der Planungsphase schon bei Weitem überschritt, wurde Eclipse und der MSPGCC Compiler gewählt.

Bild A.1.: Ablaufdiagramm

Abbildung A.1 beschreibt den Ablauf der SDK. Eclipse stellt hier die zentrale Einheit dar, über welche der Quellcode geschrieben, kompiliert und an den Mikrocontroller gesendet wird. Der MSP430Gdb-proxy stellt die Verbindung zwischen dem Programmieradapter und Eclipse her. Der Proxy hat die zentrale Aufgabe, die Steuerdaten des Compilers, das gdb-remote controll protocol, in Steuerdaten für den MSP430 umzuwandeln.

Eine ausführliche Anleitung über die Installation und Konfiguration von Eclipe, im Zusammenhang mit dem MSPGCC Compiler, wurde während der Arbeiten erstellt. Diese liegt bei Prof. Dr.-Ing. Karl-Ragmar Riemschneider, HAW Hamburg, vor.

A.1.1. Eclipse

Eclipse ist eine Open-Source SDK, die für nahezu jede Entwicklung kundenspezifischer Programme und Anwendungen eingesetzt werden kann. Entwickelt wurde Eclipse in Java um Java zu programmieren. Durch Zusatzmodule kann in nahezu jeder Prorammiersprache entwickelt werden. Zu beziehen ist Eclipse unter der „Eclipse Public License - v 1.0“ von der Herstellerhomepage (http://www.eclipse.org). Die für diese Arbeit verwendete Version ist 3.3.1.1 Europa vom 23. Oktober 2007.

In Abbildung A.2 sind die beiden Benutzeroberflächen von Eclipse dargestellt. In der Programmieroberfläche (markiert mit 2) werden die Projekte erstellt und bearbeitet. Nachdem der Code geschrieben ist, kann in die Debuggeroberfläche (markiert mit 1)

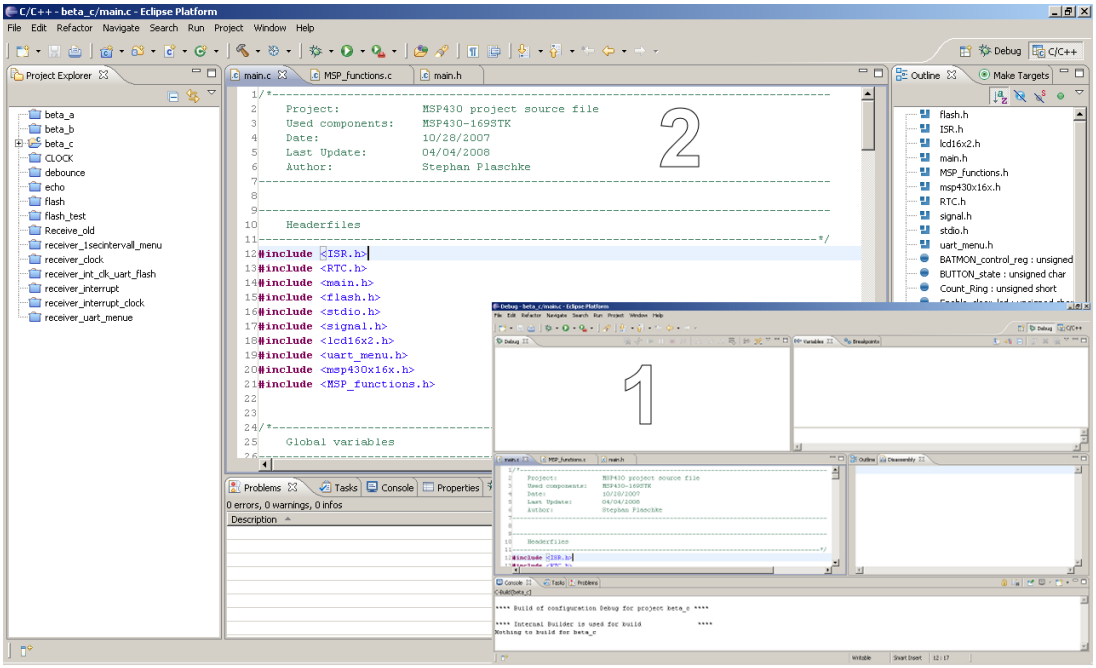


Bild A.2.: Eclipse Benutzeroberfläche

gewechselt werden. Dort kann der Code an den Mikroprozessor übertragen werden und Schritt für Schritt debugged werden.

A.1.2. MSPGCC

Der MSPGCC Compiler ist ein Teil der „GNU Compiler Collection (GCC)“. Zu beziehen ist dieser unter den Bedingungen der „General Public License (GPU)“ von der Herstellerhomepage (http://mspgcc.sourceforge.net). Die derzeit aktuelle Version MSPGCC 20070216 vom 17. Februar 2007 enthält einige Fehler, daher wird die Version 20060502 vom 01. Mai 2006 verwendet.

A.2. MATLAB

MATLAB ist eine kostenpflichtige Software der Firma The MathWorks. Eingesetzt wird es, um numerische Berechnungen durchzuführen und die Ergebnisse grafisch darzustellen. Der Name MATLAB wurde von seinem primären Einsatzgebiet, der Matritzenberechnung, abgeleitet (*MAT*rix *LAB*oratory). Die in der vorliegenden Arbeit verwendete Version wurde von der HAW Hamburg für Studierende zur Verfügung gestellt. Die Versionsnummer lautet 7.6.0.324 (R2008b) vom 10. Februar 2008.

Die in Abbildung A.3 dargestellte Oberfläche zeigt den Editor im Vordergrund (markiert mit 1) und die reguläre Benutzeroberfläche im Hintergrund (markiert mit 2). Der

Editor erlaubt es, MATLAB Skripte zu schreiben, sogenannte M-Files. Diese werden in der regulären Benutzeroberfläche ausgeführt und abgearbeitet.

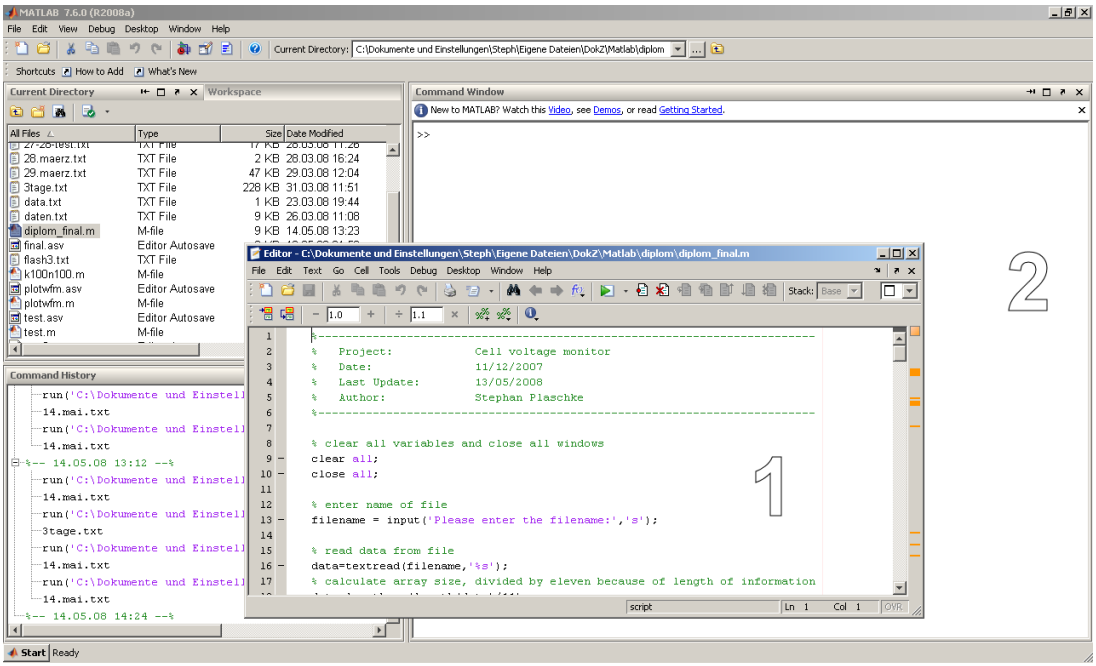
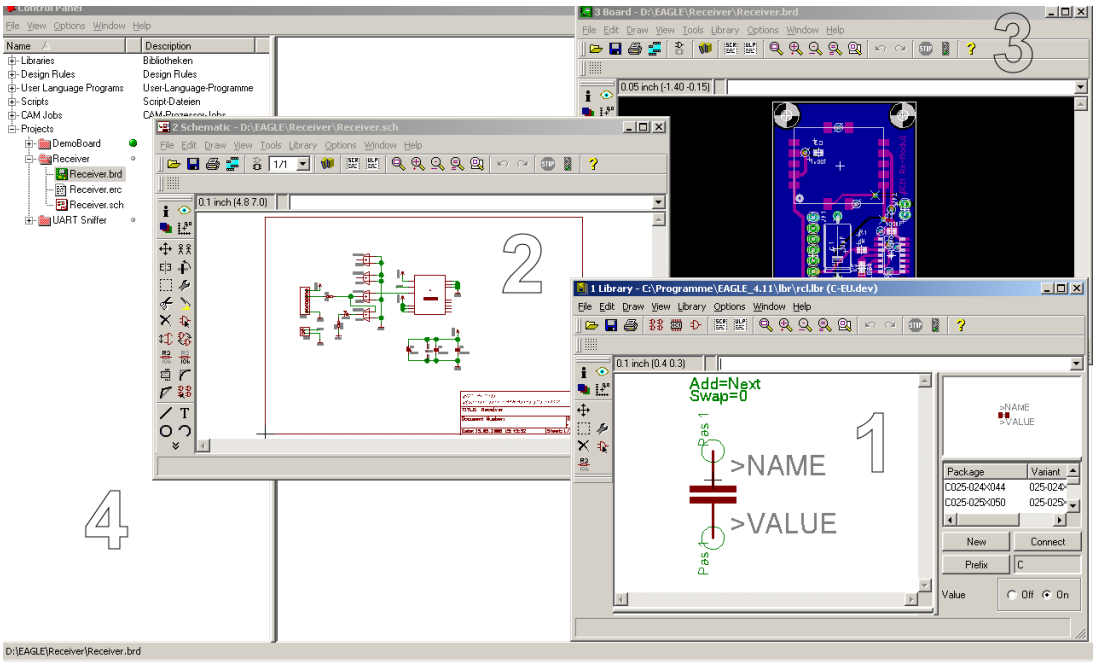


Bild A.3.: Matlab Benutzeroberfläche

A.3. Eagle Layout Editor

EAGLE ist eine kostenpflichtige Software der Firma Cadsoft zur Erstellung von Leiterplatten. Es besteht aus mehreren Teilprogrammen, mit denen in mehreren Schritten eine Leiterplatte entworfen wird. Der Name „EAGLE“ setzt sich aus den Anfangsbuchstaben von: **E**infach **A**nzuwendender **G**rafischer **L**ayout-**E**ditor zusammen. Die verwendete Version ist 4.11 Professional Edition.

In Abbildung A.4 sind alle Teilprogramme von EAGLE dargestellt. Im Vordergrund ist der Bauteileditor (markiert mit 1) dargestellt. Hier können spezielle Bauteile, die nicht in den mitgelieferten Bibliotheken enthalten sind, erstellt und bearbeitet werden. Dahinter ist der Schaltplaneditor (markiert mit 2) dargestellt. Hier wird der Schaltplan mit den Bauteilen aus der Bibliothek entworfen. Um daraus die Platine zu erstellen, wird in den Platineneditor (markiert mit 3) gewechselt. Hier sind die Bauteile, die im Schaltplaneditor ausgewählt wurden, abgelegt und können auf der Platine angeordnet werden. Auch die erforderlichen Leitungen sind durch Linien gekennzeichnet und können entweder per Hand einzeln oder automatisch durch eine Softwareroutine, den Autorouter, verlegt werden. Um die einzelnen Projekte zu koordinieren, kann in die allgemeine Benutzeroberfläche (markiert mit 4) gewechselt werden.



1 Bauteileditor, 2 Schaltplaneditor, 3 Platineneditor, 4 allgemeine Benutzeroberfläche

Bild A.4.: EAGLE Benutzeroberfläche

B. Quellcode

B.1. *C* Quellcode

B.1.1. Sensor-System

Listing B.1: Quellcode des Sensor-Systems (batsens1.c)

```
1  #include <io.h>
2  #include <signal.h>
3
4  // -------------------------------------------------------------------------
5
6  unsigned char txCount = 0;
7  unsigned char txNextOut = 0;
8  unsigned char txByte = 0;
9  unsigned char txBit = 0x80;
10
11 unsigned char txData[13] =
12 {
13         0x00, 0x01,               // RunIn
14         0x00, 0x00, 0x00, 0x00,   // ID
15         0x00, 0x00,               // Temperature
16         0x00, 0x00,               // Battery
17         0x00, 0x00,               // Supply
18         0x00                      // Parity XOR[2 .. 11]
19 };
20
21 interrupt (TIMERA0_VECTOR) timerRoutine()
22 {
23         if (txCount)
24         {
25                 if (txNextOut)
26                         P1OUT |= 0x01; // P1.0 set
27                 else
28                         P1OUT &= ~0x01; // P1.0 clear
29
30                 if (txCount < 6*2)
31                         txNextOut = 0; // 6 Takte uncoded '0'
32                 else if (txCount < (6+4)*2)
33                         txNextOut = 1; // 4 Takte uncoded '1'
34                 else if (txCount < ((6+4)+13*8)*2)
35                 {
36                         if (txCount & 0x01)
37                                 // Manchester Code bilden
38                                 // zweite Hälfte des Taktes => Signal negieren
39                                 txNextOut = !txNextOut;
40                         else
41                         {
42                                 txNextOut = txData[txByte] & txBit;
43                                 if (txBit == 0x01)
44                                 {
45                                         txBit = 0x80;
46                                         txByte++;
47                                 }
```

```
48                                      else
49                                              txBit = txBit >> 1;
50                              }
51                      }
52                      else
53                              txNextOut = 0; // am Ende '0'
54
55                      if (txCount <= ((6+4)+13*8)*2)
56                              txCount++;
57                      else
58                      {
59                              txCount = 0;
60
61                              P1OUT &= ~0x03; // P1.0 + P1.1 clear
62                      }
63              }
64      }
65
66      unsigned short parity()
67      {
68              unsigned char p = 0;
69              unsigned char n;
70              for (n = 2; n < 12; n++)
71                      p ^= txData[n];
72
73              return p;
74      }
75
76      void transmit(unsigned short temperature,
77                    unsigned short battery,
78                    unsigned short supply)
79      {
80              if (txCount == 0)
81              {
82                      P1OUT &= ~0x01; // P1.0 clear
83                      P1OUT |= 0x02;  // P1.1 set
84
85                      txNextOut = 0;
86                      txCount = 1;
87                      txByte = 0;
88                      txBit = 0x80;
89
90                      txData[6] = (unsigned char)(temperature >> 8);
91                      txData[7] = (unsigned char)(temperature & 0x00FF);
92
93                      txData[8] = (unsigned char)(battery >> 8);
94                      txData[9] = (unsigned char)(battery & 0x00FF);
95
96                      txData[10] = (unsigned char)(supply >> 8);
97                      txData[11] = (unsigned char)(supply & 0x00FF);
98
99                      txData[12] = parity();
100             }
101     }
102
103     void initTransmit(unsigned long id)
104     {
105             // P1.0 Data Transmit
106             // P1.1 Select Transmit
107             P1OUT &= ~0x03; // P1.0 + P1.1 clear
108             P1SEL &= ~0x03; // P1.0 + P1.1 => IO Port
109             P1DIR |= 0x03;  // P1.0 + P1.1 => outputs
110
111             txCount = 0;
112             txNextOut = 0;
113
114             txData[2] = (unsigned char)(id >> 24);
```

```
115         txData[3] = (unsigned char)(id >> 16);
116         txData[4] = (unsigned char)(id >> 8);
117         txData[5] = (unsigned char)(id & 0x000000FF);
118 
119         txData[12] = parity();
120 }
121 
122 // ------------------------------------------------------------------------
123 void wait(void); // prototype for wait()
124 
125 void wait(void) // delay function - mit 2000 step => 18..20 ms
126 {
127         volatile int i;              // declare i as volatile int
128         for (i = 0; i < 2000; i++);  // repeat 2000 times
129 }
130 
131 // ------------------------------------------------------------------------
132 struct InfoBlock
133 {
134         unsigned long magicID;
135         unsigned long sonsorID;
136         // Daten für Kalibrierung
137         unsigned short vFactor;
138         unsigned short tOffset;
139         unsigned short tFactor;
140 };
141 
142 // daten im flash des Speichers ablegen!!!
143 const struct InfoBlock *const infoBlock = (const struct InfoBlock *)0x1000;
144 
145 const unsigned long magicID = 0x64490F01;
146 const unsigned char version = 0x01;
147 
148 // ------------------------------------------------------------------------
149 // Schieberegister für Zufallszahl
150 unsigned long sr(void);
151 
152 // Standartmäßige Initialisierung des Schieberegisters
153 static unsigned long sr_r = 0x08000000;
154 
155 unsigned long sr(void)
156 {
157         sr_r <<= 1;
158         sr_r |= (((sr_r>>28) ^ (sr_r>>19)) & 0x00000001);
159         sr_r &= 0x0FFFFFFF;
160 
161         return sr_r;
162 }
163 
164 void initSR(unsigned long initValue)
165 {
166         sr_r = initValue | 0x08000000;
167 }
168 
169 // ------------------------------------------------------------------------
170 unsigned short vArray[64];
171 unsigned char vIndex = 0;
172 unsigned char vCount = 0;
173 
174 unsigned short tArray[16];
175 unsigned char tIndex = 0;
176 unsigned char tCount = 0;
177 
178 // supply voltage factor
179 unsigned short vFactor = 5000;
180 // temperature offset
181 unsigned short tOffset = 673;
```

```
182 // temperature factor
183 unsigned short tFactor = 423;
184
185 // -------------------------------------------------------------------------
186 int main(void)
187 {
188         if (infoBlock->magicID != magicID)
189                 initTransmit(0x00000000);
190         else
191         {
192                 unsigned long sonsorID = infoBlock->sonsorID;
193                 sonsorID &= 0x000FFFFF;
194                 unsigned long versionID = version;
195                 versionID <<= 24;
196                 sonsorID |= versionID;
197
198                 if (infoBlock->vFactor != 0xFFFF)
199                 {
200                         vFactor = infoBlock->vFactor;
201                         sonsorID |= 0x00400000;
202                 }
203                 if (infoBlock->tOffset != 0xFFFF)
204                 {
205                         tOffset = infoBlock->tOffset;
206                         sonsorID |= 0x00200000;
207                 }
208                 if (infoBlock->tFactor != 0xFFFF)
209                 {
210                         tFactor = infoBlock->tFactor;
211                         sonsorID |= 0x00100000;
212                 }
213
214                 initSR(sonsorID<<1);
215                 initTransmit(sonsorID);
216         }
217         // avert Watchdog reset
218         WDTCTL = WDTPW + WDTHOLD;
219         // set DCO to 2MHz; RSEL=6 DOC=3 MOD=0 DCOR=0
220         DCOCTL = DCO1 | DCO0;  // DOC=3 MOD=0
221         BCSCTL1 = RSEL2 | RSEL1;  // RSEL=6
222         BCSCTL2 = 0;  // DCOR=0 (no external R)
223         // set up Timer A
224         //  want SMCLK
225         //  count up (sawtooth), period = COUNTS
226         TACTL = MC_1 | ID_0 | TASSEL_2;
227         // compare mode, interrupt enabled
228         TACCTL0 = CCIE;
229         // other CCTLx regs not used
230         // cycle length
231         TACCR0 = 200;
232         // other CCRx regs not used
233         // other CCTLx regs not used
234         // enable interrupts
235         eint();
236         // P2.0 Output für Steuerung StepUpConverter
237         P2OUT |= 0x01;  // P2.0 set
238         P2SEL &= ~0x01; // P2.0 IO Port
239         P2DIR |= 0x01;  // P2.0 output
240         // P3.6 Output für LED
241         P3OUT |= 0x40;  // P3.6 set - LED off
242         P3SEL &= ~0x40; // P3.6 IO Port
243         P3DIR |= 0x40;  // P3.6 output
244
245         unsigned short temperature;
246         unsigned short battery;
247         unsigned short supply;
248
```

```
249         unsigned short batteryMin = 5000;
250         unsigned short batteryMax = 0000;
251
252         unsigned short batteryMinMin = 5000;
253         unsigned short batteryMaxMax = 0000;
254
255         unsigned short supMinMin = 5000;
256         unsigned short supMaxMax = 0000;
257
258         unsigned short temMinMin = 5000;
259         unsigned short temMaxMax = 0000;
260
261         unsigned short batMinMin = 5000;
262         unsigned short batMaxMax = 0000;
263
264         unsigned short count = 0;
265
266         unsigned short timerP1 = 0;
267         unsigned short timerP2 = 0;
268
269         unsigned waitCount1 = 0;
270         unsigned waitCount2 = 0;
271
272         int x = 0;
273
274         for (;;)
275         {
276                 int n;
277
278                 unsigned int long temp;
279                 unsigned int long intDegC = 0;
280                 unsigned int long intSup = 0;
281                 unsigned int long intBat = 0;
282
283                 // disable interrupts
284                 dint();
285                 // P2.0 clear => StepUpConverter disabled
286                 P2OUT &= ~0x01;
287
288                 for (n = 0; n < 5; n++) x++;
289
290                 ADC10CTL0 &= ~ENC;              // reset ENC
291                 ADC10CTL1 = INCH_10 + ADC10DIV_2;   // Temp Sensor
292                 ADC10CTL0 = SREF_1 + ADC10SHT_1 + REFON + ADC10ON;
293
294                 for (n = 0; n < 5; n++) x++;
295
296                 ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
297
298                 while ((ADC10CTL1 & ADC10BUSY)) ;   // ADC10BUSY?
299
300                 temp = ADC10MEM;
301
302                 intDegC = ((temp - 673) * 423) / 128;
303
304                 if (temMinMin > temp)
305                         temMinMin = temp;
306                 if (temMaxMax < temp)
307                         temMaxMax = temp;
308
309                 ADC10CTL0 &= ~ENC;                      // reset ENC
310                 ADC10CTL1 = INCH_11 + ADC10DIV_2;       // Ch 11
311                 ADC10CTL0 = SREF_1 + ADC10SHT_1 + REFON + REF2_5V + ADC10ON;
312
313                 for (n = 0; n < 3; n++) x++;
314
315                 ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
```

```
316
317                 while ((ADC10CTL1 & ADC10BUSY)) ;   // ADC10BUSY?
318
319                 //  V = 2 * A11/1023*2500mV = x*5000/1023
320                 temp = ADC10MEM;
321                 intSup = (temp * 5000) / 1023;
322                 intSup = intSup * 4096 / 1000;
323
324                 if (supMinMin > temp)
325                         supMinMin = temp;
326                 if (supMaxMax < temp)
327                         supMaxMax = temp;
328
329                 ADC10CTL0 &= ~ENC;              // reset ENC
330                 ADC10CTL1 = INCH_2 + ADC10DIV_2;       // Ch 2
331                 ADC10CTL0 = SREF_1 + ADC10SHT_1 + REFON + REF2_5V + ADC10ON;
332
333                 for (n = 0; n < 3; n++) x++;
334                 ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
335
336                 while ((ADC10CTL1 & ADC10BUSY)) ;   // ADC10BUSY?
337
338                 //  V = 3/2 * A0/1023*2500mV = x*3750/1023
339                 temp = ADC10MEM;
340                 intBat = (temp * 3750 + 125) / 250;
341
342                 if (batMinMin > temp)
343                         batMinMin = temp;
344                 if (batMaxMax < temp)
345                         batMaxMax = temp;
346
347                 unsigned short mess[6];
348                 unsigned short indexMin = 0;
349                 unsigned short indexMax = 0;
350                 unsigned int long messAll = 0;
351
352
353                 for (n = 0; n < 6; n++)
354                 {
355                         ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
356                         timerP1 = TAR;
357                         while ((ADC10CTL1 & ADC10BUSY)) ;   // ADC10BUSY?
358                         timerP2 = TAR;
359                         mess[n] = ADC10MEM;
360                 }
361
362                 if (mess[0] < mess[1])
363                         indexMax = 1;
364                 else
365                         indexMin = 1;
366                 for (n = 2; n < 6; n++)
367                 {
368                         if (mess[n] < mess[indexMin])
369                                 indexMin = n;
370                         if (mess[n] > mess[indexMax])
371                                 indexMax = n;
372                 }
373                 for (n = 0; n < 6; n++)
374                 {
375                         if (n != indexMin  &&  n != indexMax)
376                                 messAll += mess[n];
377                 }
378
379                 if (batteryMinMin > mess[indexMin])
380                         batteryMinMin = mess[indexMin];
381                 if (batteryMaxMax < mess[indexMax])
382                         batteryMaxMax = mess[indexMax];
```

```
383
384                 ADC10CTL0 &= ~ENC;                      // reset ENC
385                 ADC10CTL1 = INCH_10 + ADC10DIV_2;       // Temp Sensor
386                 ADC10CTL0 = SREF_1 + ADC10SHT_1 + REFON + ADC10ON;
387                 ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
388
389                 // P2.0 set => StepUpConverter enabled
390                 P2OUT |= 0x01;
391                 // enable interrupts
392                 eint();
393
394                 temperature = intDegC;
395                 battery = intBat;
396                 supply = intSup;
397
398                 // Senden der Meßdaten
399                 // -------------------
400                 transmit(temperature, battery, supply);
401
402                 intBat = battery;
403                 intBat *= 1000;
404                 intBat /= 4096;
405                 if (batteryMin > intBat)
406                         batteryMin = intBat;
407                 if (batteryMax < intBat)
408                         batteryMax = intBat;
409
410                 // Phase für Einschalten der LED ca. 50 ms
411                 // ---------------------------------------
412                 // P3.6 Output für LED
413                 P3OUT &= ~0x40; // P3.6 clear - LED on
414
415                 for (n = 0; n < 3; n++)
416                         wait(); // call delay function
417
418                 // P3.6 Output für LED
419                 P3OUT |= 0x40;  // P3.6 set - LED off
420
421                 // Ruhephase - Länge mit Zufallsanteil
422                 // -----------------------------------
423                 unsigned max = sr() % 80 + 100;
424                 for (n = 0; n < max; n++)
425                         wait(); // call delay function
426
427                 // Zähler für Breakpoints
428                 // ----------------------
429                 count++;
430
431                 waitCount1++;
432
433                 if (count % 10 == 0)
434                         x++;
435
436                 if (count % 100 == 0)
437                         x++;
438
439                 if (count % 1000 == 0)
440                         x++;
441         }
442 }
```

B.1.2. Steuergerät

Listing B.2: Quellcode zur Ansteuerung des FLASH Speichers (Flash.c)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430-169STK onboard nand-flash functions
3          Used components:                MSP430-169STK
4          Date:                           12/01/2007
5          Last update:                    09/05/2008
6          Author:                         Tobias Krannich
7          Modified:                       Stephan Plaschke
8  -------------------------------------------------------------------------------
9
10 -------------------------------------------------------------------------------
11         Headerfiles
12 -------------------------------------------------------------------------------*/
13 #include <main.h>
14 #include <Flash.h>
15 #include <stdio.h>
16 #include <lcd16x2.h>
17 #include <msp430x16x.h>
18 #include <MSP_functions.h>
19
20
21
22 static unsigned char *FLASH_clear_frame[] = {
23                 "                ",
24                 "  Clear flash   "
25 };
26
27 //-----------------NAND FLASH-----------------------------------------------------
28 //-------------------------------------------------------------------------------
29 void init_Flash (void)
30 //-------------------------------------------------------------------------------
31 {
32         P2OUT = 0x07;               //NAND FLASH ini
33         P2DIR = 0x1F;
34 }
35
36 //-------------------------------------------------------------------------------
37 // pull flash pins to inactive condition
38 void Inactive_Flash(void)
39 //-------------------------------------------------------------------------------
40 {
41   IO_DIR=INPUT;         //IO is inputs
42   _CE_OFF;              //=1
43   _RE_OFF;              //=1
44   _WE_OFF;              //=1
45   ALE_OFF;              //=0
46   CLE_OFF;              //=0
47 }
48
49 //-------------------------------------------------------------------------------
50 void Write_Data(unsigned char a)
51 //-------------------------------------------------------------------------------
52 {
53   IO_DIR=OUTPUT;        //IO is outputs
54   _WE_ON;
55   OUT_PORT=a;
56   _WE_OFF;              //latch data
57 }
58
59 //-------------------------------------------------------------------------------
60 unsigned char Read_Data(void)
61 //-------------------------------------------------------------------------------
62 {
```

```
63    unsigned char f;
64    IO_DIR=INPUT;          //IO is inputs
65    _RE_ON;
66    f=IN_PORT;
67    _RE_OFF;               //read data
68    return (f);
69  }
70
71  //-------------------------------------------------------------------------------
72  unsigned char Programm_Bytes(unsigned char COL_ADD,
73                               unsigned char ROW_ADDL,
74                               unsigned char ROW_ADDH,
75                               unsigned char NUMBER)
76  {
77  //-------------------------------------------------------------------------------
78          unsigned char k, l;
79          Inactive_Flash();
80                  CLE_ON;
81                  _CE_ON;
82                  Write_Data(WRITE_PAGE);
83                  CLE_OFF;
84                  ALE_ON;
85          Write_Data(COL_ADD);
86          Write_Data(ROW_ADDL);
87          Write_Data(ROW_ADDH);
88          ALE_OFF;
89          for (k=0; k != NUMBER; k++)
90            {
91               l=WRITE_BUF[k];
92               Write_Data(l);
93            }
94          CLE_ON;
95          Write_Data(WRITE_AKN);
96          while ((R_B) == 0);
97          Write_Data(READ_STATUS);
98          CLE_OFF;
99          l = Read_Data();
100         Inactive_Flash();
101         return(l);
102 }
103
104 //-------------------------------------------------------------------------------
105 void Read_Bytes (unsigned char COL_ADD,
106                  unsigned char ROW_ADDL,
107                  unsigned char ROW_ADDH,
108                  unsigned char NUMBER)
109 {
110 //-------------------------------------------------------------------------------
111         unsigned char n, r;
112         Inactive_Flash();
113         CLE_ON;
114         _CE_ON;
115         Write_Data(READ_0);
116         CLE_OFF;
117         ALE_ON;
118         Write_Data(COL_ADD);
119         Write_Data(ROW_ADDL);
120         Write_Data(ROW_ADDH);
121         ALE_OFF;
122         while ((R_B) == 0);
123         for (n=0; n != NUMBER; n++)
124           {
125              r=Read_Data();
126              READ_BUF[n] = r;
127           }
128         Inactive_Flash();
129 }
```

```
130 |
131 |//-------------------------------------------------------------------------------
132 |void Data_Struct_To_Write_Buffer(DATA_STRUCT *pData)
133 |//-------------------------------------------------------------------------------
134 |{
135 |        WRITE_BUF[0] = pData->C0;
136 |        WRITE_BUF[1] = pData->C1;
137 |        WRITE_BUF[2] = pData->C2;
138 |        WRITE_BUF[3] = pData->C3;
139 |        WRITE_BUF[4] = pData->C4;
140 |        WRITE_BUF[5] = pData->C5;
141 |        WRITE_BUF[6] = pData->C6;
142 |        WRITE_BUF[7] = pData->C7;
143 |        WRITE_BUF[8] = pData->C8;
144 |        WRITE_BUF[9] = pData->C9;
145 |        WRITE_BUF[10] = pData->C10;
146 |        WRITE_BUF[11] = pData->C11;
147 |        WRITE_BUF[12] = pData->C12;
148 |        WRITE_BUF[13] = pData->C13;
149 |        WRITE_BUF[14] = pData->C14;
150 |        WRITE_BUF[15] = pData->C15;
151 |}
152 |
153 |//-------------------------------------------------------------------------------
154 |void Val_To_Data(DATA_STRUCT *pData, unsigned long i)
155 |//-------------------------------------------------------------------------------
156 |{
157 |        if(i >= 527)
158 |        {
159 |                _NOP();
160 |        }
161 |        pData->C0 = (unsigned char)((i>>0) & 0xFF);
162 |        pData->C1 = (unsigned char)((i>>8) & 0xFF);
163 |        pData->C2 = (unsigned char)((i>>16) & 0xFF);
164 |        pData->C3 = (unsigned char)((i>>24) & 0xFF);
165 |}
166 |
167 |//-------------------------------------------------------------------------------
168 |// Es wird der Block gelöscht, in dem die Adresse liegt
169 |unsigned char Erase_Flash (unsigned char BLOCK_ADDL, unsigned char BLOCK_ADDH)
170 |//-------------------------------------------------------------------------------
171 |{
172 |        unsigned char m;
173 |        Inactive_Flash();
174 |                CLE_ON;
175 |                _CE_ON;
176 |                Write_Data(ERASE_BLOCK);
177 |                CLE_OFF;
178 |                ALE_ON;
179 |        Write_Data(BLOCK_ADDL);
180 |        Write_Data(BLOCK_ADDH);
181 |        ALE_OFF;
182 |        CLE_ON;
183 |        Write_Data(ERASE_AKN);
184 |        while ((R_B) == 0);
185 |        Write_Data(READ_STATUS);
186 |        CLE_OFF;
187 |        m = Read_Data();
188 |        Inactive_Flash();
189 |        return (m);
190 |}
191 |
192 |//-------------------------------------------------------------------------------
193 |DATA_STRUCT * Read_Ring_Flash(void)
194 |//-------------------------------------------------------------------------------
195 |{
196 |        //unsigned short Index_Last = 0;
```

```
197         unsigned short ADD = 0;
198         unsigned char ADDH = 0;
199         unsigned char ADDL = 0;
200         unsigned char COL = 0;
201         static DATA_STRUCT Data = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
202         DATA_STRUCT *pData = &Data;
203
204         // Letztes Element im Ringbuffer ermitteln
205         if(Ring_Full == 0)      // Ringbuffer hatte noch keinen Überlauf
206         {
207                 // Keine Daten im Ringspeicher???
208                 if(Count_Ring == 0)
209                 {
210                         return NULL;
211                 }
212                 Index_Last = Index_Ring - Count_Ring;
213         }
214         else // Es gab schon einen Überlauf -> lese ab dem ersten Struct
215         {
216                 // Keine Daten im Ringspeicher???
217                 if((Count_Ring - 255) == 0)     //
218                 {
219                         return NULL;
220                 }
221                 Index_Last = Index_Ring - (Count_Ring + 1 - 256)/* - (Index_Ring%256)*/;
                                // +1 um auf 0x10000 zu kommen
222         }
223         Count_Ring--;
224
225         // Adresse berechnen
226         ADD = Index_Last / 16;
227         COL = Index_Last % 16;
228         ADDL = (unsigned char)(ADD & 0xFF);
229         ADDH = (unsigned char)((ADD>>8) & 0xFF);
230
231         // Speicher auslesen
232         Read_Bytes(COL*16,ADDL,ADDH,BUF_SIZE);
233
234         // Read_BUF in das Datenstruct schreiben
235         pData->C0  = READ_BUF[0];
236         pData->C1  = READ_BUF[1];
237         pData->C2  = READ_BUF[2];
238         pData->C3  = READ_BUF[3];
239         pData->C4  = READ_BUF[4];
240         pData->C5  = READ_BUF[5];
241         pData->C6  = READ_BUF[6];
242         pData->C7  = READ_BUF[7];
243         pData->C8  = READ_BUF[8];
244         pData->C9  = READ_BUF[9];
245         pData->C10 = READ_BUF[10];
246         pData->C11 = READ_BUF[11];
247         pData->C12 = READ_BUF[12];
248         pData->C13 = READ_BUF[13];
249         pData->C14 = READ_BUF[14];
250         pData->C15 = READ_BUF[15];
251
252         return pData;
253 }
254
255 //---------------------------------------------------------------------------
256 //---------------------------------------------------------------------------
257 //              These functions are modified/created by Stephan Plaschke
258 //---------------------------------------------------------------------------
259 //---------------------------------------------------------------------------
260
261 //---------------------------------------------------------------------------
262 void Erase_Flash_All(void)
```

```
263 //-------------------------------------------------------------------------------
264 {
265         int i = 0;
266         unsigned char j=0;
267         unsigned char ADDL;
268         unsigned char ADDH;
269 
270         // show splash screen on LCD and send via UART
271         LCD_send_cmd(LCD_LINE1);
272         LCD_send_text(FLASH_clear_frame[1]);
273         UART0_send('.');
274 
275         for(i=0; i<0x1000; i++)
276         {
277                 //ADD = i * 16;
278                 ADDL = (unsigned char)(i & 0xFF);
279                 ADDH = (unsigned char)((i>>8) & 0xFF);
280                 Erase_Flash(ADDL,ADDH);
281 
282                 // show splash screen on LCD and send via UART
283                 if (!(i%200))
284                 {
285                         if(j^=0x01)
286                         {
287                                 LCD_send_cmd(LCD_LINE1);
288                                 LCD_send_text(FLASH_clear_frame[1]);
289                                 UART0_send('.');
290                         }
291                         else
292                         {
293                                 LCD_send_cmd(LCD_LINE1);
294                                 LCD_send_text(FLASH_clear_frame[0]);
295                         }
296                 }
297         }
298         // set all index to the first frame in flash
299         Count_Ring = 0;
300         Ring_Full = 0;
301         // set pointer on the address 256, first block used for controll data
302         Index_Ring = 256;
303         Index_Last = 256;
304 
305         LCD_send_cmd(LCD_LINE1);
306         LCD_send_text(FLASH_clear_frame[0]);
307         UART0_send_text("\r\n");
308 
309         RADIO_init_frame();                     // set first frame
310 }
311 
312 //-------------------------------------------------------------------------------
313 void Write_Ring_Flash(DATA_STRUCT *pData)
314 //-------------------------------------------------------------------------------
315 {
316         unsigned char ADDL;
317         unsigned char ADDH;
318         unsigned short ADD;
319         unsigned char COL;
320 
321 
322         ADD = Index_Ring / 16;
323         COL = Index_Ring % 16;
324         ADDL = (unsigned char)(ADD & 0xFF);
325         ADDH = (unsigned char)((ADD>>8) & 0xFF);
326 
327         // Lösche den nächsten Block
328         if( (Index_Ring%256) == 0)
329         {
```

```
330                 Erase_Flash(ADDL,ADDH);
331         }
332
333         Data_Struct_To_Write_Buffer(pData);
334         Programm_Bytes(COL*16,ADDL,ADDH,BUF_SIZE);
335
336         if(Count_Ring < 65279) // address space: 65535 - 256
337         {
338                 Count_Ring++;
339         }
340         else
341         {
342                 Ring_Full = 1;
343         }
344
345         Index_Ring++;
346         if(Index_Ring >= 65535)
347         {
348                 Index_Ring = 256; // set pointer on the address 256, first block used for
                        controll data
349         }
350 }
351
352 //---------------------------------------------------------------------------
353 void FLASH_Write_Controll_Data(void)
354 //---------------------------------------------------------------------------
355 {
356         Erase_Flash(0,0);
357
358         WRITE_BUF[0]  = (unsigned char)(Index_Ring >> 8);
359         WRITE_BUF[1]  = (unsigned char)Index_Ring;
360         WRITE_BUF[2]  = (unsigned char)(Index_Last >> 8);
361         WRITE_BUF[3]  = (unsigned char)Index_Last;
362         WRITE_BUF[4]  = (unsigned char)(Count_Ring >> 8);
363         WRITE_BUF[5]  = (unsigned char)Count_Ring;
364         WRITE_BUF[6]  = Ring_Full;
365         WRITE_BUF[7]  = SENSOR_active;
366         WRITE_BUF[8]  = SENSOR_address[0];
367         WRITE_BUF[9]  = SENSOR_address[1];
368         WRITE_BUF[10] = SENSOR_address[2];
369         WRITE_BUF[11] = SENSOR_address[3];
370         WRITE_BUF[12] = SENSOR_address[4];
371         WRITE_BUF[13] = SENSOR_address[5];
372         WRITE_BUF[14] = SENSOR_address[6];
373         WRITE_BUF[15] = SENSOR_address[7];
374
375         Programm_Bytes(0,0,0,BUF_SIZE);
376
377         WRITE_BUF[0]  = SENSOR_address[8];
378         WRITE_BUF[1]  = SENSOR_address[9];
379         WRITE_BUF[2]  = SENSOR_address[10];
380         WRITE_BUF[3]  = SENSOR_address[11];
381         WRITE_BUF[4]  = SENSOR_address[12];
382         WRITE_BUF[5]  = SENSOR_address[13];
383         WRITE_BUF[6]  = SENSOR_address[14];
384         WRITE_BUF[7]  = SENSOR_address[15];
385         WRITE_BUF[8]  = SENSOR_address[16];
386         WRITE_BUF[9]  = SENSOR_address[17];
387         WRITE_BUF[10] = SENSOR_address[18];
388         WRITE_BUF[11] = SENSOR_address[19];
389         WRITE_BUF[12] = SENSOR_address[20];
390         WRITE_BUF[13] = SENSOR_address[21];
391         WRITE_BUF[14] = SENSOR_address[22];
392         WRITE_BUF[15] = SENSOR_address[23];
393
394         Programm_Bytes(0,1,0,BUF_SIZE);
395
```

```
396         WRITE_BUF[0]  = SENSOR_address[24];
397         WRITE_BUF[1]  = SENSOR_address[25];
398         WRITE_BUF[2]  = SENSOR_address[26];
399         WRITE_BUF[3]  = SENSOR_address[27];
400         WRITE_BUF[4]  = SENSOR_address[28];
401         WRITE_BUF[5]  = SENSOR_address[29];
402         WRITE_BUF[6]  = SENSOR_address[30];
403         WRITE_BUF[7]  = SENSOR_address[31];
404         WRITE_BUF[8]  = SENSOR_address[32];
405         WRITE_BUF[9]  = SENSOR_address[33];
406         WRITE_BUF[10] = SENSOR_address[34];
407         WRITE_BUF[11] = SENSOR_address[35];
408         WRITE_BUF[12] = SENSOR_address[36];
409         WRITE_BUF[13] = SENSOR_address[37];
410         WRITE_BUF[14] = SENSOR_address[38];
411         WRITE_BUF[15] = SENSOR_address[39];
412
413         Programm_Bytes(0,2,0,BUF_SIZE);
414 }
415
416 //---------------------------------------------------------------------------
417 void FLASH_Read_Controll_Data(void)
418 //---------------------------------------------------------------------------
419 {
420         // Speicher auslesen
421         Read_Bytes(0,0,0,BUF_SIZE);
422
423         Index_Ring                  = (READ_BUF[0] << 8) | READ_BUF[1];
424         Index_Last                  = (READ_BUF[2] << 8) | READ_BUF[3];
425         Count_Ring          = (READ_BUF[4] << 8) | READ_BUF[5];
426         Ring_Full                   = READ_BUF[6];
427         SENSOR_active       = READ_BUF[7];
428         SENSOR_address[0]   = READ_BUF[8];
429         SENSOR_address[1]   = READ_BUF[9];
430         SENSOR_address[2]   = READ_BUF[10];
431         SENSOR_address[3]   = READ_BUF[11];
432         SENSOR_address[4]   = READ_BUF[12];
433         SENSOR_address[5]   = READ_BUF[13];
434         SENSOR_address[6]   = READ_BUF[14];
435         SENSOR_address[7]   = READ_BUF[15];
436
437         // Speicher auslesen
438         Read_Bytes(0,1,0,BUF_SIZE);
439
440         SENSOR_address[8]   = READ_BUF[0];
441         SENSOR_address[9]   = READ_BUF[1];
442         SENSOR_address[10]  = READ_BUF[2];
443         SENSOR_address[11]  = READ_BUF[3];
444         SENSOR_address[12]  = READ_BUF[4];
445         SENSOR_address[13]  = READ_BUF[5];
446         SENSOR_address[14]  = READ_BUF[6];
447         SENSOR_address[15]  = READ_BUF[7];
448         SENSOR_address[16]  = READ_BUF[8];
449         SENSOR_address[17]  = READ_BUF[9];
450         SENSOR_address[18]  = READ_BUF[10];
451         SENSOR_address[19]  = READ_BUF[11];
452         SENSOR_address[20]  = READ_BUF[12];
453         SENSOR_address[21]  = READ_BUF[13];
454         SENSOR_address[22]  = READ_BUF[14];
455         SENSOR_address[23]  = READ_BUF[15];
456
457         // Speicher auslesen
458         Read_Bytes(0,2,0,BUF_SIZE);
459
460         SENSOR_address[24]  = READ_BUF[0];
461         SENSOR_address[25]  = READ_BUF[1];
462         SENSOR_address[26]  = READ_BUF[2];
```

```
463         SENSOR_address[27] = READ_BUF[3];
464         SENSOR_address[28] = READ_BUF[4];
465         SENSOR_address[29] = READ_BUF[5];
466         SENSOR_address[30] = READ_BUF[6];
467         SENSOR_address[31] = READ_BUF[7];
468         SENSOR_address[32] = READ_BUF[8];
469         SENSOR_address[33] = READ_BUF[9];
470         SENSOR_address[34] = READ_BUF[10];
471         SENSOR_address[35] = READ_BUF[11];
472         SENSOR_address[36] = READ_BUF[12];
473         SENSOR_address[37] = READ_BUF[13];
474         SENSOR_address[38] = READ_BUF[14];
475         SENSOR_address[39] = READ_BUF[15];
476 }
477
478
479 /*-------------------------------------------------------------------------------
480         read data at startup
481 -------------------------------------------------------------------------------*/
482 void FLASH_read_at_startup(void)
483 /*-------------------------------------------------------------------------------*/
484 {
485         LCD_send_cmd(LCD_LINE1);
486         LCD_send_text("Read contr.data ");
487         LCD_send_cmd(LCD_LINE2);
488         LCD_send_text("YES    NO       ");
489
490         while(1)
491         {
492                 if(BATMON_control_reg & B1)     // Button 1 = YES, read controll data
                        from flash
493                 {
494                         UART0_send_text("Read data\r\n");
495                         FLASH_Read_Controll_Data();
496                         BATMON_control_reg &= ~(B1|B2|B3);      // clear button state
497                         break;
                            // exit while loop
498                 }
499                 if(BATMON_control_reg & B2)                             // Button 2 = NO,
                         write init sequence
500                 {
501                         RADIO_init_frame();                                     // write
                            first frame with actual time and 0 as data
502                         BATMON_control_reg &= ~(B1|B2|B3);      // clear button state
503                         break;
                            // exit while loop
504                 }
505         }
506 }
```

Listing B.3: Header zur Ansteuerung des FLASH Speichers (Flash.h)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430-169STK onboard nand-flash functions header
3          Used components:                MSP430-169STK
4          Date:                           12/01/2007
5          Last update:                    09/05/2008
6          Author:                         Tobias Krannich
7          Modified:                       Stephan Plaschke
8  -------------------------------------------------------------------------------*/
9
10 /*-------------------------------------------------------------------------------
11         Defines
12 -------------------------------------------------------------------------------*/
13 #ifndef FLASH_H_
14 #define FLASH_H_
```

```
15
16 #define MAX_BLOCK_NUMB  1024
17
18 #define TRANS_LDY               50
19 #define WRITE_DLY               400
20 #define ERASE_DLY               4000
21
22 #define OUT_PORT                P5OUT
23 #define IN_PORT             P5IN
24 #define IO_DIR                  P5DIR
25
26 #define _CE_ON                  (P2OUT &= ~BIT0)
27 #define _CE_OFF                 (P2OUT |= BIT0)
28 #define _RE_ON                  (P2OUT &= ~BIT1)
29 #define _RE_OFF                 (P2OUT |= BIT1)
30 #define _WE_ON                  (P2OUT &= ~BIT2)
31 #define _WE_OFF                 (P2OUT |= BIT2)
32
33 #define ALE_ON                  (P2OUT |= BIT3)
34 #define ALE_OFF                 (P2OUT &= ~BIT3)
35 #define CLE_ON                  (P2OUT |= BIT4)
36 #define CLE_OFF                 (P2OUT &= ~BIT4)
37
38 #define R_B             (P2IN & BIT7)
39 #define DALLAS          (P2IN & BIT5)
40
41 #define READ_SPARE              0x50
42 #define READ_0                  0x00
43 #define READ_1                  0x01
44 #define READ_STATUS             0x70
45
46 #define WRITE_PAGE              0x80
47 #define WRITE_AKN               0x10
48
49 #define ERASE_BLOCK             0x60
50 #define ERASE_AKN               0xD0
51
52 #define DEV_ID                  0x90
53
54 #define SAMSUNG_ID              0xECE6
55
56 #define INPUT           0
57 #define OUTPUT          0xff
58 #define BUF_SIZE        16
59
60 #define ZEICHEN 0xAA
61
62 /*-------------------------------------------------------------------------------
63         Global variables
64 -------------------------------------------------------------------------------*/
65 extern unsigned char  SENSOR_address[40];       // array with sensor address
66 extern unsigned char  SENSOR_active;                    // active sensor count
67
68 extern unsigned char  WRITE_BUF [BUF_SIZE];
69 extern unsigned char  READ_BUF [BUF_SIZE];
70 extern unsigned short Index_Ring;
71 extern unsigned short Count_Ring;
72 extern unsigned char  Ring_Full;
73 extern unsigned short Index_Last;
74
75 /*-------------------------------------------------------------------------------
76         Structures
77 -------------------------------------------------------------------------------*/
78 typedef struct{
79         unsigned char C0;
80         unsigned char C1;
81         unsigned char C2;
```

```
82          unsigned char C3;
83          unsigned char C4;
84          unsigned char C5;
85          unsigned char C6;
86          unsigned char C7;
87          unsigned char C8;
88          unsigned char C9;
89          unsigned char C10;
90          unsigned char C11;
91          unsigned char C12;
92          unsigned char C13;
93          unsigned char C14;
94          unsigned char C15;
95          } DATA_STRUCT;
96
97  /*-------------------------------------------------------------------------------
98          Prototypes
99  -------------------------------------------------------------------------------*/
100 void init_Flash (void);
101 void Inactive_Flash(void);
102 void Write_Data(unsigned char a);
103 unsigned char Programm_Bytes(unsigned char COL_ADD,
104                              unsigned char ROW_ADDL,
105                              unsigned char ROW_ADDH,
106                              unsigned char NUMBER);
107 unsigned char Read_Data(void);
108 void Read_Bytes (unsigned char COL_ADD,
109                  unsigned char ROW_ADDL,
110                  unsigned char ROW_ADDH,
111                  unsigned char NUMBER);
112 unsigned char Erase_Flash (unsigned char BLOCK_ADDL, unsigned char BLOCK_ADDH);
113 void Erase_Flash_All(void);
114 void Write_Ring_Flash(DATA_STRUCT *);
115 void Data_Struct_To_Write_Buffer(DATA_STRUCT *);
116 void Val_To_Data(DATA_STRUCT *, unsigned long);
117 DATA_STRUCT * Read_Ring_Flash(void);
118 void FLASH_Write_Controll_Data(void);
119 void FLASH_Read_Controll_Data(void);
120 void MemTest(void);
121 void FLASH_read_at_startup(void);
122
123 #endif /*FLASH_H_*/
```

Listing B.4: Quellcode der Interrupthandler (ISR.c)

```
1   /*-------------------------------------------------------------------------------
2           Project:                        ISR functions
3           Used components:                MSP430-169STK
4           Date:                           12/15/2007
5           Last update:                    04/04/2008
6           Author:                         Stephan Plaschke
7   -------------------------------------------------------------------------------
8
9   -------------------------------------------------------------------------------
10          Headerfiles
11  -------------------------------------------------------------------------------*/
12  #include <ISR.h>
13  #include <main.h>
14  #include <signal.h>
15  #include <lcd16x2.h>
16  #include <msp430x16x.h>
17  #include <MSP_functions.h>
18
19
20  /*-------------------------------------------------------------------------------
21          Global variables
```

```
22 |------------------------------------------------------------------------------*/
23 |extern unsigned char BUTTON_state;                       // actual button state
24 |
25 |unsigned char BATMON_control_reg;
26 |extern unsigned char RADIO_rx_state;                     // data receive state
27 |extern unsigned char global_string[MAX_STRING_LENGTH];
28 |
29 |extern RADIO_FRAME_STRUCT       RADIO_frame;            // received data frame
30 |
31 |extern unsigned char SENSOR_addr_req[5];                // required sensor addresses
32 |extern unsigned char SENSOR_address[40];                // stored sensor addresses
33 |extern unsigned char SENSOR_last_data[8][40];
34 |
35 |/*------------------------------------------------------------------------------
36 |        Timer A0 interrupt service routine, Capture/compare0 interrupt
37 |------------------------------------------------------------------------------*/
38 |interrupt (TIMERA0_VECTOR) TIMER_A0(void)
39 |/*------------------------------------------------------------------------------*/
40 |{
41 |        static unsigned int RADIO_capture_old, RADIO_capture_act, RADIO_capture_interval;
42 |
43 |        static unsigned int RADIO_bit_length, RADIO_one_bit_length, RADIO_two_bit_length;
44 |        static unsigned int RADIO_three_bit_length, RADIO_four_bit_length;
45 |
46 |        static unsigned char RADIO_capture_counter;
47 |
48 |
49 |        // compare values for the received address in global array, byte = position in
           array
50 |        unsigned char address_bit, address_byte;
51 |        unsigned char i, j, frame_error;
52 |
53 |        RADIO_capture_old = RADIO_capture_act;
54 |        RADIO_capture_act = CCR0;
55 |
56 |        if (RADIO_capture_act < RADIO_capture_old)
                                                  /* overflow occured     */
57 |                RADIO_capture_interval = ((65535-RADIO_capture_old) + RADIO_capture_act);
                           /* calculate difference */
58 |        else
                                                                                /* no
                overflow occured  */
59 |                RADIO_capture_interval = RADIO_capture_act - RADIO_capture_old;
                                  /* calculate difference */
60 |
61 |
62 |        switch(RADIO_rx_state)
63 |        {
64 |        case SE_SEQ_PREPARE:                     // first interrupt, nothing to do
65 |                RADIO_frame.frame_bit_counter = 0;              // actual bit position
66 |                RADIO_frame.frame_byte_counter = 0;             // actual byte in frame
67 |                BATMON_control_reg &= !VALID_DATA_RECEIVED;     // clear global value
68 |                RADIO_rx_state = SE_SEQ_START;
69 |                break;  // exit switch-case block
70 |
71 |        case SE_SEQ_START:                               // look if start sequence
               detected
72 |                if((RADIO_capture_interval < START_SEQ_LENGTH_MAX)
73 |                        && (RADIO_capture_interval > START_SEQ_LENGTH_MIN))
74 |                {
75 |                        RADIO_rx_state = SE_SEQ_PRE_WAIT;
76 |                        RADIO_capture_counter   = 14;   // waitcounter 14 bit until
                               second byte received
77 |                }
78 |                break;  // exit switch-case block
79 |
80 |        case SE_SEQ_PRE_WAIT:                    // check next 14 bits for valid length
```

```
81                  /*if((RADIO_capture_interval < TWO_BIT_LENGTH)
82                          && (RADIO_capture_interval > ONE_BIT_LENGTH))
83                  {
84                          RADIO_capture_counter--;
85                          if(RADIO_capture_counter == 0)
86                                  RADIO_rx_state = SE_SEQ_DATA_START;
87                  }
88                  else
89                          RADIO_rx_state = SE_SEQ_PREPARE;
90                  */
91                  RADIO_capture_counter--;
92
93                  if (RADIO_capture_counter == 14)
94                          RADIO_bit_length = RADIO_capture_interval;
95                  else if (RADIO_capture_counter == 0)
96                  {
97                          RADIO_bit_length += RADIO_capture_interval;
98                          RADIO_bit_length /= 2;
99
100                         RADIO_one_bit_length   = RADIO_bit_length - 50;
101                         RADIO_two_bit_length   = RADIO_bit_length + 50;
102                         RADIO_three_bit_length = RADIO_bit_length * 2 - 50;
103                         RADIO_four_bit_length  = RADIO_bit_length * 2 + 50;
104
105                         RADIO_rx_state = SE_SEQ_DATA_START;
106                 }
107                 else
108                 {
109                         RADIO_bit_length += RADIO_capture_interval;
110                         RADIO_bit_length /= 2;
111                 }
112
113                 break;  // exit switch-case block
114
115         case SE_SEQ_DATA_START:                 // end of second byte, check second byte
116                 if((RADIO_capture_interval < RADIO_four_bit_length)
117                         && (RADIO_capture_interval > RADIO_three_bit_length))
118                 {
119                         RADIO_rx_state = SE_SEQ_RXDATA;
120                         RADIO_frame.frame_bit_counter = 1;              // actual bit
                                position
121                         RADIO_frame.frame_byte_counter = 0;             // actual byte in
                                 frame
122                         RADIO_frame.frame_last_bit = 0;                 // last bit state
                                , equals actual bit
123                         RADIO_frame.frame_byte[0] = 0x00;               // set actual bit
124                 }
125                 else
126                         RADIO_rx_state = SE_SEQ_PREPARE;                // second byte
                                not 0x01, error
127                 break;  // exit switch-case block
128
129         case SE_SEQ_RXDATA:                             // receive data, for this cases
                see documentation
130                 if(RADIO_capture_interval < RADIO_one_bit_length)       // intervall to
                        short, exit
131                 {
132                         RADIO_rx_state = SE_SEQ_PREPARE;
133                         break;  // exit switch-case block
134                 }
135
136                 else if(RADIO_capture_interval < RADIO_two_bit_length)  // same value
                        like the one before
137                 {
138                         RADIO_frame.frame_byte_counter = RADIO_frame.frame_bit_counter /
                                8;
139                         RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] <<= 1;
```

```
140                         RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] |=
                                RADIO_frame.frame_last_bit;
141                         RADIO_frame.frame_bit_counter++;
142                 }
143
144                 else if(RADIO_capture_interval < RADIO_three_bit_length)
145                 {
146                         if(RADIO_frame.frame_last_bit)  // a one and a following zero
                                detected
147                         {
148                                 RADIO_frame.frame_byte_counter = RADIO_frame.
                                        frame_bit_counter / 8;
149                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter]
                                        <<= 1;
150                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] |=
                                         0x01;
151                                 RADIO_frame.frame_bit_counter++;
152
153                                 RADIO_frame.frame_byte_counter = RADIO_frame.
                                        frame_bit_counter / 8;
154                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter]
                                        <<= 1;
155                                 RADIO_frame.frame_last_bit = 0;
156                                 RADIO_frame.frame_bit_counter++;
157                         }
158                         else    // a one detected
159                         {
160                                 RADIO_frame.frame_byte_counter = RADIO_frame.
                                        frame_bit_counter / 8;
161                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter]
                                        <<= 1;
162                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] |=
                                         0x01;
163                                 RADIO_frame.frame_last_bit = 1;
164                                 RADIO_frame.frame_bit_counter++;
165                         }
166                 }
167
168                 else if(RADIO_capture_interval < RADIO_four_bit_length) // a one with a
                        following zero detected
169                 {
170                         RADIO_frame.frame_byte_counter = RADIO_frame.frame_bit_counter /
                                8;
171                         RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] <<= 1;
172                         RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] |= 0x01;
173                         RADIO_frame.frame_bit_counter++;
174
175                         RADIO_frame.frame_byte_counter = RADIO_frame.frame_bit_counter /
                                8;
176                         RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] <<= 1;
177                         RADIO_frame.frame_bit_counter++;
178                 }
179
180                 else    // intervall to long, exit with error
181                 {
182                         if (RADIO_frame.frame_bit_counter == 87)        // last byte(CRC)
                                 is odd
183                         {
184                                 RADIO_frame.frame_byte_counter = RADIO_frame.
                                        frame_bit_counter / 8;
185                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter]
                                        <<= 1;
186                                 RADIO_frame.frame_byte[RADIO_frame.frame_byte_counter] |=
                                         0x01;
187                                 RADIO_frame.frame_bit_counter++;
188                         }
189                         else
```

```
190                     {
191                             RADIO_rx_state = SE_SEQ_PREPARE;
192                             break;  // exit switch-case block
193                     }
194             }
195
196             if (RADIO_frame.frame_bit_counter == 88)        // end of bitstream
197             {
198                     i=0;    // init address counter
199                     do              // check if address valid
200                     {
201                             if (SENSOR_address[i] == RADIO_frame.frame_byte[3])
202                                     break;                                  // exit
                                            do-while loop
203                             i++;
204                     }
205                     while(i<40);
206
207                     if (i<40)       // if sensor address valid
208                     {
209                             j = 0;
210                             frame_error = 0;
211
212                             while(j<11)             // check CRC
213                             {
214                                     frame_error = frame_error ^ RADIO_frame.
                                            frame_byte[j];
215                                     j++;
216                             }
217
218                             if (frame_error)        // CRC not valid, exit
219                             {
220                                     // CRC not valid return to the beginning
221                                     RADIO_rx_state = SE_SEQ_PREPARE;
222                                     break;  // exit switch-case block
223                             }
224
225                             // calculate position in sensor_addr_req array
226                             //address_bit = (SENSOR_address[i] - 1) % 8 ;
227                             //address_byte = (SENSOR_address[i] - 1) / 8;
228                             address_bit = i % 8 ;
229                             address_byte = i / 8;
230
231                             // check if sensor data needed
232                             if ((SENSOR_addr_req[address_byte] >> address_bit) & 0x01
                                    )
233                             {
234                                     // clear required bit value
235                                     SENSOR_addr_req[address_byte] &= ~(0x01 <<
                                            address_bit);
236
237                                     for (j=0; j<8; j++)
238                                     {
239                                             SENSOR_last_data[j][i] = RADIO_frame.
                                                    frame_byte[j+3];
240                                     }
241
242                                     RADIO_disable();
                                                                     // disable radio unit
243                                     BATMON_control_reg |= VALID_DATA_RECEIVED;
                                            // set global value
244                             }
245                     }
246                     // Receiving frame completed(valid or not), return to the
                            beginning
247                     RADIO_rx_state = SE_SEQ_PREPARE;
248             }
```

```
249                 break;  // exit switch-case block
250
251         default: RADIO_rx_state = SE_SEQ_PREPARE;
252                 break;  // exit switch-case block
253         }
254
255         TACCTL0 &= ~(CCIFG|COV);        /* reset timer occured flag, for up mode
                */
256     /*CCR0 += 410;      */              /* add value to timer, continous mode   */
257 }
258
259 /*-------------------------------------------------------------------------------
260         Timer A0 interrupt service routine
261 -------------------------------------------------------------------------------*/
262 interrupt (TIMERA1_VECTOR) TIMER_A1(void)
263 /*-------------------------------------------------------------------------------*/
264 {
265         switch(TAIV)
266         {
267                 /* Capture/compare1 interrupt   */
268                 case 0x02:
269                                 TACCTL1 &= ~(CCIFG|COV);        /* reset timer occured
                                    flag     */
270                                 /*TACCR1 += 410;        */              /* add value to
                                    timer, continous mode   */
271                                 break;
272
273                 /* Capture/compare2 interrupt   */
274                 case 0x04:
275                                 TACCTL2 &= ~(CCIFG|COV);        /* reset timer occured
                                    flag     */
276                                 /*TACCR2 += 410;        */              /* add value to
                                    timer, continous mode   */
277                                 break;
278
279                 /* Timeroverflow interrupt      */
280                 case 0x0A: TACTL &= ~TAIFG;             /* reset timer occured flag
                        */
281                                 break;
282         }
283
284 }
285
286 /*-------------------------------------------------------------------------------
287         Timer B0 interrupt service routine
288 -------------------------------------------------------------------------------*/
289 interrupt (TIMERB0_VECTOR) TIMER_B0(void)
290 /*-------------------------------------------------------------------------------*/
291 {
292         static char TB0_ct0, TB0_ct1;
293         char TB0_i;
294
295         TB0_i = BUTTON_state ^ BUTTON_INPUT;                    // input changed ?
296         TB0_ct0 = ~( TB0_ct0 & TB0_i );                                 // reset or count
                ct0
297         TB0_ct1 = (TB0_ct0 ^ (TB0_ct1 & TB0_i));                // reset or count ct1
298         TB0_i &= (TB0_ct0 & TB0_ct1);                                   // count until
                roll over
299         BUTTON_state ^= TB0_i;                                                  // toggle
                debounced state
300         BATMON_control_reg |= BUTTON_state & TB0_i;             // only 0->1:key pressing
                detect
301
302         TBCCTL0 &= ~CCIFG;              // reset timer occured flag, for up mode
303     //TBCCR0 += 410;                    // add value to timer, continous mode
304 }
305
```

```
306 /*-------------------------------------------------------------------------------
307         Timer A0 interrupt service routine
308 -------------------------------------------------------------------------------*/
309 interrupt (TIMERB1_VECTOR) TIMER_B1(void)
310 /*-------------------------------------------------------------------------------*/
311 {
312         switch(TBIV)
313         {
314                 /* Capture/compare1 interrupt   */
315                 case 0x02:
316
317                                 TBCCTL1 &= ~(CCIFG|COV);        /* reset timer occured
                                        flag     */
318                                 //TBCCR1 += 2;                  /* add value to timer,
                                        continous mode   */
319                                 break;
320
321                 /* Capture/compare2 interrupt   */
322                 case 0x04:
323                                 TBCCTL2 &= ~(CCIFG|COV);        /* reset timer occured
                                        flag     */
324                                 /*TBCCR2 += 410;        */              /* add value to
                                        timer, continous mode   */
325                                 break;
326
327                 /* Timeroverflow interrupt      */
328                 case 0x0E:
329                                 TBCTL &= ~TAIFG;                /* reset timer occured
                                        flag     */
330                                 break;
331         }
332
333 }
334
335 /*-------------------------------------------------------------------------------
336         UART0 RX interrupt service routine
337 -------------------------------------------------------------------------------*/
338 interrupt (USART0RX_VECTOR) USART0_RX(void)
339 /*-------------------------------------------------------------------------------*/
340 {
341         static unsigned char rx_string[MAX_STRING_LENGTH];
342         static unsigned char i;
343
344         switch (RXBUF0)
345         {
346         case (RETURN):                                  // if carriage return jump to
            next line first position and store string global
347                 UART0_send(NEWLINE);
348                 UART0_send(RETURN);
349
350                 rx_string[i] = NULLTERMINATOR;          // terminate string
351                 i++;
352
353                 for (; i>0; i--)                                        // copy string
354                 {
355                         global_string[i-1] = rx_string[i-1];
356                 };
357                 BATMON_control_reg |= GLOBAL_STRING_RECEIVED;   // set control register
                        to global string received
358                 break;
359
360         case ('%'):                                             // if '%' jump to next
            line first position and store string global
361                 UART0_send(NEWLINE);
362                 UART0_send(RETURN);
363
364                 rx_string[i] = NULLTERMINATOR;          // terminate string
```

```
365                 i++;
366
367                 for (; i>0; i--)                                        // copy string
368                 {
369                         global_string[i-1] = rx_string[i-1];
370                 };
371                 BATMON_control_reg |= GLOBAL_STRING_RECEIVED;   // set control register
                        to global string received
372                 break;
373
374         case (BACKSPACE):                                   // backspace, delete last char
375                 UART0_send(BACKSPACE);
376                 UART0_send(SPACE);
377                 UART0_send(BACKSPACE);
378                 i--;
379                 break;
380
381         default:                                                    // echo
382                 if(i<(MAX_STRING_LENGTH-1))     // string length reached?
383                 {
384                         UART0_send(RXBUF0);     // string length not reached store
                            received char
385                         rx_string[i] = RXBUF0;
386                         i++;
387                 }
388                 else                                                                // string
                        length reached, save string in global variable
389                 {
390                         rx_string[i] = NULLTERMINATOR;  // terminate string
391                         for (; i>0; i--)                                // copy string
392                         {
393                                 global_string[i-1] = rx_string[i-1];
394                         };
395                         BATMON_control_reg |= GLOBAL_STRING_RECEIVED;   // set control
                            register to global string received
396                 }
397                 break;
398         }
399 }
400
401 /*-------------------------------------------------------------------------------
402         UART0 TX interrupt service routine
403 -------------------------------------------------------------------------------*/
404 interrupt (USART0TX_VECTOR) USART0_TX(void)
405 /*-------------------------------------------------------------------------------*/
406 {
407
408 }
409
410 /*-------------------------------------------------------------------------------
411         PORT1 interrupt service routine
412 -------------------------------------------------------------------------------*/
413 interrupt (PORT1_VECTOR) PORT_1(void)
414 /*-------------------------------------------------------------------------------*/
415 {
416
417 }
418
419 /*-------------------------------------------------------------------------------
420         PORT2 interrupt service routine
421 -------------------------------------------------------------------------------*/
422 interrupt (PORT2_VECTOR) PORT_2(void)
423 /*-------------------------------------------------------------------------------*/
424 {
425
426 }
427
```

```
428 /*-------------------------------------------------------------------------------
429         DAC12 interrupt service routine
430 -------------------------------------------------------------------------------*/
431 interrupt (DACDMA_VECTOR) DAC12(void)
432 /*-------------------------------------------------------------------------------*/
433 {
434
435 }
436
437 /*-------------------------------------------------------------------------------
438         ADC12 interrupt service routine
439 -------------------------------------------------------------------------------*/
440 interrupt (ADC12_VECTOR) ADC12(void)
441 /*-------------------------------------------------------------------------------*/
442 {
443
444 }
445
446 /*-------------------------------------------------------------------------------*/
447 /*      DAC12 interrupt service routine
448 -------------------------------------------------------------------------------*/
449 interrupt (COMPARATORA_VECTOR) COMPERATOR(void)
450 /*-------------------------------------------------------------------------------*/
451 {
452
453 }
```

Listing B.5: Header der Interrupthandler (ISR.h)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430 ISR header
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------*/
8
9  #ifndef ISR_H_
10 #define ISR_H_
11
12 /*-------------------------------------------------------------------------------
13         Structures
14 -------------------------------------------------------------------------------*/
15 typedef struct {unsigned char frame_bit_counter;
16                 unsigned char frame_byte_counter;
17                 unsigned char frame_last_bit;
18                 unsigned char frame_byte[11];
19 } RADIO_FRAME_STRUCT;
20
21 /*-------------------------------------------------------------------------------
22         Defines
23 -------------------------------------------------------------------------------*/
24 /* defines for the capture interrupt    */
25
26 /*      Receive sequences in interrupt  */
27 enum SE_SEQ_POS{
28         SE_SEQ_PREPARE,         /* 0 -> nothing to do, first interrupt          */
29         SE_SEQ_START,           /* 1 -> first measurement completed, start sequence
               detected?           */
30         SE_SEQ_PRE_WAIT,        /* 2 -> complete start sequence detected
               */
31         SE_SEQ_DATA_START,      /* 3 -> check for firt 1
                              */
32         SE_SEQ_RXDATA           /* 4 -> first zero detected, receive data       */
33 };
34
```

```
35 /* min length for the start sequence    */
36 #define START_SEQ_LENGTH_MIN    700
37 /* max length for the start sequence    */
38 #define START_SEQ_LENGTH_MAX    1200
39
40 /* define length for a 1 bit in manchester code */
41 #define HIGH_BIT_LENGTH                 140
42 /* define length for a 0 bit in manchester code */
43 #define LOW_BIT_LENGTH                  60
44
45 /* min length for two bits mancester code or 1 bit real code    */
46 //#define ONE_BIT_LENGTH                        130
47 /* max length for two bits mancester code or 1 bit real code    */
48 //#define TWO_BIT_LENGTH                        230
49 /* length for three bits mancester code or 1.5 bit real code    */
50 //#define THREE_BIT_LENGTH              350
51 /* length for four bits mancester code or 2 bit real code               */
52 //#define FOUR_BIT_LENGTH                       450
53
54
55 #endif /*ISR_H_*/
```

Listing B.6: Quellcode der LCD Ansteuerung (LCD16x2.c)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        LCD16x2 functions
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------*/
8
9  /*-------------------------------------------------------------------------------
10         Headerfiles
11 -------------------------------------------------------------------------------*/
12 #include <main.h>
13 #include <lcd16x2.h>
14 #include <msp430x16x.h>
15 //#include <MSP_functions.h>
16
17 /*-------------------------------------------------------------------------------
18         Static
19 -------------------------------------------------------------------------------*/
20 static unsigned char *RTC_days[7]={
21                 "MON ",
22                 "TUE ",
23                 "WED ",
24                 "THU ",
25                 "FRY ",
26                 "SAT ",
27                 "SUN "
28 };
29
30 static unsigned char *RTC_months[12]={
31                 "Jan",
32                 "Feb",
33                 "Mar",
34                 "Apr",
35                 "May",
36                 "Jun",
37                 "Jul",
38                 "Aug",
39                 "Sep",
40                 "Oct",
41                 "Nov",
42                 "Dec"
```

```
43 };
44
45 /*---------------------------------------------------------------------------
46         x ms delay loop
47 ---------------------------------------------------------------------------*/
48 void LCD_delay_ms(unsigned int LCD_delay_ms)
49 {
50         unsigned int LCD_delay_loop;
51
52         switch (uC_FREQUENCY)
53         {
54             case MHZ_1: LCD_delay_loop = 73;    //(88*b) cycles (for 1MHz)
55                                 break;
56                 case MHZ_2: LCD_delay_loop = 150;       //(175*b) cycles (for 2MHz)
57                                         break;
58                 case MHZ_4: LCD_delay_loop = 304;       //(350*b) cycles (for 4MHz)
59                                         break;
60                 case MHZ_8: LCD_delay_loop = 610;       //(610*b) cycles (for 8MHz)
61                                         break;
62                 default: break;
63         }
64
65         for (; LCD_delay_ms>0; LCD_delay_ms--)
66         {
67                 LCD_delay(LCD_delay_loop);
68         }
69 }
70
71
72 /*---------------------------------------------------------------------------
73         delay loop
74         counts x times
75 ---------------------------------------------------------------------------*/
76 void LCD_delay(unsigned int LCD_delay_loop)
77 {
78         for(; LCD_delay_loop>0; LCD_delay_loop--) _NOP();
79 }
80
81
82 /*---------------------------------------------------------------------------
83         Init 16x2LCD
84 ---------------------------------------------------------------------------*/
85 void LCD_init(void)
86 {
87         LCD_DATA_PORT &= ~LCD_RS_PIN;           // sets LCD in COMMAND MODE
88     LCD_delay_ms(15);                           // Delay ca.15ms
89
90     LCD_DATA_PORT |= BIT4 | BIT5;       // D7-D4 = 0011
91     LCD_DATA_PORT &= ~BIT6 & ~BIT7;
92     LCD_enable();
93     LCD_delay_ms(5);
94     LCD_enable();
95     LCD_delay_ms(1);
96     LCD_enable();
97     LCD_delay_ms(1);
98     LCD_DATA_PORT &= ~BIT4;             // D7-D4 = 0010
99     LCD_enable();
100
101    LCD_send_cmd(LCD_ON);                               //      switch LCD on
102    LCD_send_cmd(LCD_CLR);                      //      clear LCD
103    LCD_send_cmd(LCD_LINE1);                    //      set cursor to x=0, y=0
104 }
105
106
107 /*---------------------------------------------------------------------------
108         Toggle LCD_E pin
109 ---------------------------------------------------------------------------*/
```

```
110 void LCD_enable(void)
111 {
112         LCD_DATA_PORT |= LCD_E_PIN;
113         _NOP();
114         _NOP();
115         LCD_DATA_PORT &= ~LCD_E_PIN;
116 }
117
118
119 /*---------------------------------------------------------------------------
120         Send command to LCD
121 ---------------------------------------------------------------------------*/
122 void LCD_send_cmd (unsigned char LCD_out_byte)
123 {
124         unsigned char LCD_tmp;
125
126         LCD_delay_ms(1);
127         LCD_tmp = LCD_out_byte & 0xf0;          // get upper nibble
128         LCD_DATA_PORT &= 0x0f;
129         LCD_DATA_PORT |= LCD_tmp;           // send CMD to LCD
130         LCD_DATA_PORT &= ~LCD_RS_PIN;           // sets LCD in COMMAND MODE
131         LCD_enable();
132         LCD_tmp = LCD_out_byte & 0x0f;
133         LCD_tmp = LCD_tmp << 4;                 // get down nibble
134         LCD_DATA_PORT &= 0x0f;
135         LCD_DATA_PORT |= LCD_tmp;
136         LCD_DATA_PORT &= ~LCD_RS_PIN;           // sets LCD in COMMAND MODE
137         LCD_enable();
138 }
139
140
141 /*---------------------------------------------------------------------------
142         Send DATA to LCD
143 ---------------------------------------------------------------------------*/
144 void LCD_send_data (unsigned char LCD_out_byte)
145 {
146         unsigned char LCD_tmp;
147
148         LCD_delay_ms(1);
149         LCD_tmp = LCD_out_byte & 0xf0;          // get upper nibble
150         LCD_DATA_PORT &= 0x0f;
151         LCD_DATA_PORT |= LCD_tmp;           // send DATA to LCD
152         LCD_DATA_PORT |= LCD_RS_PIN;            // sets LCD in DATA MODE
153         LCD_enable();
154         LCD_tmp = LCD_out_byte & 0x0f;
155         LCD_tmp = LCD_tmp << 4;             // get lower nibble
156         LCD_DATA_PORT &= 0x0f;
157         LCD_DATA_PORT |= LCD_tmp;
158         LCD_DATA_PORT |= LCD_RS_PIN;            // sets LCD in DATA MODE
159         LCD_enable();
160 }
161
162
163 /*---------------------------------------------------------------------------
164         Send string to LCD
165 ---------------------------------------------------------------------------*/
166 void LCD_send_text(char *out_string)
167 {
168     while (*out_string)                             // *out_string != '\0'
169     {
170         LCD_send_data(*out_string);             // send byte to LCD
171         out_string++;
172     }
173 }
174
175
176 /*---------------------------------------------------------------------------
```

```
177         send date
178 ------------------------------------------------------------------------------*/
179 void LCD_send_time(void)
180 /*------------------------------------------------------------------------------*/
181 {
182         unsigned char time_char;
183 
184         LCD_send_cmd(LCD_LINE2);
185         LCD_send_text("                ");
186 
187         LCD_send_cmd(LCD_RTC_WDAY);
188         LCD_send_text(RTC_days[RTC_values.wday]);
189 
190         time_char = (RTC_values.hr / 10) + 48;
191         LCD_send_data(time_char);
192         time_char = (RTC_values.hr % 10) + 48;
193         LCD_send_data(time_char);
194         LCD_send_data(':');
195 
196         time_char = (RTC_values.min / 10) + 48;
197         LCD_send_data(time_char);
198         time_char = (RTC_values.min % 10) + 48;
199         LCD_send_data(time_char);
200         LCD_send_data(':');
201 
202         time_char = (RTC_values.sec / 10) + 48;
203         LCD_send_data(time_char);
204         time_char = (RTC_values.sec % 10) + 48;
205         LCD_send_data(time_char);
206 }
207 
208 
209 /*------------------------------------------------------------------------------
210         send date
211 ------------------------------------------------------------------------------*/
212 void LCD_send_date(void)
213 /*------------------------------------------------------------------------------*/
214 {
215         unsigned char time_char;
216 
217         LCD_send_cmd(LCD_LINE2);
218         LCD_send_text("                ");
219 
220         LCD_send_cmd(LCD_RTC_DAY);
221         time_char = (RTC_values.day / 10) + 48;
222         LCD_send_data(time_char);
223         time_char = (RTC_values.day % 10) + 48;
224         LCD_send_data(time_char);
225 
226         LCD_send_data('-');
227         LCD_send_text(RTC_months[RTC_values.month]);
228 
229         LCD_send_data('-');
230         time_char = RTC_values.year / 1000;
231         LCD_send_data(time_char+48);
232         time_char = (RTC_values.year / 100) - ((RTC_values.year/1000) * 10);
233         LCD_send_data(time_char+48);
234         time_char = (RTC_values.year / 10) - ((RTC_values.year/100) * 10);
235         LCD_send_data(time_char+48);
236         time_char = RTC_values.year % 10;
237         LCD_send_data(time_char+48);
238 }
239 
240 
241 /*------------------------------------------------------------------------------
242         send sensor address
243 ------------------------------------------------------------------------------*/
```

```
244 void LCD_send_sensor(unsigned char address_value)
245 /*---------------------------------------------------------------------------*/
246 {
247         unsigned char address_hundred, address_tenner, address_ones;
248 
249         address_hundred = address_value / 100;
250         address_value %= 100;
251         address_tenner = address_value / 10;
252         address_value %= 10;
253         address_ones = address_value / 1;
254 
255         LCD_send_cmd(LCD_LINE1);
256 
257         LCD_send_text("Sensor ");
258         LCD_send_data(address_hundred+48);
259         LCD_send_data(address_tenner+48);
260         LCD_send_data(address_ones+48);
261         LCD_send_text(" aktiv");
262 
263 }
```

Listing B.7: Header der LCD Ansteuerung (LCD16x2.h)

```
1  /*---------------------------------------------------------------------------
2          Project:                        LCD16x2 module header
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last update:                    10/28/2007
6          Author:                         Stephan Plaschke
7  ---------------------------------------------------------------------------*/
8 
9  #ifndef LCD16X2_H_
10 #define LCD16X2_H_
11 
12 /*---------------------------------------------------------------------------
13         Headerfiles
14 ---------------------------------------------------------------------------*/
15 #include <rtc.h>
16 
17 /*---------------------------------------------------------------------------
18         Global variables
19 ---------------------------------------------------------------------------*/
20 
21 extern unsigned char BATMON_control_reg;
22 
23 extern RTC_MSP430               RTC_values;                     // global RTC values
24 
25 /*---------------------------------------------------------------------------
26         Defined values, change for each project
27 ---------------------------------------------------------------------------*/
28 
29 #define   LCD_E_PIN                     BIT1
30 #define   LCD_RS_PIN            BIT3
31 #define   LCD_LIGHT_PIN         BIT0
32 #define   LCD_DATA_PORT         P4OUT
33 
34 /*---------------------------------------------------------------------------
35         Defined values, equal in each project, don't change
36 ---------------------------------------------------------------------------*/
37 
38 #define   LCD_LIGHT_ON          (LCD_DATA |= LCD_LIGHT_PIN)     // sets pin4.0 HIGH
39 #define   LCD_LIGHT_OFF         (LCD_DATA &= ~LCD_LIGHT_PIN)    // clears pin4.0
40 
41 #define   LCD_ON                        0x0c    //      LCD control command, ON
42 #define   LCD_OFF                       0x08    //  LCD control command, OFF
43 #define   LCD_CLR                       0x01    //  LCD control command, CLEARSCREEN
```

```
44 #define   LCD_LINE1                     0x80    //      LCD control command, sets cursor
       on y=0,x=0
45 #define   LCD_LINE2                     0xc0    //      LCD control command, sets cursor
       on y=1,x=0
46 #define   LCD_ENDLINE1          0x8f    //      LCD control command, sets cursor on y=0,x
       =15
47 #define   LCD_ENDLINE2          0xcf    //      LCD control command, sets cursor on y=1,x
       =15
48
49 /*-------------------------------------------------------------------------------
50         Prototypes
51 -------------------------------------------------------------------------------*/
52
53 void LCD_delay_ms(unsigned int);
54 void LCD_delay(unsigned int);
55 void LCD_send_cmd (unsigned char);
56 void LCD_send_data (unsigned char);
57 void LCD_send_text(char *);
58 void LCD_enable(void);
59 void LCD_init(void);
60 void LCD_send_time(void);
61 void LCD_send_date(void);
62 void LCD_send_sensor(unsigned char);
63
64
65 #endif /*16X2LCD_H_*/
```

Listing B.8: Quellcode der Hauptschleife (main.c)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430 project source file
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last Update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------
8
9  -------------------------------------------------------------------------------
10         Headerfiles
11 -------------------------------------------------------------------------------*/
12 #include <ISR.h>
13 #include <RTC.h>
14 #include <main.h>
15 #include <flash.h>
16 #include <stdio.h>
17 #include <signal.h>
18 #include <lcd16x2.h>
19 #include <uart_menu.h>
20 #include <msp430x16x.h>
21 #include <MSP_functions.h>
22
23
24 /*-------------------------------------------------------------------------------
25         Global variables
26 -------------------------------------------------------------------------------*/
27 unsigned char BATMON_control_reg;       // global controll register
28 unsigned char BUTTON_state;                     // actual button state
29 unsigned char RADIO_rx_state;           // data receive state
30 unsigned char UART_menu_position;       // UART_menue position
31
32 unsigned char WDT_function;                     // WDT function, BLINKY for RTC SETUP or
       RTC for RTC mode
33 unsigned char RTC_menue_position;       // menue position in RTC setup
34
35 unsigned char Enable_clear_lcd;         // clear first LCD row
36
```

```
37 unsigned char SENSOR_addr_req[5];              // array for intervall mode, shows
       required address
38 unsigned char SENSOR_address[40];              // array with sensor address
39 unsigned char SENSOR_active;                   // active sensor count
40 unsigned char SENSOR_last_data[8][40];  // last received data from ecah sensor
41
42 RADIO_FRAME_STRUCT      RADIO_frame;    // received data frame
43 RTC_MSP430                      RTC_values;             // global RTC values
44
45 unsigned char global_string[MAX_STRING_LENGTH];         // global command string
46
47 //alles für den Flash
48 unsigned char  WRITE_BUF [BUF_SIZE];
49 unsigned char  READ_BUF [BUF_SIZE];
50 unsigned char  Ring_Full;
51 unsigned short Index_Ring;
52 unsigned short Count_Ring;
53 unsigned short Index_Last;
54
55
56 /*-------------------------------------------------------------------------------
57         MAIN LOOP
58 -------------------------------------------------------------------------------*/
59 int main (void)
60 {
61         unsigned char OSC_error;
62         unsigned char UART_error;
63
64         RTC_MSP430 RTC_tmp={0,0,0,0};
65
66 /*-------------------------------------------------------------------------------
67         Set local/global variables
68 -------------------------------------------------------------------------------*/
69         // init clock
70         RTC_values.day          = 29;
71         RTC_values.month        = 01;
72         RTC_values.year         = 2008;
73         RTC_values.wday         = MON;
74         RTC_values.hr           = 23;
75         RTC_values.min          = 59;
76         RTC_values.sec          = 55;
77
78         Index_Ring = 256;                       // default address in flash
79         Index_Last = 256;
80
81         UART_menu_position = UART_MENU_FIRSTCHAR;
82         Enable_clear_lcd = OFF;
83
84 //-------------------------------------------------------------------------------
85
86         //OSC_error = DCO_set(uC_FREQUENCY);    // internal oscillator
87         OSC_error = XT2_set(uC_FREQUENCY);              // external oscillator
88         // initialise USART, enable RX&TX interrupt and RX&TX modul
89         USART0_init(RX_INT, RX_TX);
90         // initialise PORT direction/function
91         PORTS_init();
92         // initialise 16x2 character LCD
93         LCD_init();
94         // initilise TIMERB, debounce function for button B1-B3
95         TIMERB_init();
96         // initialise RTC
97         RTC_startup();
98         // initialise Flash
99         init_Flash();
100        // initialise radio unit
101        RADIO_init();
102        // global interrupt enable
```

```
103         _EINT();
104         // read controll data from flash?
105         FLASH_read_at_startup();
106         // display time on LCD
107         LCD_send_time();
108         // stop measurement
109         UART_menu_stop();
110         // switch LEDS off
111         LED1_OFF;
112         LED2_OFF;
113
114         while(1)
115         {
116                 RTC_compare(&RTC_tmp);
                            // look if real time clock values changed
117
118                 if (BATMON_control_reg & GLOBAL_STRING_RECEIVED)        // string from
                    UART received
119                 {
120                         UART_error = UART0_cmp_string();
                            // compare string with valid comands
121
122                         if (UART_error)
123                                 UART0_send_text("NOK\r\n");
124                         else
125                                 UART0_send_text("OK\r\n");
126
127                         BATMON_control_reg &= !GLOBAL_STRING_RECEIVED;  // clear global
                            value
128                 }
129
130                 if (BATMON_control_reg & VALID_DATA_RECEIVED)           // data received
                    from sensor
131                 {
132                         RADIO_store_frame();
                                    // store valid frame in flash
133                         LCD_send_sensor(RADIO_frame.frame_byte[3]);             // show
                            sensor address on LCD
134                         RADIO_enable();
                                            // enable radio modul
135                         BATMON_control_reg &= !VALID_DATA_RECEIVED;             // clear
                            global value
136                 }
137
138                 if (BATMON_control_reg & CLEAR_FIRST_LCD_ROW)           // clear first
                    LCD row
139                 {
140                         LCD_send_cmd(LCD_LINE1);
141                         LCD_send_text("                ");
142                         BATMON_control_reg &= !CLEAR_FIRST_LCD_ROW;
143                 }
144
145                 if((BATMON_control_reg & B1) || (BATMON_control_reg & B2)
146                                 || (BATMON_control_reg & B3))
                                    // Button pressed jump into menu
147                 {
148                         BATMON_control_reg &= ~(B1|B2|B3);
                            // clear button state
149                         BATMON_menu_options();
150                 }
151         }
152
153         return (0);
154 }
```

Listing B.9: Header der Hauptschleife (main.h)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430 project header
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------*/
8
9  #ifndef MAIN_H_
10 #define MAIN_H_
11
12 /*-------------------------------------------------------------------------------
13         Defined values, equal in each project, don't change
14 -------------------------------------------------------------------------------*/
15
16 /* Defines for selected frequencies, claculation:       */
17 /* low power crystal frequency divided by 8 = 4096Khz   */
18 /* timer uses DCO frequency and counts up in one ACLK wave      */
19 /* e.g. 1/(2MHz)*488 = 0,000244s = 4098kHz      */
20 #define MHZ_1                                   244
21 #define MHZ_2                                   488
22 #define MHZ_4                                   976
23 #define MHZ_8                                   1952
24
25 #define MAX_STRING_LENGTH               15              // max stringlength for UART
26
27 #define GLOBAL_STRING_RECEIVED  0x02    // value in BATMON_control_reg
28 #define VALID_DATA_RECEIVED             0x04    // value in BATMON_control_reg
29 #define CLEAR_FIRST_LCD_ROW             0x08    // value in BATMON_control_reg
30
31 #define PRINTF_LCD                              (BATMON_control_reg |= 0x01)
32 #define PRINTF_UART                             (BATMON_control_reg &= ~0x01)
33
34 enum boolean    {FALSE, TRUE};
35 enum status     {OFF, ON};
36
37 enum uart_interrupts    {RX_INT, TX_INT, RX_TX_INT};
38 enum uart_modules               {RX, TX, RX_TX};
39 enum uart_speed                 {BAUD_4800, BAUD_9600, BAUD_19k2, BAUD_115k2};
40
41 enum escapes{   SPACE = ' ', BACKSPACE = '\b', HTAB = '\t',
42                         RETURN = '\r', NEWLINE = '\n', VTAB = '\v',
43                         NULLTERMINATOR = '\0'};
44
45
46 /*-------------------------------------------------------------------------------
47         Defined values
48 -------------------------------------------------------------------------------*/
49
50 #define MSP430x169              1                       /* select the used MSP430
                                        */
51 #define uC_FREQUENCY    MHZ_8           /* select the MSP430 frequency  (1,2,4,8 MHZ)
       */
52
53 #define ADC0_BIT                0x10            /* define ADC0_portpin for PxSEL register
                  */
54 #define ADC0_BS                 INCH_4          /* define ADC0_portpin for ADC1xMCTL0
       register  */
55
56 #define USART_SEL               0x30            /* define USART pins
       */
57 #define USART_DIR               0x10            /* define USART pin direction   */
58 #define USART_BAUD              BAUD_115k2      /* define USART BAUDRATE (BAUD_4800,
       BAUD_9600, BAUD_19k2, BAUD_115k2(only@4&8MHz) )           */
59
```

```
60 /* PORT definitions, 1->output, 0->input        */
61 /* PORTX_IES 0-> rising edge, 1->falling edge   */
62 /* PORTX_IE  0-> interrupt disabled, 1->interrupt enabled */
63 #define PORT1_DIR               0x00
64 #define PORT1_SEL               0x02
65 #define PORT1_IES               0xE0
66 #define PORT1_IE                0x00
67
68 #define PORT2_DIR               0x1F
69 #define PORT2_SEL               0x00
70 #define PORT2_IES               0x00
71 #define PORT2_IE                0x00
72
73 #define PORT3_DIR               BIT7 | BIT6 | USART_DIR
74 #define PORT3_SEL               USART_SEL
75
76 #define PORT4_DIR               0xFF
77 #define PORT4_SEL               0x00
78
79 #define PORT5_DIR               0xFF
80 #define PORT5_SEL               0x00
81
82 #define PORT6_DIR               ADC0_BIT
83 #define PORT6_SEL       ~ADC0_BIT
84
85 #define LED1_ON                 (P3OUT &= ~BIT6)
86 #define LED1_OFF                (P3OUT |= BIT6)
87 #define LED1_TOGGLE             (P3OUT ^= BIT6)
88
89 #define LED2_ON                 (P3OUT &= ~BIT7)
90 #define LED2_OFF                (P3OUT |= BIT7)
91 #define LED2_TOGGLE             (P3OUT ^= BIT7)
92
93 #define B1                              0x20
94 #define B2                              0x40
95 #define B3                              0x80
96 #define BUTTON_INPUT    (~(P1IN | 0x1F))
97
98
99
100 #endif /*MAIN_H_*/
```

Listing B.10: Quellcode der Mikrocontroller spezifischen Funktionen (MSP_functions.c)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        Specialfunctions for the MSP430x1232 and
               MSP430x169
3          Used components:                MSP430-169STK
4          Date:                           10/28/2007
5          Last update:                    09/05/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------
8
9  -------------------------------------------------------------------------------
10         Headerfiles
11 -------------------------------------------------------------------------------*/
12 #include <ISR.h>
13 #include <RTC.h>
14 #include <main.h>
15 #include <flash.h>
16 #include <stdio.h>
17 #include <flash.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include <signal.h>
21 #include <lcd16x2.h>
```

```
22 #include <msp430x16x.h>
23 #include <uart_menu.h>
24 #include <MSP_functions.h>
25
26 /*-------------------------------------------------------------------------------
27         Static
28 -------------------------------------------------------------------------------*/
29 static unsigned char *BATMON_menu_msg[8]={
30                 " Show sensors   ",
31                 " Start recording",
32                 " Stop recording ",
33                 " Load contr.data",
34                 " Save contr.data",
35                 " Set date/time  ",
36                 "NEXT   OK   EXIT",
37                 "YES    NO       "
38 };
39
40
41 /*-------------------------------------------------------------------------------
42         Functions
43 -------------------------------------------------------------------------------*/
44
45 /*-------------------------------------------------------------------------------
46         Set DCO to selected frequency
47         clk_multi is the multiplier of the  A_clk (32768KHz)
48         Error_code:
49         0       ->      no error
50         1       ->      oscillator error
51 -------------------------------------------------------------------------------*/
52 unsigned char DCO_set (unsigned int clk_multi)
53 /*-------------------------------------------------------------------------------*/
54 {
55         unsigned int Compare, Oldcapture = 0;
56         unsigned int Max_turns=10000;
57         unsigned char Error_code=0;
58
59         BCSCTL1 |= DIVA_3;                              // ACLK= LFXT1CLK/8
60         CCTL2 = CM_1 + CCIS_1 + CAP;                    // CAP, ACLK
61         TACTL = TASSEL_2 + MC_2 + TACLR;                // SMCLK, cont-mode, clear
62
63         while (1)
64         {
65         while (!(CCIFG & CCTL2));                       // Wait until capture occured
66         CCTL2 &= ~CCIFG;                          // Clear flag
67         Compare = CCR2;                           // Get current captured SMCLK
68         Compare = Compare - Oldcapture;           // SMCLK difference
69         Oldcapture = CCR2;                     // get current captured SMCLK
70
71         if (clk_multi == Compare)                       // clk_multi = compare then exit
72         {
73                 Error_code = 0;                                 // no error occurs
74                 break;
75         }
76         else if (clk_multi < Compare)           // too fast, slow it down
77         {
78                 DCOCTL--;       /* decrease DCOCTL register
79                  *      if DCOCTL overrun and basic clock register RSELx bits are not
                         equal 0
80                  * decrease RSELx bits
                                                                */
81                 if ((DCOCTL == 0xFF) && (!(BCSCTL1 == (XT2OFF + DIVA_3))))
82                         BCSCTL1--;
83         }
84         else                                                            // too slow, fast
               it up
85         {
```

```
86                  DCOCTL++;       /* increase DCOCTL register
87                   * if DCOCTL overrun and basic clock register RSELx bits are equal 1
88                   * increase RSELx bits
                                                                    */
89                  if ((DCOCTL == 0x00) && (!(BCSCTL1 == (XT2OFF + 0x07 + DIVA_3))))
90                          BCSCTL1++;
91          }
92
93          if (!(--Max_turns))     // if 10k turns reached exit with error
94          {
95                  Error_code = 1;
96                  break;
97          }
98          }
99          CCTL2 = 0;                                              // Stop CCR2
100         TACTL = 0;                                          // Stop Timer_A
101         BCSCTL1 &= ~DIVA_3;                                 // restore BCSCTL1
102         BCSCTL2 &= ~(DIVS_3 | DIVM_3);  // restore BCSCTL2
103
104         return (Error_code);
105 }
106
107
108 /*-------------------------------------------------------------------------------
109         Set XT2 to selected frequency
110         clk_multi is the multiplier of the  A_clk (32768KHz)
111         Error_code:
112         0       ->      no error
113         1       ->      external oscillator error
114         2       ->      clock wrong, set to 4MHZ
115 -------------------------------------------------------------------------------*/
116 unsigned char XT2_set (unsigned int clk_multi)
117 /*-------------------------------------------------------------------------------*/
118 {
119         unsigned int Max_turns = 1000;
120         unsigned char i, Error_code = 0;
121
122         _BIC_SR(OSCOFF);                                    // switch LFXT on
123         BCSCTL1 &= ~XT2OFF;             // XT2on
124
125         do
126         {
127                 IFG1 &= ~OFIFG;              // Clear OSCFault flag
128             for (i = 0xFF; i > 0; i--); // Time for flag to set
129
130             if (!(--Max_turns))         // stop after 1k tries, error with ext osc!
131             {
132                 Error_code = 1;
133                 break;
134             }
135         }
136         while ((IFG1 & OFIFG));         // OSCFault flag still set?
137
138         BCSCTL2 |= SELM_2 | SELS;               // MCLK = SMCLK = XT2 (safe)
139
140         switch (clk_multi)
141         {
142         case MHZ_8:
143                 BCSCTL2 &= ~(DIVS_3 | DIVM_3);  // SMCLK&MCLK = XT2 / 1
144                 break;
145         case MHZ_4:
146                 BCSCTL2 |= DIVS_1 | DIVM_1;             // SMCLK&MCLK = XT2 / 2
147                 break;
148         case MHZ_2:
149                 BCSCTL2 |= DIVS_2 | DIVM_2;             // SMCLK&MCLK = XT2 / 4
150                 break;
151         case MHZ_1:
```

```
152                 BCSCTL2 |= DIVS_3 | DIVM_3;             // SMCLK&MCLK = XT2 / 8
153                 break;
154         default:
155                 Error_code = 2;
156                 BCSCTL2 |= DIVS_1 | DIVM_1;             // SMCLK&MCLK = XT2 / 2
157                 break;
158         }
159
160         return (Error_code);
161 }
162
163
164 /*-------------------------------------------------------------------------------
165         putchar() defined for use with printf
166         PRINTF_LCD :    output on LCD of the MSP430x169STK board
167         PRINTF_UART:    output through RS232
168 -------------------------------------------------------------------------------*/
169 int putchar(int c)
170 //-------------------------------------------------------------------------------
171 {
172         UART0_send((unsigned char)c);
173         return(c);
174 }
175
176 //*******************************************************************************/
177 /*-------------------------------------------------------------------------------
178         MSP430x169 specific functions
179 -------------------------------------------------------------------------------*/
180
181 /*-------------------------------------------------------------------------------
182         PORTS init
183 -------------------------------------------------------------------------------*/
184 void PORTS_init(void)
185 /*-------------------------------------------------------------------------------*/
186 {
187
188         P1SEL = PORT1_SEL;              // Port1 I/O Funktion
189         P1DIR = PORT1_DIR;              // Port1 Ausgang
190         P1IES = PORT1_IES;              // Port1 interrupt edge select
191         P1IE  = PORT1_IE;               // Port1 interrupt enable
192
193         P2SEL = PORT2_SEL;              // Port2 I/O Funktion
194         P2DIR = PORT2_DIR;              // Port2 Ausgang
195         P2IES = PORT2_IES;              // Port2 interrupt edge select
196         P2IE  = PORT2_IE;               // Port2 interrupt enable
197
198         P3SEL = PORT3_SEL;              // enable special funktion on PIN3.4, 3.5
199         P3DIR = PORT3_DIR;              // set direction, PIN3.4 OUTPUT
200
201         P4OUT = PORT4_SEL;      // LCD ini, set PORT4 as output
202     P4DIR = PORT4_DIR;
203
204     P5SEL = PORT5_SEL;          // Port5 I/O Funktion
205     P5DIR = PORT5_DIR;          // Port5 Ausgang
206
207         P6SEL = PORT6_SEL;      // P6.x ADC option select
208         P6DIR = PORT6_DIR;
209 }
210
211
212 /*-------------------------------------------------------------------------------
213         ADC0 init
214 -------------------------------------------------------------------------------*/
215 void ADC0_init(void)
216 /*-------------------------------------------------------------------------------*/
217 {
218         ADC12CTL0 = ADC12ON+SHT0_0+REFON+REF2_5V;       // ADC12ON / reference on 2.5V
```

```
219         ADC12CTL1 = SHP;                                                    // SHP benutzen
220         ADC12MCTL0 = SREF_1+ADC0_BS;                                // Vr+=Vref+, ADC0
                bitselect
221 }
222
223
224 /*-------------------------------------------------------------------------------
225         get ADC0 result
226 -------------------------------------------------------------------------------*/
227 int ADC0_sampling(void)
228 /*-------------------------------------------------------------------------------*/
229 {
230         ADC12CTL0 |= ADC12SC + ENC;                     // Sampling open
231         ADC12CTL0 &= ~ADC12SC;                          // Sampling closed, start
                conversion
232         while ((ADC12CTL1 & ADC12BUSY) == 1);   // ADC12 busy?
233         return(*ADC12MEM);                                              // return the
                value read from ADC
234 }
235
236
237 /*-------------------------------------------------------------------------------
238         Init USART0, automatic baud rate generation for selected uC frequency
239 -------------------------------------------------------------------------------*/
240 void USART0_init(unsigned char USART0_IRQ, unsigned char USART0_MODE)
241 /*-------------------------------------------------------------------------------*/
242 {
243         UCTL0   =  SWRST + CHAR;        // set 8bit, none parity and 1 stoppbit
244         UTCTL0  |= SSEL_2;                      // set SMCLK as USART0_CLK
245
246         switch (uC_FREQUENCY)           // set UART to selected mode with selected uC
                frequency
247         {
248         case MHZ_1:
249                 switch (USART_BAUD)
250                 {
251                 case BAUD_19k2: UBR00 = 0x34;                   // see manual
252                         UBR10 = 0x00;
253                         UMCTL0 = 0x20;
254                         break;
255                 case BAUD_9600: UBR00 = 0x68;                   // see manual
256                         UBR10 = 0x00;
257                         UMCTL0 = 0x04;
258                         break;
259                 case BAUD_4800: UBR00 = 0xd0;                   // see manual
260                         UBR10 = 0x00;
261                         UMCTL0 = 0x92;
262                         break;
263                 default: UBR00 = 0x68;                                  // see manual
264                         UBR10 = 0x00;
265                         UMCTL0 = 0x04;
266                 break;
267                 };
268                 break;
269         case MHZ_2:
270                 switch (USART_BAUD)
271                 {
272                 case BAUD_115k2: UBR00 = 0x11;                  // see manual
273                         UBR10 = 0x00;
274                         UMCTL0 = 0x52;
275                         break;
276                 case BAUD_19k2: UBR00 = 0x68;                   // see manual
277                         UBR10 = 0x00;
278                         UMCTL0 = 0x04;
279                         break;
280                 case BAUD_9600: UBR00 = 0xD0;                   // see manual
281                         UBR10 = 0x00;
```

```
282                         UMCTL0 = 0x92;
283                         break;
284                 case BAUD_4800: UBR00 = 0xA0;                   // see manual
285                         UBR10 = 0x01;
286                         UMCTL0 = 0x6D;
287                         break;
288                 default: UBR00 = 0x68;                                  // see manual
289                         UBR10 = 0x00;
290                         UMCTL0 = 0x04;
291                 break;
292                 };
293                 break;
294         case MHZ_4:
295                 switch (USART_BAUD)
296                 {
297                 case BAUD_115k2: UBR00 = 0x22;                  // see manual
298                         UBR10 = 0x00;
299                         UMCTL0 = 0xDD;
300                         break;
301                 case BAUD_19k2: UBR00 = 0xD0;                   // see manual
302                         UBR10 = 0x00;
303                         UMCTL0 = 0x92;
304                         break;
305                 case BAUD_9600: UBR00 = 0xA0;                   // see manual
306                         UBR10 = 0x01;
307                         UMCTL0 = 0x6D;
308                         break;
309                 case BAUD_4800: UBR00 = 0x41;                   // see manual
310                         UBR10 = 0x03;
311                         UMCTL0 = 0x92;
312                         break;
313                 default: UBR00 = 0x68;                                  // see manual
314                         UBR10 = 0x00;
315                         UMCTL0 = 0x04;
316                 break;
317                 };
318                 break;
319         case MHZ_8:
320                 switch (USART_BAUD)
321                 {
322                 case BAUD_115k2: UBR00 = 0x45;                  // see manual
323                         UBR10 = 0x00;
324                         UMCTL0 = 0xAA;
325                         break;
326                 case BAUD_19k2: UBR00 = 0xa0;                   // see manual
327                         UBR10 = 0x01;
328                         UMCTL0 = 0x6D;
329                         break;
330                 case BAUD_9600: UBR00 = 0x41;                   // see manual
331                         UBR10 = 0x03;
332                         UMCTL0 = 0x92;
333                         break;
334                 case BAUD_4800: UBR00 = 0x82;                   // see manual
335                         UBR10 = 0x6;
336                         UMCTL0 = 0x6D;
337                         break;
338                 default: UBR00 = 0x68;                                  // see manual
339                         UBR10 = 0x00;
340                         UMCTL0 = 0x04;
341                 break;
342                 };
343                 break;
344         }
345
346         switch (USART0_MODE)
347                 {
348                 case RX: ME1 |= URXE0;                          // enable UART0 RX
```

```
349                                 break;
350                 case TX: ME1 |= UTXE0;                          // enable UART0 TX
351                                 break;
352                 case RX_TX: ME1 |= (URXE0 | UTXE0); // enable UART0 TX&RX
353                                 break;
354                 default: ME1 &= ~(UTXE0 | URXE0);       // disable UART0 TX&RX
355                                 break;
356                 }
357
358         UCTL0   &= ~SWRST;                                                  // clear SWRST
359
360
361         switch (USART0_IRQ)
362                 {
363                 case RX_INT: IE1 |= URXIE0;                     // enable UART0 RX ISR
364                                 break;
365                 case TX_INT: IE1 |= UTXIE0;                     // enable UART0 TX ISR
366                                 break;
367                 case RX_TX_INT: IE1 |= UTXIE0+URXIE0;   // enable UART0 TX&RX ISR
368                                 break;
369                 default: IE1 &= ~UTXIE0+URXIE0;                 // disable UART0 TX&RX
                        ISR
370                                 break;
371                 }
372 }
373
374 /*-------------------------------------------------------------------------------
375         send character
376 -------------------------------------------------------------------------------*/
377 void UART0_send(unsigned char out_char)
378 /*-------------------------------------------------------------------------------*/
379 {
380         while(!(UTCTL0&TXEPT));         // look if USART0 still busy
381         TXBUF0 = out_char;                      // write char in transmit buffer
382 }
383
384
385 /*-------------------------------------------------------------------------------
386         Send string to USART
387 -------------------------------------------------------------------------------*/
388 void UART0_send_text(char *out_string)
389 /*-------------------------------------------------------------------------------*/
390 {
391     while (*out_string)                                 // *out_string != '\0'
392     {
393         UART0_send(*out_string);                // send byte to LCD
394         out_string++;
395     }
396 }
397
398
399 /*-------------------------------------------------------------------------------
400         TIMERB init
401 -------------------------------------------------------------------------------*/
402 void TIMERB_init(void)
403 /*-------------------------------------------------------------------------------*/
404 {
405         // initialise TIMERB0, ACLK/1, up mode, capture/compare0 int
406         TBCTL   &= ~MC_3;                                                   // disable
                Timer_B
407         TBCTL   = TBSSEL_1 | TBCLR;                             //  set timer register,
                clear TAR
408         TBCCTL0 = CCIE;                                                     //  set control
                register
409         TBCCR0  = 500;
410         TBCTL   |= MC_1;                                                    // start Timer_B
411
```

```
412         _EINT();                // enable interrupts
413 }
414
415
416 /*---------------------------------------------------------------------------
417         store receive frame
418 ---------------------------------------------------------------------------*/
419 void RADIO_store_frame(void)
420 /*---------------------------------------------------------------------------*/
421 {
422         static DATA_STRUCT Data;
423         DATA_STRUCT *pData = &Data;
424
425         pData->C0  = RTC_values.day;
426         pData->C1  = RTC_values.month;
427         pData->C2  = (unsigned char)(RTC_values.year >> 8);
428         pData->C3  = (unsigned char)RTC_values.year;
429         pData->C4  = RTC_values.wday;
430         pData->C5  = RTC_values.hr;
431         pData->C6  = RTC_values.min;
432         pData->C7  = RTC_values.sec;
433         pData->C8  = RADIO_frame.frame_byte[3];
434         pData->C9  = RADIO_frame.frame_byte[4];
435         pData->C10 = RADIO_frame.frame_byte[5];
436         pData->C11 = RADIO_frame.frame_byte[6];
437         pData->C12 = RADIO_frame.frame_byte[7];
438         pData->C13 = RADIO_frame.frame_byte[8];
439         pData->C14 = RADIO_frame.frame_byte[9];
440         pData->C15 = RADIO_frame.frame_byte[10];
441
442         Write_Ring_Flash(pData);
443 }
444
445
446 /*---------------------------------------------------------------------------
447         init receive frame
448 ---------------------------------------------------------------------------*/
449 void RADIO_init_frame(void)
450 /*---------------------------------------------------------------------------*/
451 {
452         static DATA_STRUCT Data;
453         DATA_STRUCT *pData;
454
455         pData = &Data;
456
457         // store time and data
458         pData->C0  = RTC_values.day;
459         pData->C1  = RTC_values.month;
460         pData->C2  = (unsigned char)(RTC_values.year >> 8);
461         pData->C3  = (unsigned char)RTC_values.year;
462         pData->C4  = RTC_values.wday;
463         pData->C5  = RTC_values.hr;
464         pData->C6  = RTC_values.min;
465         pData->C7  = RTC_values.sec;
466         pData->C8  = 0x00;
467         pData->C9  = 0x00;
468         pData->C10 = 0x00;
469         pData->C11 = 0x00;
470         pData->C12 = 0x00;
471         pData->C13 = 0x00;
472         pData->C14 = 0x00;
473         pData->C15 = 0x00;
474
475         // write data to flash
476         Write_Ring_Flash(pData);
477 }
478
```

```
479
480 /*---------------------------------------------------------------------------
481         init radio unit
482 ---------------------------------------------------------------------------*/
483 void RADIO_init(void)
484 /*---------------------------------------------------------------------------*/
485 {
486         // initialise TIMERA0, SMCLK/8, continous mode, capture on both edges, capture/
                compare0 int
487         TACTL   = TASSEL_2 | ID_3 | TACLR;      //  set timer register, clear TAR
488         TACCTL0 = CM_1 | CAP | SCS | CCIE;  //  set control register
489         TACTL  |= MC_2;                                         // start Timer_A
490 }
491
492
493 /*---------------------------------------------------------------------------
494         enable radio unit
495 ---------------------------------------------------------------------------*/
496 void RADIO_enable(void)
497 /*---------------------------------------------------------------------------*/
498 {
499         //      begin with new frame
500         global_string[0] = NULLTERMINATOR;
501         BATMON_control_reg &= ~VALID_DATA_RECEIVED;     // clear global value
502         RADIO_rx_state = SE_SEQ_PREPARE;
503
504         TACCTL0 |= CCIE | CAP;  // enable capture interrupt
505         TACTL   |= MC_2;                // stop Timer_A
506 }
507
508
509 /*---------------------------------------------------------------------------
510         disable radio unit
511 ---------------------------------------------------------------------------*/
512 void RADIO_disable(void)
513 /*---------------------------------------------------------------------------*/
514 {
515         TACTL   &= ~MC_2;               // stop Timer_A
516         TACCTL0 &= ~CAP;
517         TACCTL0 &= ~CCIE;               // disable interrupt
518 }
519
520
521 /*---------------------------------------------------------------------------
522         batmon main menu
523 ---------------------------------------------------------------------------*/
524 void BATMON_menu_options(void)
525 /*---------------------------------------------------------------------------*/
526 {
527         unsigned char BATMON_menu_position;
528         unsigned char BATMON_menu_counter;
529         unsigned char time_tmp, i;
530
531         BATMON_menu_position = BATMON_MENU_MAIN;
532         BATMON_menu_counter = 0;
533
534         while(BATMON_menu_position != BATMON_MENU_EXIT)
535         {
536                 switch(BATMON_menu_position)
537                 {
538                 case BATMON_MENU_START:
539                         LCD_send_cmd(LCD_LINE1);
540                         LCD_send_text(BATMON_menu_msg[1]);
541                         LCD_send_cmd(LCD_LINE2);
542                         LCD_send_text(BATMON_menu_msg[7]);
543
544                         while(1)
```

```
545                         {
546                                 if(BATMON_control_reg & B1)     // Button 1 = YES,
547                                 {
548                                         UART_menu_start();
549                                         BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
550                                         break;                                  // exit
                                                while loop
551                                 }
552                                 if(BATMON_control_reg & B2)     // Button 2 = NO,
553                                 {
554                                         BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
555                                         break;                                  // exit
                                                while loop
556                                 }
557                         }
558
559                         BATMON_menu_position = BATMON_MENU_EXIT;
560                         break;          // exit switch case
561
562                 case BATMON_MENU_STOP:
563                         LCD_send_cmd(LCD_LINE1);
564                         LCD_send_text(BATMON_menu_msg[2]);
565                         LCD_send_cmd(LCD_LINE2);
566                         LCD_send_text(BATMON_menu_msg[7]);
567
568                         while(1)
569                         {
570                                 if(BATMON_control_reg & B1)     // Button 1 = YES,
571                                 {
572                                         UART_menu_stop();
573                                         BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
574                                         break;                                  // exit
                                                while loop
575                                 }
576                                 if(BATMON_control_reg & B2)     // Button 2 = NO,
577                                 {
578                                         BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
579                                         break;                                  // exit
                                                while loop
580                                 }
581                         }
582
583                         BATMON_menu_position = BATMON_MENU_EXIT;
584                         break;          // exit switch case
585
586                 case BATMON_MENU_SAVE:
587                         LCD_send_cmd(LCD_LINE1);
588                         LCD_send_text(BATMON_menu_msg[4]);
589                         LCD_send_cmd(LCD_LINE2);
590                         LCD_send_text(BATMON_menu_msg[7]);
591
592                         while(1)
593                         {
594                                 if(BATMON_control_reg & B1)     // Button 1 = YES,
595                                 {
596                                         UART_menu_write_controll();
597                                         BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
598                                         break;                                  // exit
                                                while loop
599                                 }
600                                 if(BATMON_control_reg & B2)     // Button 2 = NO,
601                                 {
```

```
602                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                // clear button state
603                                     break;                                  // exit
                                            while loop
604                             }
605                     }
606
607                     BATMON_menu_position = BATMON_MENU_EXIT;
608                     break;          // exit switch case
609
610             case BATMON_MENU_LOAD:
611                     LCD_send_cmd(LCD_LINE1);
612                     LCD_send_text(BATMON_menu_msg[3]);
613                     LCD_send_cmd(LCD_LINE2);
614                     LCD_send_text(BATMON_menu_msg[7]);
615
616                     while(1)
617                     {
618                             if(BATMON_control_reg & B1)     // Button 1 = YES,
619                             {
620                                     UART_menu_read_controll();
621                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                // clear button state
622                                     break;                                  // exit
                                            while loop
623                             }
624                             if(BATMON_control_reg & B2)     // Button 2 = NO,
625                             {
626                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                // clear button state
627                                     break;                                  // exit
                                            while loop
628                             }
629                     }
630
631                     BATMON_menu_position = BATMON_MENU_EXIT;
632                     break;          // exit switch case
633
634             case BATMON_MENU_RTC:
635                     LCD_send_cmd(LCD_LINE1);                // clear first LCD row
636                     LCD_send_text("                ");
637                     RTC_init();                                             // enter
                            RTC init mode
638
639                     BATMON_menu_position = BATMON_MENU_EXIT;
640                     break;          // exit switch case
641
642             case BATMON_MENU_SHOW:
643                     time_tmp = RTC_values.sec + 5;
644
645                     if (time_tmp > 59)
646                             time_tmp -= 60;
647
648                     LCD_send_cmd(LCD_LINE1);
649                     LCD_send_text("                ");
650                     LCD_send_cmd(LCD_LINE2);
651                     LCD_send_text("                ");
652                     LCD_send_cmd(LCD_LINE1);
653
654                     while(time_tmp != RTC_values.sec)
655                     {
656                             for(i=0; i<SENSOR_active; i++)
657                             {
658                                     // jump to second line if maximum of line one
                                            reached
659                                     if(((i%16) == 0) && (i!=0)) LCD_send_cmd(
                                            LCD_LINE2);
```

```
660
661                                     // i/8 = byte-, i%8 = bit position in array
662                                     if((SENSOR_addr_req[(i/8)] >> (i%8)) & 0x01)
663                                             LCD_send_data('.');
664                                     else
665                                             LCD_send_data('*');
666                             }
667
668                             LCD_send_cmd(LCD_LINE1);
669                     }
670
671                     LCD_send_cmd(LCD_LINE1);
672                     LCD_send_text("                ");
673
674                     BATMON_menu_position = BATMON_MENU_EXIT;
675                     break;          // exit switch case
676
677             case BATMON_MENU_MAIN:
678                     LCD_send_cmd(LCD_LINE1);
679                     LCD_send_text(BATMON_menu_msg[0]);
680                     LCD_send_cmd(LCD_LINE2);
681                     LCD_send_text(BATMON_menu_msg[6]);
682
683                     while(1)
684                     {
685                             // Button B1 pressed, scroll through menu
686                             if(BATMON_control_reg & B1)
687                             {
688                                     if (++BATMON_menu_counter > 5)
689                                     {
690                                             BATMON_menu_counter = 0;
691                                     }
692
693                                     LCD_send_cmd(LCD_LINE1);
694                                     LCD_send_text(BATMON_menu_msg[BATMON_menu_counter
                                            ]);
695                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                     // clear button state
696                             }
697
698                             // Button B2 pressed, accept menu item
699                             else if(BATMON_control_reg & B2)
700                             {
701                                     switch (BATMON_menu_counter)
702                                     {
703                                     case 0:
704                                             BATMON_menu_position = BATMON_MENU_SHOW;
705                                             break;  // exit switch case
706                                     case 1:
707                                             BATMON_menu_position = BATMON_MENU_START;
708                                             break;  // exit switch case
709                                     case 2:
710                                             BATMON_menu_position = BATMON_MENU_STOP;
711                                             break;  // exit switch case
712                                     case 3:
713                                             BATMON_menu_position = BATMON_MENU_LOAD;
714                                             break;  // exit switch case
715                                     case 4:
716                                             BATMON_menu_position = BATMON_MENU_SAVE;
717                                             break;  // exit switch case
718                                     case 5:
719                                             BATMON_menu_position = BATMON_MENU_RTC;
720                                             break;  // exit switch case
721                                     }
722
723                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                     // clear button state
```

```
724                                     break;          // exit while loop
725                                 }
726                                 // Button B3 pressed, exit menu
727                                 else if(BATMON_control_reg & B3)
728                                 {
729                                     if (Enable_clear_lcd == ON)
730                                     {
731                                         LCD_send_cmd(LCD_LINE1);
732                                         LCD_send_text("                ");
733                                     }
734                                     else
735                                     {
736                                         LCD_send_cmd(LCD_LINE1);
737                                         LCD_send_text(" Radio disabled ");
738                                     }
739                                     LCD_send_time();
740                                     BATMON_menu_position = BATMON_MENU_EXIT;
741                                     BATMON_control_reg &= ~(B1|B2|B3);
                                                    // clear button state
742                                     break;          // exit while loop
743                                 }
744                             }
745                             break;          // exit switch case
746                 }
747         }
748         // set text in first LCD line
749         if (Enable_clear_lcd == OFF)
750         {
751                 LCD_send_cmd(LCD_LINE1);
752                 LCD_send_text(" Radio disabled ");
753         }
754
755         // set time on second LCD line
756         LCD_send_time();
757 }
```

Listing B.11: Header der Mikrocontroller spezifischen Funktionen (MSP_functions.h)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430 MSP_functions header
3          Used components:                MSP430-169STK
4          Date:                           02/14/2008
5          Last update:                    09/05/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------*/
8
9  #ifndef MSP_FUNCIONS_H_
10 #define MSP_FUNCIONS_H_
11
12 /*-------------------------------------------------------------------------------
13         Includes
14 -------------------------------------------------------------------------------*/
15 #include <RTC.h>
16 #include <ISR.h>
17 #include <flash.h>
18
19 /*-------------------------------------------------------------------------------
20         Defined values
21 -------------------------------------------------------------------------------*/
22 enum bmon_menu{ BATMON_MENU_START, BATMON_MENU_STOP, BATMON_MENU_LOAD,
23                                 BATMON_MENU_SAVE, BATMON_MENU_RTC, BATMON_MENU_EXIT,
24                                 BATMON_MENU_SHOW, BATMON_MENU_MAIN};
25
26
27 /*-------------------------------------------------------------------------------
28         Global variables
```

```
29 ------------------------------------------------------------------------------*/
30 extern unsigned char BATMON_control_reg;
31
32 extern unsigned char global_string[MAX_STRING_LENGTH];
33 extern unsigned char RADIO_rx_state;                                    // data receive
       state
34
35 extern RTC_MSP430                      RTC_values;
       // global RTC values
36 extern RADIO_FRAME_STRUCT       RADIO_frame;                           // received data
       frame
37
38 extern unsigned char SENSOR_addr_req[5];        // array for intervall mode, shows
       required address
39 extern unsigned char SENSOR_active;                     // active sensor count
40
41 /*------------------------------------------------------------------------------
42         Prototypes
43 ------------------------------------------------------------------------------*/
44
45 unsigned char DCO_set(unsigned int);
46 unsigned char XT2_set(unsigned int);
47
48 void PORTS_init(void);
49 void ADC0_init(void);
50 void USART0_init (unsigned char, unsigned char);
51
52 int  ADC0_sampling(void);
53 void UART0_send (unsigned char);
54 void UART0_send_text(char *);
55 unsigned char UART0_cmp_string(void);
56
57 void RADIO_init_frame(void);
58 void RADIO_init(void);
59 void RADIO_show_received_data(void);
60 void RADIO_store_frame(void);
61 void RADIO_enable(void);
62 void RADIO_disable(void);
63
64 void BATMON_menu_options(void);
65
66 void TIMERB_init(void);
67
68 unsigned char* DEZ2HEX(unsigned char);
69
70
71 #endif /*MSP_FUNCIONS_H_*/
```

Listing B.12: Quellcode der Real Time Clock (RTC.c)

```
1  /*------------------------------------------------------------------------------
2          Project:                        MSP430 RTC source file
3          Used components:                MSP430-169STK
4          Date:                           02/08/2008
5          Last update:                    09/05/2008
6          Author:                         Stephan Plaschke
7  ------------------------------------------------------------------------------
8
9  ------------------------------------------------------------------------------
10         Headerfiles
11 ------------------------------------------------------------------------------*/
12 #include <RTC.h>
13 #include <main.h>
14 #include <stdio.h>
15 #include <signal.h>             // needed for interrupt
16 #include <lcd16x2.h>
```

```
17 #include <msp430x16x.h>
18 #include <MSP_functions.h>
19
20
21 /*-------------------------------------------------------------------------
22         Static
23 -------------------------------------------------------------------------*/
24 static unsigned char *RTC_days[7]={
25                 "MON ",
26                 "TUE ",
27                 "WED ",
28                 "THU ",
29                 "FRY ",
30                 "SAT ",
31                 "SUN "
32 };
33
34 static unsigned char *RTC_months[12]={
35                 "Jan",
36                 "Feb",
37                 "Mar",
38                 "Apr",
39                 "May",
40                 "Jun",
41                 "Jul",
42                 "Aug",
43                 "Sep",
44                 "Oct",
45                 "Nov",
46                 "Dec"
47 };
48 /*-------------------------------------------------------------------------
49         RTC init
50 -------------------------------------------------------------------------*/
51 void RTC_init(void)
52 /*-------------------------------------------------------------------------*/
53 {
54         unsigned char time_char;
55
56         RTC_menue_position = RTC_MENUE_DAY;
57         BATMON_control_reg &= ~(B1 | B2 | B3);
58
59         /* initialise watchdog timer used for BLINKY, intervall mode, 1/4sec    */
60         WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTSSEL | WDTIS_1;       /* 1/4sec
               intervall mode        */
61         WDT_function = BLINKY;
62         IE1 |= WDTIE;                   /* enable Watchdog interrupt */
63
64         /* initialise TIMERB0, ACLK/1, up mode, capture/compare0 int*/
65         /*TBCTL         &= ~MC_3;                                                 //
               disable Timer_B
66         TBCTL   = TBSSEL_1 | TBCLR;                             //  set timer register,
               clear TAR
67         TBCCTL0 = CCIE;                                                 //  set control
               register
68         TBCCR0  = 500;
69         TBCTL   |= MC_1;                                                // start Timer_B
70         */
71         /* global interrupt enable      */
72         _EINT();
73
74         /* display real time and weekday */
75         LCD_send_date();
76 /*-------------------------------------------------------------------------
77  Edit the date
78 -------------------------------------------------------------------------*/
79         do
```

```
80          {
81  /* if B1 and B2 are pressed exit the RTC_EDIT_MODE      */
82                  if ((BATMON_control_reg & B1) && (BATMON_control_reg & B2))
83                  {
84                          BATMON_control_reg &= ~(B1 | B2);
85                          RTC_menue_position = RTC_MENUE_EXIT;
86                  }
87  /* only B1 is pressed decrement the editable value(day,sec,min,hr)      */
88                  else if(BATMON_control_reg & B1)
89                  {
90                                  BATMON_control_reg &= ~B1;              // clear button
                                        state variable
91                                  LED2_TOGGLE;
92                                  switch(RTC_menue_position)
93                                  {
94                                  case RTC_MENUE_DAY: if(RTC_values.day == 1) RTC_values.
                                        day = 31;
95                                          else RTC_values.day--;
96                                          break;
97                                  case RTC_MENUE_MONTH: if(RTC_values.month == 0)
                                        RTC_values.month = 11;
98                                          else RTC_values.month--;
99                                          break;
100                                 case RTC_MENUE_YEAR: if(RTC_values.year == 0) RTC_values.
                                        year = 4000;
101                                         else RTC_values.year--;
102                                         break;
103                                 default: break;
104                                 }
105                 }
106 /* only B2 is pressed increment the editable value(day,sec,min,hr)      */
107                 else if(BATMON_control_reg & B2)
108                 {
109                                 BATMON_control_reg &= ~B2;              // clear button
                                        state variable
110                                 LED2_TOGGLE;
111                                 switch(RTC_menue_position)
112                                 {
113                                 case RTC_MENUE_DAY: if(RTC_values.day == 31) RTC_values.
                                        day = 1;
114                                         else RTC_values.day++;
115                                         break;
116                                 case RTC_MENUE_MONTH: if(RTC_values.month == 11)
                                        RTC_values.month = 0;
117                                         else RTC_values.month++;
118                                         break;
119                                 case RTC_MENUE_YEAR: if(RTC_values.year == 4000)
                                        RTC_values.year = 0;
120                                         else RTC_values.year++;
121                                         break;
122                                 default: break;
123                                 }
124                 }
125 /* only B3 is pressed change the editable value */
126                 else if(BATMON_control_reg & B3)
127                 {
128                         BATMON_control_reg &= ~B3;      // clear button state variable
129                         LED2_TOGGLE;
130                         /* display previews editable value */
131                         switch(RTC_menue_position)
132                         {
133                                 case RTC_MENUE_DAY: LCD_send_cmd(LCD_RTC_DAY);
134                                         time_char = (RTC_values.day / 10) + 48;
135                                         LCD_send_data(time_char);
136                                         time_char = (RTC_values.day % 10) + 48;
137                                         LCD_send_data(time_char);
138                                         break;
```

```
139                                 case RTC_MENUE_MONTH: LCD_send_cmd(LCD_RTC_MONTH);
140                                         LCD_send_text(RTC_months[RTC_values.month]);
141                                         break;
142                                 case RTC_MENUE_YEAR: LCD_send_cmd(LCD_RTC_YEAR);
143                                         time_char = RTC_values.year / 1000;
144                                         LCD_send_data(time_char+48);
145                                         time_char = (RTC_values.year / 100) - ((
                                                RTC_values.year/1000) * 10);
146                                         LCD_send_data(time_char+48);
147                                         time_char = (RTC_values.year / 10) - ((RTC_values
                                                .year/100) * 10);
148                                         LCD_send_data(time_char+48);
149                                         time_char = RTC_values.year % 10;
150                                         LCD_send_data(time_char+48);
151                                         break;
152                                 default: break;
153                         }
154                         /* change editable value */
155                         if(RTC_menue_position == RTC_MENUE_YEAR)
156                                 RTC_menue_position = RTC_MENUE_DAY;
157                         else
158                                 RTC_menue_position++;
159                 }
160         }
161         while(RTC_menue_position != RTC_MENUE_EXIT);
162
163 /*-------------------------------------------------------------------------------
164  Edit the weekday and time
165 -------------------------------------------------------------------------------*/
166         /* display real time and weekday */
167         IE1 &= ~WDTIE;                                                  // disable
                Watchdog interrupt
168         RTC_menue_position = RTC_MENUE_WDAY;
169         LCD_send_time();
170         IE1 |= WDTIE;                   // enable Watchdog interrupt
171
172         do
173         {
174 /* if B1 and B2 are pressed exit the RTC_EDIT_MODE      */
175                 if ((BATMON_control_reg & B1) && (BATMON_control_reg &B2))
176                 {
177                         BATMON_control_reg &= ~(B1 | B2);
178                         RTC_menue_position = RTC_MENUE_EXIT;
179                 }
180 /* only B1 is pressed decrement the editable value(day,sec,min,hr)      */
181                 else if(BATMON_control_reg & B1)
182                 {
183                                 BATMON_control_reg &= ~B1;              // clear button
                                        state variable
184                                 LED2_TOGGLE;
185                                 switch(RTC_menue_position)
186                                 {
187                                 case RTC_MENUE_WDAY: if(RTC_values.wday == MON)
                                        RTC_values.wday = SUN;
188                                         else RTC_values.wday--;
189                                         break;
190                                 case RTC_MENUE_HR: if(RTC_values.hr == 0) RTC_values.hr =
                                         23;
191                                         else RTC_values.hr--;
192                                         break;
193                                 case RTC_MENUE_MIN: if(RTC_values.min == 0) RTC_values.
                                        min = 59;
194                                         else RTC_values.min--;
195                                         break;
196                                 case RTC_MENUE_SEC: if(RTC_values.sec == 0) RTC_values.
                                        sec = 59;
197                                         else RTC_values.sec--;
```

```
198                                     break;
199                                 default: break;
200                                 }
201                 }
202 /* only B2 is pressed increment the editable value(day,sec,min,hr)      */
203                 else if(BATMON_control_reg & B2)
204                 {
205                                 BATMON_control_reg &= ~B2;              // clear button
                                        state variable
206                                 LED2_TOGGLE;
207                                 switch(RTC_menue_position)
208                                 {
209                                 case RTC_MENUE_WDAY: if(RTC_values.wday == SUN)
                                        RTC_values.wday = MON;
210                                         else RTC_values.wday++;
211                                         break;
212                                 case RTC_MENUE_HR: if(RTC_values.hr == 23) RTC_values.hr
                                        = 0;
213                                         else RTC_values.hr++;
214                                         break;
215                                 case RTC_MENUE_MIN: if(RTC_values.min == 59) RTC_values.
                                        min = 0;
216                                         else RTC_values.min++;
217                                         break;
218                                 case RTC_MENUE_SEC: if(RTC_values.sec == 59) RTC_values.
                                        sec = 0;
219                                         else RTC_values.sec++;
220                                         break;
221                                 default: break;
222                                 }
223                 }
224 /* only B3 is pressed change the editable value */
225                 else if(BATMON_control_reg & B3)
226                 {
227                         BATMON_control_reg &= ~B3;      // clear button state variable
228                         LED2_TOGGLE;
229                         /* display previews editable value */
230                         switch(RTC_menue_position)
231                         {
232                                 case RTC_MENUE_WDAY: LCD_send_cmd(LCD_RTC_WDAY);
233                                         LCD_send_text(RTC_days[RTC_values.wday]);
234                                         break;
235                                 case RTC_MENUE_HR: LCD_send_cmd(LCD_RTC_HR);
236                                         time_char = (RTC_values.hr / 10) + 48;
237                                         LCD_send_data(time_char);
238                                         time_char = (RTC_values.hr % 10) + 48;
239                                         LCD_send_data(time_char);
240                                         break;
241                                 case RTC_MENUE_MIN: LCD_send_cmd(LCD_RTC_MIN);
242                                         time_char = (RTC_values.min / 10) + 48;
243                                         LCD_send_data(time_char);
244                                         time_char = (RTC_values.min % 10) + 48;
245                                         LCD_send_data(time_char);
246                                         break;
247                                 case RTC_MENUE_SEC: LCD_send_cmd(LCD_RTC_SEC);
248                                         time_char = (RTC_values.sec / 10) + 48;
249                                         LCD_send_data(time_char);
250                                         time_char = (RTC_values.sec % 10) + 48;
251                                         LCD_send_data(time_char);
252                                         break;
253                                 default: break;
254                         }
255                         /* change editable value */
256                         if(RTC_menue_position == RTC_MENUE_SEC)
257                                 RTC_menue_position = RTC_MENUE_WDAY;
258                         else
259                                 RTC_menue_position++;
```

```
260                 }
261         }
262         while(RTC_menue_position != RTC_MENUE_EXIT);
263 
264         /* initialise watchdog timer used for RTC, intervall mode, 1sec */
265         WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTSSEL;         /* 1sec intervall mode
                */
266         WDT_function = RTC;
267         IE1 |= WDTIE;
                    /* enable Watchdog interrupt    */
268 
269         /* global interrupt disable     */
270         //_DINT();
271 
272         // switch LEDS off
273         LED1_OFF;
274         LED2_OFF;
275 
276         /* display real time and day */
277         LCD_send_time();
278 
279         if (Enable_clear_lcd == OFF)
280         {
281                 LCD_send_cmd(LCD_LINE1);
282                 LCD_send_text(" Radio disabled ");
283         }
284 }
285 
286 /*-------------------------------------------------------------------------------
287         RTC compare tmp and real time
288 -------------------------------------------------------------------------------*/
289 void RTC_compare(RTC_MSP430 *RTC_tmp)
290 /*-------------------------------------------------------------------------------*/
291 {
292         unsigned char time_char;
293 
294         /* compare old RTC values with real values */
295         if(RTC_values.wday != RTC_tmp->wday)
296         {
297                 RTC_tmp->wday = RTC_values.wday;
298                 LCD_send_cmd(LCD_RTC_WDAY);
299                 LCD_send_text(RTC_days[RTC_values.wday]);
300         }
301         if(RTC_values.hr != RTC_tmp->hr)
302         {
303                 RTC_tmp->hr = RTC_values.hr;
304                 LCD_send_cmd(LCD_RTC_HR);
305                 time_char = (RTC_values.hr / 10) + 48;
306                 LCD_send_data(time_char);
307                 time_char = (RTC_values.hr % 10) + 48;
308                 LCD_send_data(time_char);
309         }
310         if(RTC_values.min != RTC_tmp->min)
311         {
312                 RTC_tmp->min = RTC_values.min;
313                 LCD_send_cmd(LCD_RTC_MIN);
314                 time_char = (RTC_values.min / 10) + 48;
315                 LCD_send_data(time_char);
316                 time_char = (RTC_values.min % 10) + 48;
317                 LCD_send_data(time_char);
318         }
319         if(RTC_values.sec != RTC_tmp->sec)
320         {
321                 RTC_tmp->sec = RTC_values.sec;
322                 LCD_send_cmd(LCD_RTC_SEC);
323                 time_char = (RTC_values.sec / 10) + 48;
324                 LCD_send_data(time_char);
```

```
325                 time_char = (RTC_values.sec % 10) + 48;
326                 LCD_send_data(time_char);
327         }
328
329 }
330
331
332 /*-------------------------------------------------------------------------------
333         RTC startup, set time/date or init timer only
334 -------------------------------------------------------------------------------*/
335 void RTC_startup(void)
336 //-------------------------------------------------------------------------------
337 {
338         LCD_send_cmd(LCD_LINE1);
339         LCD_send_text(" Set date/time  ");
340         LCD_send_cmd(LCD_LINE2);
341         LCD_send_text("YES    NO       ");
342
343         while(1)
344         {
345                 if(BATMON_control_reg & B1)     // Button 1 = YES,
346                 {
347                         RTC_init();
348                         BATMON_control_reg &= ~(B1|B2|B3);                      // clear
                                button state
349                         break;                                  // exit while loop
350                 }
351                 if(BATMON_control_reg & B2)     // Button 2 = NO,
352                 {
353                         /* initialise watchdog timer used for RTC, intervall mode, 1sec
                                */
354                         WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDTSSEL;         /* 1sec
                                intervall mode  */
355                         WDT_function = RTC;
356                         IE1 |= WDTIE;
                                                        /* enable Watchdog interrupt    */
357
358                         /* display real time and day */
359                         LCD_send_time();
360                         BATMON_control_reg &= ~(B1|B2|B3);                      // clear
                                button state
361                         break;                                  // exit while loop
362                 }
363         }
364 }
365
366 /*-------------------------------------------------------------------------------
367         Watchdog interrupt service routine
368 -------------------------------------------------------------------------------*/
369 interrupt (WDT_VECTOR) WATCH_DOG(void)
370 /*-------------------------------------------------------------------------------*/
371 {
372         static uint8_t WDT_tmp;
373         static uint8_t BLINKY_state;
374
375         unsigned char time_char, j, address_byte, address_bit, i;
376
377         switch(WDT_function)
378         {
379 /* 1 sec timer for the real time clock  */
380         case RTC:
381                 if(++RTC_values.sec == 60)                              // 60 seconds
                        reached?
382                 {
383                         RTC_values.sec = 0;
384
385                         for (i=0; i<SENSOR_active; i++)
```

```
386                         {
387                                 address_byte = i / 8;
388                                 address_bit = i % 8 ;
389
390                                 if (SENSOR_addr_req[address_byte] & (0x01 << address_bit)
                                        )
391                                 {
392                                         for (j=0; j<8; j++)
393                                         {
394                                                 RADIO_frame.frame_byte[j+3] =
                                                        SENSOR_last_data[j][i];
395                                         }
396
397                                         RADIO_store_frame();
398                                 }
399                         }
400                 // set sensors active every 60 seconds
401                         switch(SENSOR_active)
402                         {
403                         case 12:                                                    // 12
                                sensors are configured
404                                 SENSOR_addr_req[0] = 0xFF;      // set required sensors
                                    values
405                                 SENSOR_addr_req[1] = 0x0F;
406                                 break;
407                         case 24:                                                    // 24
                                sensors are configured
408                                 SENSOR_addr_req[0] = 0xFF;      // set required sensors
409                                 SENSOR_addr_req[1] = 0xFF;
410                                 SENSOR_addr_req[2] = 0xFF;
411                                 break;
412                         case 40:                                                    // 24
                                sensors are configured
413                                 SENSOR_addr_req[0] = 0xFF;      // set required sensors
414                                 SENSOR_addr_req[1] = 0xFF;
415                                 SENSOR_addr_req[2] = 0xFF;
416                                 SENSOR_addr_req[3] = 0xFF;
417                                 SENSOR_addr_req[4] = 0xFF;
418                                 break;
419                         default:                                                    // 6
                                sensors are configured
420                                 SENSOR_addr_req[0] = 0x3F;      // set required sensors
421                                 break;
422                         }
423
424                         if(++RTC_values.min == 60)
                                // 60 minutes reached?
425                         {
426                                 RTC_values.min = 0;
427                                 if(++RTC_values.hr == 24)
                                        // 0 o'clock reached?
428                                 {
429                                         RTC_values.hr = 0;
430                                         if ((RTC_values.year % 4) == 0)
                                                        // leap year
431                                         {
432                                                 RTC_values.day++;
433                                                 if((RTC_values.month == 1)&&(RTC_values.
                                                        day==30))       // february
434                                                 {
435                                                         RTC_values.month++;
436                                                         RTC_values.day = 1;
437                                                 }
438                                         }
439                                         else
                                                                                    // normal
                                            year
```

```
440                                         {
441                                                 RTC_values.day++;
442                                                 if((RTC_values.month == 1)&&(RTC_values.
                                                        day==29))
443                                                 {
444                                                         RTC_values.month++;
445                                                         RTC_values.day = 1;
446                                                 }
447                                         }
448
449                                         // 30 day update (April, June, August, October,
                                                December)
450                                         if (((RTC_values.month == 3) || (RTC_values.month
                                              == 5) || (RTC_values.month == 7) ||
451                                                         (RTC_values.month == 9) || (
                                                             RTC_values.month == 11)) &&
                                                             RTC_values.day == 31)
452                                         {
453                                                 RTC_values.day = 1;
454                                                 RTC_values.month++;
455                                         }
456
457                                         // 31 day update (january, March, Mai, July,
                                                September, November)
458                                         if (RTC_values.day == 32)
459                                         {
460                                                 RTC_values.day = 1;
461                                                 RTC_values.month++;
462                                         }
463
464                                         // year update
465                                         if(RTC_values.month == 12)
466                                         {
467                                                 RTC_values.month = 0;
468                                                 RTC_values.year++;
469                                         }
470
471                                         // weekday update
472                                         if(++RTC_values.wday == (SUN+1))
473                                                 RTC_values.wday = MON;
474                                 }
475                         }
476                 }
477
478                 if (Enable_clear_lcd & ON)              // if radio enabled clear LCD/
                        reset rx process
479                 {
480                         // Clear LCD active message every 4 seconds
481                         if( (!((RTC_values.sec+1) % 4)))
482                                 BATMON_control_reg |= CLEAR_FIRST_LCD_ROW;
483
484                         // reset rx process every 30 seconds
485                         if(!((RTC_values.sec+1) % 30))
486                                 RADIO_enable();
487                 }
488                 break;
489 /* switch every 0,5 sec the editable value on or off, only in RTC_EDIT_MODE     */
490         case BLINKY:
491                 if(++WDT_tmp == 2)
492                 {
493                         WDT_tmp = 0;
494                         LED1_TOGGLE;
495                         if(BLINKY_state ^= 0x01)
496                         {
497                                 switch (RTC_menue_position)
498                                 {
499                                 case RTC_MENUE_WDAY: LCD_send_cmd(LCD_RTC_WDAY);
```

```
500                                     LCD_send_text(RTC_days[RTC_values.wday]);
501                                     break;
502                             case RTC_MENUE_HR: LCD_send_cmd(LCD_RTC_HR);
503                                     time_char = (RTC_values.hr / 10) + 48;
504                                     LCD_send_data(time_char);
505                                     time_char = (RTC_values.hr % 10) + 48;
506                                     LCD_send_data(time_char);
507                                     break;
508                             case RTC_MENUE_MIN: LCD_send_cmd(LCD_RTC_MIN);
509                                     time_char = (RTC_values.min / 10) + 48;
510                                     LCD_send_data(time_char);
511                                     time_char = (RTC_values.min % 10) + 48;
512                                     LCD_send_data(time_char);
513                                     break;
514                             case RTC_MENUE_SEC: LCD_send_cmd(LCD_RTC_SEC);
515                                     time_char = (RTC_values.sec / 10) + 48;
516                                     LCD_send_data(time_char);
517                                     time_char = (RTC_values.sec % 10) + 48;
518                                     LCD_send_data(time_char);
519                                     break;
520                             case RTC_MENUE_DAY: LCD_send_cmd(LCD_RTC_DAY);
521                                     time_char = (RTC_values.day / 10) + 48;
522                                     LCD_send_data(time_char);
523                                     time_char = (RTC_values.day % 10) + 48;
524                                     LCD_send_data(time_char);
525                                     break;
526                             case RTC_MENUE_MONTH: LCD_send_cmd(LCD_RTC_MONTH);
527                                     LCD_send_text(RTC_months[RTC_values.month]);
528                                     break;
529                             case RTC_MENUE_YEAR: LCD_send_cmd(LCD_RTC_YEAR);
530                                     time_char = RTC_values.year / 1000;
531                                     LCD_send_data(time_char+48);
532                                     time_char = (RTC_values.year / 100) - ((
                                            RTC_values.year/1000) * 10);
533                                     LCD_send_data(time_char+48);
534                                     time_char = (RTC_values.year / 10) - ((RTC_values
                                            .year/100) * 10);
535                                     LCD_send_data(time_char+48);
536                                     time_char = RTC_values.year % 10;
537                                     LCD_send_data(time_char+48);
538                                     break;
539                             }
540                     }
541                     else
542                     {
543                             switch (RTC_menue_position)
544                             {
545                             case RTC_MENUE_WDAY: LCD_send_cmd(LCD_RTC_WDAY);
546                                     LCD_send_text("   ");
547                                     break;
548                             case RTC_MENUE_HR: LCD_send_cmd(LCD_RTC_HR);
549                                     LCD_send_text("  ");
550                                     break;
551                             case RTC_MENUE_MIN: LCD_send_cmd(LCD_RTC_MIN);
552                                     LCD_send_text("  ");
553                                     break;
554                             case RTC_MENUE_SEC: LCD_send_cmd(LCD_RTC_SEC);
555                                     LCD_send_text("  ");
556                                     break;
557                             case RTC_MENUE_DAY: LCD_send_cmd(LCD_RTC_DAY);
558                                     LCD_send_text("  ");
559                                     break;
560                             case RTC_MENUE_MONTH: LCD_send_cmd(LCD_RTC_MONTH);
561                                     LCD_send_text("   ");
562                                     break;
563                             case RTC_MENUE_YEAR: LCD_send_cmd(LCD_RTC_YEAR);
564                                     LCD_send_text("    ");
```

```
565                                         break;
566                                 }
567                         }
568                 }
569                 break;
570         default: break;
571         }
572
573         IFG1 &= ~WDTIFG;        /* clear watchdog interrupt flag        */
574 }
```

Listing B.13: Header der Real Time Clock (RTC.h)

```
1  /*-------------------------------------------------------------------------
2          Project:                        MSP430 RTC header
3          Used components:                MSP430-169STK
4          Date:                           02/08/2008
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------*/
8
9  #ifndef RTC_H_
10 #define RTC_H_
11
12 /*-------------------------------------------------------------------------
13         Structures
14 -------------------------------------------------------------------------*/
15 typedef struct{
16             unsigned char hr;
17                 unsigned char min;
18                 unsigned char sec;
19                 unsigned char wday;
20                 unsigned char day;
21                 unsigned char month;
22                 unsigned int  year;
23 } RTC_MSP430;
24
25 /*-------------------------------------------------------------------------
26         Global variables
27 -------------------------------------------------------------------------*/
28
29 extern RTC_MSP430 RTC_values;                           // global RTC values
30
31 extern unsigned char WDT_function;                      // WDT function, BLINKY for RTC
       SETUP or RTC for RTC mode
32 extern unsigned char RTC_menue_position;        // menue position in RTC setup
33 extern unsigned char RADIO_rx_state;            // data receive state
34
35 extern unsigned char Enable_clear_lcd;
36 extern unsigned char SENSOR_active;
37
38 extern unsigned char SENSOR_addr_req[5];
39 extern unsigned char SENSOR_addr_get[5];
40 extern unsigned char SENSOR_address[40];
41 extern unsigned char SENSOR_last_data[8][40];
42
43 extern unsigned char BATMON_control_reg;        // global control register
44
45 /*-------------------------------------------------------------------------
46         Defines
47 -------------------------------------------------------------------------*/
48 enum week {MON, TUE, WED, THU, FRY, SAT, SUN};
49 enum wdt_state {RTC, BLINKY};
50 enum rtc_edit_menue {RTC_MENUE_WDAY, RTC_MENUE_HR, RTC_MENUE_MIN, RTC_MENUE_SEC,
51                                         RTC_MENUE_DAY, RTC_MENUE_MONTH, RTC_MENUE_YEAR,
                                                RTC_MENUE_EXIT};
```

```
52
53
54 /*      clock position on LCD   */
55 #define LCD_RTC_SEC             (LCD_LINE2+10)
56 #define LCD_RTC_MIN             (LCD_LINE2+7)
57 #define LCD_RTC_HR              (LCD_LINE2+4)
58 #define LCD_RTC_WDAY     LCD_LINE2
59 #define LCD_RTC_YEAR    (LCD_LINE2+7)
60 #define LCD_RTC_MONTH   (LCD_LINE2+3)
61 #define LCD_RTC_DAY             LCD_LINE2
62
63 /*-------------------------------------------------------------------------------
64         Prototypes
65 -------------------------------------------------------------------------------*/
66 void RTC_init(void);
67 void RTC_compare(RTC_MSP430 *);
68 void RTC_startup(void);
69
70
71
72 #endif /*RTC_H_*/
```

Listing B.14: Quellcode der Menüfunktionen (UART_menu.c)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        UART menu
3          Used components:                MSP430-169STK
4          Date:                           04/04/2008
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------
8
9  -------------------------------------------------------------------------------
10         Headerfiles
11 -------------------------------------------------------------------------------*/
12 #include <ISR.h>
13 #include <main.h>
14 #include <stdio.h>
15 #include <flash.h>
16 #include <stdlib.h>
17 #include <string.h>
18 #include <lcd16x2.h>
19 #include <uart_menu.h>
20 #include <msp430x16x.h>
21 #include <MSP_functions.h>
22
23
24 /*-------------------------------------------------------------------------------
25         Static
26 -------------------------------------------------------------------------------*/
27 //      helpscreen and UART menu information
28 static unsigned char *UART_menu_cmd[9]={
29         "\r\n\tBattery monitor help screen\r\n"
30         "\tEnter commands via the Hyperterminal\n\r"
31         "\tShortcuts are displayed in brackets\n\r"
32         "\tCapital and small letters are allowed\n\r"
33         "\tJump back from each submenu via EXIT (ex)\n\n\r"
34         "Help\t (he) :\tDisplay this screen\n\r"
35         "Realtime (rtc):\tClock time adjustment via UART\n\r"
36         "Sensors\t (se) :\tSet the address of each used sensor\n\r"
37         "Memory\t (mem):\tRead the complete flash-memory of the receiver\n\r"
38         "Clear\t (clr):\tClear the flash-memory \n\r"
39         "Start\t (st) :\tStart measurement\n\r"
40         "Stop\t (stp):\tStop measuremnt\n\r"
41         "Read\t (rd) :\tRead controll data from flash\n\r"
42         "Write\t (wr) :\tWrite controll data to flash\n\n\r",
```

```
43
44          "Enter time in following format:\r\n"
45          "WDAY,HR,MIN,SEC (d,hh,mm,ss: 0->Monday, 6->Sunday)\r\n"
46          "Jump back via >EXIT< or jump further with >DATE<\r\n",
47
48          "Enter date in following format:\r\n"
49          "DAY,MONTH,YEAR (dd,mm,yyyy: 01->January, 12->December)\r\n",
50
51          "Enter number of Sensors (6,12,24,40) and confirm with >ENTER<\r\n",
52
53          "Enter address of one sensor and confirm with >ENTER<\r\n",
54
55          "Read memory\r\n",
56
57          "Clear memory\r\n",
58
59          "Illegal command, type >help< for more information\r\n",
60
61          "Jump back to main menu\r\n"
62
63  };
64
65  //      first char in an UART command
66  static unsigned char *UART_menu_cmp_1st_char = "chrsmw" ;
67
68  //      shortcut for weekdays
69  static unsigned char *RTC_days[7]={
70                  "MON ",
71                  "TUE ",
72                  "WED ",
73                  "THU ",
74                  "FRY ",
75                  "SAT ",
76                  "SUN "
77  };
78
79  // shortcuts for months
80  static unsigned char *RTC_months[12]={
81                  "Jan",
82                  "Feb",
83                  "Mar",
84                  "Apr",
85                  "May",
86                  "Jun",
87                  "Jul",
88                  "Aug",
89                  "Sep",
90                  "Oct",
91                  "Nov",
92                  "Dec"
93  };
94
95  //      structure arrays for each first char, 1st element compare string, 2nd element
        pointer to function
96  //      if 1st char found compare string with received string, if this matches execute
        function where pointer points at
97  //      each string can contain capital and small letters
98  static UART_MEN UART_menu_C[]={
99                  {"lr", UART_menu_flash_clr},    // flash clear command
100                 {"lear", UART_menu_flash_clr},  // flash clear command
101                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
102 };
103
104 static UART_MEN UART_menu_H[]={
105                 {"e", UART_menu_show_help},             // show help screen
106                 {"elp", UART_menu_show_help},   // show help screen
```

```
107                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
108 };
109
110 static UART_MEN UART_menu_R[]={
111                 {"tc", UART_menu_setrtc},               // set rtc, jump in submenu
112                 {"ealtime", UART_menu_setrtc},  // set rtc, jump in submenu
113                 {"d", UART_menu_read_controll},                 // read controll data
                        from flash
114                 {"ead", UART_menu_read_controll},               // read controll data
                        from flash
115                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
116 };
117
118 static UART_MEN UART_menu_S[]={
119                 {"e", UART_menu_setsensor},             // set sensor address, jump in
                        submenu
120                 {"ensor", UART_menu_setsensor}, // set sensor address, jump in submenu
121                 {"t", UART_menu_start},                 // start measurement
122                 {"tart", UART_menu_start},              // start measurement
123                 {"tp", UART_menu_stop},                 // stop measurement
124                 {"top", UART_menu_stop},                // stop measurement
125                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
126 };
127
128 static UART_MEN UART_menu_M[]={
129                 {"em", UART_menu_read_data},    // set sensor address, jump in submenu
130                 {"emory", UART_menu_read_data},// set sensor address, jump in submenu
131                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
132 };
133
134 static UART_MEN UART_menu_W[]={
135                 {"r", UART_menu_write_controll},        // set sensor address, jump in
                        submenu
136                 {"rite", UART_menu_write_controll},// set sensor address, jump in submenu
137                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
138 };
139
140 static UART_MEN UART_menu_err[]={
141                 {"\0", UART_menu_error},                // display error message, wrong
                        command...
142                 {"\0", 0}                                                   // dummy to see
                        the end of array of structs
143 };
144
145 //      array of pointer on array of structs
146 //      used to find a valid command, compare each command in array of structs untill
        dummy element is found
147 static UART_MEN *pUART_menu[]={
148                 UART_menu_C,
149                 UART_menu_H,
150                 UART_menu_R,
151                 UART_menu_S,
152                 UART_menu_M,
153                 UART_menu_W,
154                 UART_menu_err
155 };
156
157 /*---------------------------------------------------------------------------
158         compare received string from UART
159 ---------------------------------------------------------------------------*/
160 unsigned char UART0_cmp_string(void)
161 /*---------------------------------------------------------------------------*/
```

```
162 {
163         static unsigned char sensor_address_index;
164
165         UART_MEN *pUART_menu_sub;
166
167         unsigned char *pglobal_string = global_string;
168         unsigned char *pUART_menu_cmp_1st_char = UART_menu_cmp_1st_char;
169         unsigned char UART_cmd_counter=0;
170         unsigned char convert_string[4];
171         unsigned int UART0_tmp;
172         unsigned char i;
173
174         switch (UART_menu_position)
175         {
176 //-------------------------------------------------------------------------------
177 //-------------------------------------------------------------------------------
178         case  UART_MENU_FIRSTCHAR:
                          // compare first char
179                 sensor_address_index=0;
180                 if ((*pglobal_string > 48) & (*pglobal_string < 122))
181                 {
182                                 // look for first command character in small or capital
                                    letters
183                                 // used is the chararray with 6 letters, if all 6 letters
                                     aren't
184                                 // found exit with error
185                                 do
186                                 {
187                                         if ((*pglobal_string == *pUART_menu_cmp_1st_char)
                                                 |
188                                            (*pglobal_string == *
                                                pUART_menu_cmp_1st_char-32))
189                                                 break;
                                                                 // exit do while()
                                                                    loop
190                                         pUART_menu_cmp_1st_char++;              // counts
                                             compare char one up
191                                         UART_cmd_counter++;
192                                 }
193                                 while(UART_cmd_counter < 7);
194
195                                 pUART_menu_sub = pUART_menu[UART_cmd_counter];
                                    // set pointer on control strukture
196
197                                 // if no valid character detected return with error
198                                 if (pUART_menu_sub[0].cmd == '\0')
199                                 {
200                                         pUART_menu_sub[0].fct();
201                                         break;
                                            // exit switch case
202                                 }
203
204                                 pglobal_string++;                               // set
                                    global pointer on second char
205
206                                 // compare global string with command string
207                                 // strcasecmp() compares all characters in command string
208                                 // with all characters in control structure(control
                                    structure is the
209                                 // structure with the valid first char), exit with valid
                                    command or
210                                 // when dummy in strucutre found (error)
211                                 do
212                                 {
213                                         if( strcasecmp( pUART_menu_sub[0].cmd,
                                            pglobal_string ) == 0 )
214                                         {
```

```
215                                             if( pUART_menu_sub[0].fct != NULL )
216                                             {
217                                                     (*(pUART_menu_sub[0].fct))();
                                                            // execute function
218                                             return (0);
                                                                    // return with no
                                                    error
219                                             }
220                                     }
221                                     pUART_menu_sub++;
222                             }while(*pUART_menu_sub[0].cmd != '\0');
223
224                             UART_menu_error();      // display error
225
226             }
227             else
228                     UART_menu_error();              // display error
229
230             break;                                          // exit switch case
231
232 //-------------------------------------------------------------------------------
233 //-------------------------------------------------------------------------------
234         case UART_MENU_CLOCK:                   // set rtc via UART
235                 UART_cmd_counter = strlen(pglobal_string);
236
237                 // check string for "EXIT"
238                 // if exit jump back to main menu
239                 if (strcasecmp(pglobal_string, "EXIT")==0)
240                 {
241                         UART0_send_text(UART_menu_cmd[8]);
242                         UART_menu_position = UART_MENU_FIRSTCHAR;
243                         return(0);
244                 }
245                 // check string for "DATE"
246                 // if exit jump back to main menu
247                 else if (strcasecmp(pglobal_string, "DATE")==0)
248                 {
249                         UART0_send_text(UART_menu_cmd[2]);
250                         UART_menu_position = UART_MENU_DATE;
251                         return(0);
252                 }
253                 // string to short, error
254                 else if (UART_cmd_counter<10)
255                 {
256                         UART_menu_setrtc();             // display error
257                         break;
258                 }
259                 // check each char
260                 while(((*pglobal_string > 43) && (*pglobal_string < 58)))
261                 {
262                         UART_cmd_counter--;
263                         pglobal_string++;               // set global pointer on next
                                char
264                 }
265                 // if only one char isn't ok, error
266                 if(UART_cmd_counter != 0)
267                 {
268                         UART_menu_setrtc();             // display error
269                         break;
270                 }
271
272                 // convert weekday string to int, if value valid store else exit with
                    error
273                 convert_string[0] = '0';
274                 convert_string[1] = global_string[0];
275                 UART0_tmp = atoi(convert_string);
276                 if ((UART0_tmp >= 0)&&(UART0_tmp < 7))
```

```
277                                 RTC_values.wday = (unsigned char)UART0_tmp;
278                 else
279                 {
280                         UART_menu_setrtc();             // display error
281                         break;
282                 }
283
284                 // convert hour string to int, if value valid store else exit with error
285                 convert_string[0] = global_string[2];
286                 convert_string[1] = global_string[3];
287                 UART0_tmp = atoi(convert_string);
288                 if ((UART0_tmp >= 0)&&(UART0_tmp < 24))
289                         RTC_values.hr = (unsigned char)UART0_tmp;
290                 else
291                 {
292                         UART_menu_setrtc();             // display error
293                         break;
294                 }
295
296                 // convert minute string to int, if value valid store else exit with
                        error
297                 convert_string[0] = global_string[5];
298                 convert_string[1] = global_string[6];
299                 UART0_tmp = atoi(convert_string);
300                 if ((UART0_tmp >= 0)&&(UART0_tmp < 60))
301                         RTC_values.min = (unsigned char)UART0_tmp;
302                 else
303                 {
304                         UART_menu_setrtc();             // display error
305                         break;
306                 }
307
308                 // convert seconds string to int, if value valid store else exit with
                        error
309                 convert_string[0] = global_string[8];
310                 convert_string[1] = global_string[9];
311                 UART0_tmp = atoi(convert_string);
312                 if ((UART0_tmp >= 0)&&(UART0_tmp < 60))
313                         RTC_values.sec = (unsigned char)UART0_tmp;
314                 else
315                 {
316                         UART_menu_setrtc();             // display error
317                         break;
318                 }
319
320                 UART0_send_text(UART_menu_cmd[2]);                      // print next
                        command, enter date
321                 UART_menu_position = UART_MENU_DATE;
322                 return(0);
                                // return with no error
323
324                 break;
325
326 //-------------------------------------------------------------------------------
327 //-------------------------------------------------------------------------------
328         case UART_MENU_DATE:                    // set rtc via UART
329                 UART_cmd_counter = strlen(pglobal_string);
330
331                 // compare string if "EXIT" send and if the length and information iss OK
332                 // if exit jump back to main menu
333                 if (strcasecmp(pglobal_string, "EXIT")==0)
334                 {
335                         UART0_send_text(UART_menu_cmd[8]);
336                         UART_menu_position = UART_MENU_FIRSTCHAR;
337                         return(0);
338                 }
339                 // string to short, error
```

```
340                 else if (UART_cmd_counter<10)
341                 {
342                         UART0_send_text(UART_menu_cmd[2]);              // display error
343                         break;
344                 }
345                 // check each char
346                 while(((*pglobal_string > 43) && (*pglobal_string < 58)))
347                 {
348                         UART_cmd_counter--;
349                         pglobal_string++;               // set global pointer on next
                                char
350                 }
351                 // if only one char isn't ok, error
352                 if(UART_cmd_counter != 0)
353                 {
354                         UART0_send_text(UART_menu_cmd[2]);              // display error
355                         break;
356                 }
357
358                 // convert day string to int, if value valid store else exit with error
359                 convert_string[0] = global_string[0];
360                 convert_string[1] = global_string[1];
361                 UART0_tmp = atoi(convert_string);
362                 if ((UART0_tmp >= 0)&&(UART0_tmp < 32))
363                                 RTC_values.day = (unsigned char)UART0_tmp;
364                 else
365                 {
366                         UART0_send_text(UART_menu_cmd[2]);              // display error
367                         break;
368                 }
369
370                 // convert month string to int, if value valid store else exit with error
371                 convert_string[0] = global_string[3];
372                 convert_string[1] = global_string[4];
373                 UART0_tmp = atoi(convert_string);
374                 if ((UART0_tmp >= 1)&&(UART0_tmp < 13))
375                         RTC_values.month = (unsigned char)(UART0_tmp - 1);
376                 else
377                 {
378                         UART0_send_text(UART_menu_cmd[2]);              // display error
379                         break;
380                 }
381
382                 // convert year string to int, if value valid store else exit with error
383                 convert_string[0] = global_string[6];
384                 convert_string[1] = global_string[7];
385                 convert_string[2] = global_string[8];
386                 convert_string[3] = global_string[9];
387                 UART0_tmp = atoi(convert_string);
388                 if ((UART0_tmp >= 0)&&(UART0_tmp < 4000))
389                         RTC_values.year = UART0_tmp;
390                 else
391                 {
392                         UART0_send_text(UART_menu_cmd[2]);              // display error
393                         break;
394                 }
395
396                 UART0_send_text(UART_menu_cmd[8]);                      // print next
                        command, enter date
397                 UART_menu_position = UART_MENU_FIRSTCHAR;
398                 return(0);
                                // return with no error
399
400                 break;
401
402 //-------------------------------------------------------------------------------
403 //-------------------------------------------------------------------------------
```

```
404         case UART_MENU_SENSORADDR:                      // set sensor addresses via UART
405                 UART_cmd_counter = strlen(pglobal_string);
406                 // compare string if "EXIT" send and if the length and information iss OK
407                 // if exit jump back to main menu
408                 if (strcasecmp(pglobal_string, "EXIT")==0)
409                 {
410                         UART0_send_text(UART_menu_cmd[8]);
411                         UART_menu_position = UART_MENU_FIRSTCHAR;
412                         return(0);
413                 }
414
415                 // check each char
416                 while(((*pglobal_string > 47) && (*pglobal_string < 58)))
417                 {
418                         UART_cmd_counter--;
419                         pglobal_string++;               // set global pointer on next
                                char
420                 }
421
422                 // Enter the maximum number of sensors
423                 if(sensor_address_index == 0)
424                 {
425                         // if only one char isn't ok, error
426                         if(UART_cmd_counter != 0)
427                         {
428                                 UART0_send_text(UART_menu_cmd[3]);              //
                                    display error
429                                 break;
430                         }
431
432                         // convert string to integer
433                         UART0_tmp = atoi(global_string);
434
435                         // only 4 options: 6,12,24,40 sensors configured
436                         if ((UART0_tmp == 6) || (UART0_tmp == 12) || (UART0_tmp == 24) ||
                                 (UART0_tmp == 40))
437                         {
438                                 if (UART0_tmp < SENSOR_active)
439                                 {
440                                         for (i=0; i < SENSOR_active; i++)
441                                                 SENSOR_address[i] = 0;
442                                 }
443                                 SENSOR_active = (unsigned char)UART0_tmp;
444
445                         }
446                         else    // invalid value, exit
447                         {
448                                 UART0_send_text(UART_menu_cmd[3]);      // display error
449                                 break;
                                            // exit switch case
450                         }
451
452                         sensor_address_index = 1;
453                         UART0_send_text(UART_menu_cmd[4]);
454                         return(0);
455                 }
456                 // Enter the address of each sensors
457                 else
458                 {
459                         // if only one char isn't ok, error
460                         if(UART_cmd_counter != 0)
461                         {
462                                 UART0_send_text(UART_menu_cmd[4]);              //
                                    display error
463                                 break;
464                         }
465
```

```
466                         UART0_tmp = atoi(global_string);
467                         SENSOR_address[(sensor_address_index-1)] = (unsigned char)
                                UART0_tmp;    // store sensor address
468                         sensor_address_index++;
469
470                         // all sensors configured
471                         if (sensor_address_index > SENSOR_active)
472                         {
473                                 UART0_send_text(UART_menu_cmd[8]);
474                                 UART_menu_position = UART_MENU_FIRSTCHAR;
475                                 sensor_address_index = 0;
476                                 FLASH_Write_Controll_Data();
477                                 for (i=0; i<SENSOR_active; i++)
478                                 {
479                                         SENSOR_last_data[0][i] = SENSOR_address[i];
480                                 }
481
482                                 // set sensors active
483                                 switch(SENSOR_active)
484                                 {
485                                 case 12:
486                                         SENSOR_addr_req[0] = 0xFF;              // 12
                                            sensors are configured
487                                         SENSOR_addr_req[1] = 0x0F;
488                                         break;
489                                 case 24:
490                                         SENSOR_addr_req[0] = 0xFF;              // 24
                                            sensors are configured
491                                         SENSOR_addr_req[1] = 0xFF;
492                                         SENSOR_addr_req[2] = 0xFF;
493                                         break;
494                                 case 40:
495                                         SENSOR_addr_req[0] = 0xFF;              // 24
                                            sensors are configured
496                                         SENSOR_addr_req[1] = 0xFF;
497                                         SENSOR_addr_req[2] = 0xFF;
498                                         SENSOR_addr_req[3] = 0xFF;
499                                         SENSOR_addr_req[4] = 0xFF;
500                                         break;
501                                 default:
502                                         SENSOR_addr_req[0] = 0x3F;              // 6
                                            sensors are configured
503                                         break;
504                                 }
505                         }
506                         return(0);
507                 }
508                 break;
509         };
510
511         return(1);                                              // return with error
512 }
513
514 /*---------------------------------------------------------------------------
515         UART menu function to clear the flash
516 ---------------------------------------------------------------------------*/
517 void UART_menu_flash_clr(void)
518 /*---------------------------------------------------------------------------*/
519 {
520         UART0_send_text(UART_menu_cmd[6]);
521         Erase_Flash_All();
522 }
523
524 /*---------------------------------------------------------------------------
525         UART menu function to send error
526 ---------------------------------------------------------------------------*/
527 void UART_menu_error(void)
```

```
528 /*---------------------------------------------------------------------------*/
529 {
530         UART0_send_text(UART_menu_cmd[7]);
531 }
532
533 /*---------------------------------------------------------------------------
534         UART menu function to show help screen
535 ---------------------------------------------------------------------------*/
536 void UART_menu_show_help(void)
537 /*---------------------------------------------------------------------------*/
538 {
539         UART0_send_text(UART_menu_cmd[0]);
540 }
541
542 /*---------------------------------------------------------------------------
543         UART menu function to show help screen
544 ---------------------------------------------------------------------------*/
545 void UART_menu_setrtc(void)
546 /*---------------------------------------------------------------------------*/
547 {
548         UART0_send_text(UART_menu_cmd[1]);
549         UART_menu_position = UART_MENU_CLOCK;
550 }
551
552 /*---------------------------------------------------------------------------
553         UART menu function to show help screen
554 ---------------------------------------------------------------------------*/
555 void UART_menu_setsensor(void)
556 /*---------------------------------------------------------------------------*/
557 {
558         UART0_send_text(UART_menu_cmd[3]);
559         UART_menu_position = UART_MENU_SENSORADDR;
560 }
561
562 /*---------------------------------------------------------------------------
563         UART menu function to read the whole measurement from flash
564 ---------------------------------------------------------------------------*/
565 void UART_menu_read_data(void)
566 /*---------------------------------------------------------------------------*/
567 {
568         DATA_STRUCT *pData;
569         unsigned int year_temp;
570         unsigned char temp;
571         unsigned char control;
572         // flash temp variables
573         unsigned char  tmp_Ring_Full;
574         unsigned short tmp_Index_Ring;
575         unsigned short tmp_Count_Ring;
576         unsigned short tmp_Index_Last;
577
578         RADIO_disable();        // disable radio modul
579         control = 1;
580
581         tmp_Ring_Full = Ring_Full;
582         tmp_Index_Ring = Index_Ring;
583         tmp_Count_Ring = Count_Ring;
584         tmp_Index_Last = Index_Last;
585
586
587         PRINTF_UART;            // set printf to UART
588
589         while(control)
590         {
591
592                 pData = Read_Ring_Flash();
593                 while(pData != NULL)
594                 {
```

```
595                         // send weekday
596                         UART0_send_text(RTC_days[pData->C4]);
597
598                         // send date
599                         UART0_send(pData->C0/10+48);
600                         UART0_send(pData->C0%10+48);
601                         UART0_send('-');
602                         UART0_send_text(RTC_months[pData->C1]);
603                         UART0_send('-');
604                         // get year integer value
605                         year_temp = (pData->C2 << 8) | pData->C3;
606                         temp = year_temp / 1000;
607                         UART0_send(temp+48);
608                         temp = (year_temp / 100) - ((year_temp/1000) * 10);
609                         UART0_send(temp+48);
610                         temp = (year_temp / 10) - ((year_temp/100) * 10);
611                         UART0_send(temp+48);
612                         temp = year_temp % 10;
613                         UART0_send(temp+48);
614                         UART0_send(' ');
615
616                         // send time
617                         UART0_send(pData->C5/10+48);
618                         UART0_send(pData->C5%10+48);
619                         UART0_send(':');
620                         UART0_send(pData->C6/10+48);
621                         UART0_send(pData->C6%10+48);
622                         UART0_send_text(":00");
623                         //UART0_send(pData->C7/10+48);
624                         //UART0_send(pData->C7%10+48);
625
626                         // send temperature, cell-voltage and supply-voltage values in
                                hex
627                         printf(" %2x %2x %2x %2x %2x %2x %2x %2x\r\n", pData->C8, pData->
                                C9,
628                                         pData->C10, pData->C11, pData->C12, pData->C13,
                                             pData->C14,
629                                         pData->C15);
630
631                         pData = Read_Ring_Flash();
632                 };
633
634                 UART0_send_text("OK\r\n");
635
636                 LCD_send_cmd(LCD_LINE1);
637                 LCD_send_text(" Data received? ");
638                 LCD_send_cmd(LCD_LINE2);
639                 LCD_send_text("YES    NO       ");
640
641                 while(1)
642                 {
643                         if(BATMON_control_reg & B1)     // Button 1 = YES,
644                         {
645                                 control = 0;
646                                 BATMON_control_reg &= ~(B1|B2|B3);
                                        // clear button state
647
648                                 LCD_send_cmd(LCD_LINE1);
649                                 if (Enable_clear_lcd == ON)
650                                         LCD_send_text("                ");
651                                 else
652                                         LCD_send_text(" Radio disabled ");
653
654                                 // set time on second LCD line
655                                 LCD_send_time();
656
657                                 RADIO_init_frame();     // write first frame with actual
```

```
                                    time and 0 as data
658                                 FLASH_Write_Controll_Data();
659
660                                 break;                                  // exit while
                                        loop
661                         }
662                         if(BATMON_control_reg & B2)     // Button 2 = NO,
663                         {
664                                 Ring_Full  = tmp_Ring_Full;
665                                 Index_Ring = tmp_Index_Ring;
666                                 Count_Ring = tmp_Count_Ring;
667                                 Index_Last = tmp_Index_Last;
668
669                                 BATMON_control_reg &= ~(B1|B2|B3);
                                        // clear button state
670                                 break;                                  // exit while
                                        loop
671                         }
672                 }
673         }
674         /*      begin with new frame    */
675         //RADIO_enable();               // enable radio modul
676 }
677
678 /*-------------------------------------------------------------------------------
679         UART menu function to start the measurement
680 -------------------------------------------------------------------------------*/
681 void UART_menu_start(void)
682 /*-------------------------------------------------------------------------------*/
683 {
684         LCD_send_cmd(LCD_LINE1);
685         LCD_send_text(" Radio enabled  ");
686         UART0_send_text("Radio enabled \r\n");
687         Enable_clear_lcd = ON;
688         RADIO_enable();
689 }
690
691 /*-------------------------------------------------------------------------------
692         UART menu function to stop the measurement
693 -------------------------------------------------------------------------------*/
694 void UART_menu_stop(void)
695 /*-------------------------------------------------------------------------------*/
696 {
697         LCD_send_cmd(LCD_LINE1);
698         LCD_send_text(" Radio disabled ");
699         UART0_send_text("Radio disabled\r\n");
700         Enable_clear_lcd = OFF;
701         RADIO_disable();
702 }
703
704 /*-------------------------------------------------------------------------------
705         UART menu function to read controll data from flash
706 -------------------------------------------------------------------------------*/
707 void UART_menu_read_controll(void)
708 /*-------------------------------------------------------------------------------*/
709 {
710         UART0_send_text("Read data\r\n");
711         FLASH_Read_Controll_Data();
712 }
713
714 /*-------------------------------------------------------------------------------
715         UART menu function to write controll data to flash
716 -------------------------------------------------------------------------------*/
717 void UART_menu_write_controll(void)
718 /*-------------------------------------------------------------------------------*/
719 {
720         UART0_send_text("Write data\r\n");
```

```
721         FLASH_Write_Controll_Data();
722 }
```

Listing B.15: Header der Menüfunktionen (UART_menu.h)

```
1  /*-------------------------------------------------------------------------------
2          Project:                        MSP430 MSP_functions header
3          Used components:                MSP430-169STK
4          Date:                           02/14/2008
5          Last update:                    04/04/2008
6          Author:                         Stephan Plaschke
7  -------------------------------------------------------------------------------*/
8
9  #ifndef UART_MENUE_H_
10 #define UART_MENUE_H_
11
12
13 /*-------------------------------------------------------------------------------
14         Structures
15 -------------------------------------------------------------------------------*/
16 typedef struct{                         // structure for UART menu
17         unsigned char *cmd;
18         void (*fct)(void);              // pointer on function with no return- and no
               parameter value
19 } UART_MEN;
20
21 /*-------------------------------------------------------------------------------
22         Global variables
23 -------------------------------------------------------------------------------*/
24 extern unsigned char BATMON_control_reg;
25 extern unsigned char Enable_clear_lcd;
26
27 extern unsigned char SENSOR_active;
28 extern unsigned char SENSOR_address[40];        // array with sensor address
29 extern unsigned char SENSOR_last_data[8][40];
30
31 extern unsigned char global_string[MAX_STRING_LENGTH];
32 extern unsigned char RADIO_rx_state;                                    // data receive
       state
33 extern unsigned char UART_menu_position;
34
35 extern RTC_MSP430                       RTC_values;
       // global RTC values
36 extern RADIO_FRAME_STRUCT       RADIO_frame;                            // received data
       frame
37 extern DATA_STRUCT *test[128];
38
39
40 /*-------------------------------------------------------------------------------
41         Defines
42 -------------------------------------------------------------------------------*/
43 enum uart_menu_positions {      UART_MENU_FIRSTCHAR, UART_MENU_CLOCK, UART_MENU_DATE,
44                                                         UART_MENU_SENSORADDR};
45
46 /*-------------------------------------------------------------------------------
47         Prototypes
48 -------------------------------------------------------------------------------*/
49
50 void UART_menu_flash_clr(void);
51 void UART_menu_error(void);
52 void UART_menu_show_help(void);
53 void UART_menu_read_data(void);
54 void UART_menu_setrtc(void);
55 void UART_menu_setsensor(void);
56 void UART_menu_start(void);
57 void UART_menu_stop(void);
```

```
58 void UART_menu_read_controll(void);
59 void UART_menu_write_controll(void);
60
61
62
63 #endif /*UART_MENUE_H_*/
```

B.2. *MATLAB* Quellcode

B.2.1. Temperaturkalibrierung

Listing B.16: Berechnung der Differenzfunktion und Speichern der errechneten Werte

```
1  %--------------------------------------------------------------------------
2  % Project:      Diploma
3  % Date:         02/06/2008
4  % Last Update:  12/06/2008
5  % Author:       Stephan Plaschke
6  % Discreption:  Read and plot cell-sensor data, calculate difference
7  %               function for temperature and save polynome values
8  %               in "sensorsystem.txt"
9  %--------------------------------------------------------------------------
10
11 function show_cell_status(figure1)
12
13 % clear all variables and close all windows
14 clear all;
15 close all;
16
17 % arangement of the sensors on the battery, write down address in specific
18 % order
19 valid_sensor_address = [2,3,4,5,6,7,8,9,10,11,12,14];
20 max_sensor = 12;
21 function_order = 2;     % quadratic function
22 % real temperature from test
23 real_temp = [10 15 20 25 30 35 40 45 50 55 60];
24
25 disp('--------------------------------------------------------------------------')
26 disp('Read cell-sensor-data and plot data from each sensor on a own figure')
27 disp('Use only *.txt files as input')
28 disp('As input use only filename with no extension')
29 disp('Author:   Stephan Plaschke')
30 disp('Date:     13.05.2008')
31 disp('--------------------------------------------------------------------------')
32 disp('  ');
33
34 [filename, pathname, filterindex] = uigetfile( ...
35 {'*.txt','test files (*.txt)'; ...
36 '*.dat','data files (*.dat)'; ...
37 '*.*',  'All Files (*.*)'}, ...
38 'Pick a test file or data file from a battery experiment');
39 if isequal(filename,0) | isequal(pathname,0)
40     disp('User pressed cancel in File selection dialog')
41     return
42 else
43     disp(['User has selected the file:']);
44     disp(fullfile(pathname, filename));
45     filename_compl = fullfile(pathname, filename);
46 end
47
48 % read data from file
49 data=textread(filename_compl,'%s');
```

```
50
51 % create sensor address array, later you will find here each used sensor
52 % address
53 sensor_addr = zeros(1,40);
54 index_data = ones(1,40);                % create index array
55 sensor_index = 1;                       % number of sensors
56
57 % calculate array size, divided by eleven because of length of information
58 data_length = (length(data)/11);
59 % reshape array from [x][1] to [11][x]
60 data = reshape(data, length(data)/data_length, data_length);
61
62 % convert char to dec
63 for i=1:8
64     temp(i,:) = hex2dec(data((i+3),:));
65 end
66
67 B(1,:) = temp(1,:);                     % sensor address
68 % calculate battery-, supply voltage and temperature values
69 for i=2:4
70    B(i,:)= 256 * temp(i+((i-2)*1),:) + temp(i+((i-1)*1),:);
71 end
72
73 % sort data
74 %--------------------------------------------------------------------------
75 for i=1:data_length
76     j = 1;                            % counter
77     % sort sensor data from valid addresses
78     while B(1,i) ~= valid_sensor_address(j)
79         j = j + 1;
80         % address not found: exit with error
81         if j == max_sensor+1
82             error('Invalid sensor address!');
83         end
84     end
85
86     % if new valid sensor address found increment index and store address
87     if sensor_addr(j) == 0
88         sensor_addr(j) = B(1,i);
89         %sensor_index = sensor_index + 1;
90     end
91
92     % write timestamp into cellarray, for cell use {}!!
93     date_sensor(j, index_data(j)) = {sprintf('%s %s',char(data(2,i)),....
94         char(data(3,i)))};
95
96     % write temperaturevalue into array
97     data_sorted((j*3-2), index_data(j))= B(2,i);
98     % write batteryvoltage value into array
99     data_sorted((j*3-1), index_data(j))= B(3,i);
100    % write supplyvoltage value into array
101    data_sorted((j*3),   index_data(j))= B(4,i);
102    % increment index value
103    index_data(j) = index_data(j) + 1;
104
105 end
106 %--------------------------------------------------------------------------
107 % create ymin and ymax values
108 ymin_temp = 100;
109 ymax_temp = 0;
110 ymin_supply = 5;
111 ymax_supply = 0;
112 ymin_cell = 5;
113 ymax_cell = 0;
114
115 % open file to store polynome values
116 fid = fopen('sensorsystem.txt', 'a+');
```

```
117 fprintf(fid, '%i\r', max_sensor);
118 for i=1:max_sensor
119     fprintf(fid, '%i\r', valid_sensor_address(i));
120 end
121 fprintf(fid, '%i\r', function_order+1);
122 
123 %--------------------------------------------------------------------------
124 % display result for each sensor
125 for i = 1:max_sensor
126 
127     %create new figure window for each sensor
128     figurename = sprintf('Sensor: %d, Sensoraddresse: %d', i, char(sensor_addr(1,i)));
129     screensize = get(0,'ScreenSize');
130     figure('Name',figurename, 'NumberTitle','off', 'Position',....
131          [screensize]);
132 
133     % calculate X-axis length
134     x_data = find(data_sorted((i*3),:));
135     x_data = length(x_data);
136 
137     % error if not enough data
138     if (x_data == 1)
139         close all;
140         clear all;
141         error('Only one value received, at least two values are needed');
142     end
143 
144     % get X-Axis values, datenum() is needed for timestamp
145     x(1,1:x_data) = datenum(date_sensor(i,1:x_data),....
146         'dd-mmm-yyyy HH:MM:SS');
147 
148     % scale temperature graph
149     XTick = linspace(x(1), x(end), 5);
150     XTickLabel = datestr(XTick, 'dd-mmm-yyyy HH:MM:SS');
151 
152     % get Y-axis values, divide by 8 for temperature
153     y(1,1:x_data) = data_sorted(i*3-2,1:x_data)/8;
154 
155     % get max and min y-achse value
156     if (min(y) < ymin_temp)
157         ymin_temp = min(y);
158     end
159     if (max(y) > ymax_temp)
160         ymax_temp = max(y);
161     end
162 
163     % calculate difference function
164     x_trendline(1,:) = y(1,:);
165     y_trendline(1,:) = real_temp(1,:) - y(1,:);
166 
167     % calculate polynomes
168     p = polyfit(x_trendline, y_trendline, function_order);
169 
170     % store polynome values in polynome.txt
171     for j=1:(function_order+1)
172         fprintf(fid, '%i\r', p(j));
173     end
174 
175     y_trendline = polyval(p,x_trendline);
176     y_trendline(1,:) = y_trendline(1,:) + y(1,:);
177 
178     % plot temperature
179     % create 3 plotwindows, select first one
180     subplot(3,1,1);
181     % plot temperature graph
182     plot(x,real_temp,'green', x,y,'red', x,y_trendline,'blue');
183     set(gca,'XTick',XTick, 'XTickLabel',char(XTickLabel));
```

```
184     grid on;
185
186     % create title
187     figure_title = sprintf(....
188         'FILE: "%s", Start: %s, End: %s, Sensoraddress:%d',filename,....
189         char(date_sensor(i,1)), char(date_sensor(i,x_data)),....
190         char(sensor_addr(1,i)));
191     % set title
192     title(figure_title);
193     % set Y-label
194     ylabel('Temperature/°C');
195
196
197     %--------------------------------------------------------------------
198     % get cel voltage y-achse values
199     y(1,1:x_data) = data_sorted(i*3-1,1:x_data)/4096;
200
201     % get max and min y-achse value
202     if (min(y) < ymin_cell)
203         ymin_cell = min(y);
204     end
205     if (max(y) > ymax_cell)
206         ymax_cell = max(y);
207     end
208
209     % plot cell voltage
210     subplot(3,1,2);
211     plot(x,y);
212     grid on;
213     set(gca,'XTick',XTick, 'XTickLabel',char(XTickLabel));
214     ylabel('Cell voltage/V');
215
216     %--------------------------------------------------------------------
217     % get supply voltage y-achse values
218     y(1,1:x_data) = data_sorted(i*3,1:x_data)/4096;
219
220     % get max and min y-achse value
221     if (min(y) < ymin_supply)
222         ymin_supply = min(y);
223     end
224     if (max(y) > ymax_supply)
225         ymax_supply = max(y);
226     end
227
228     % plot supply voltag
229     subplot(3,1,3);
230     plot(x,y);
231     grid on;
232     set(gca,'XTick',XTick, 'XTickLabel',char(XTickLabel));
233     ylabel('Supply voltage/V');
234
235     % clear x and y each cycle
236     clear x;
237     clear y;
238 end
239
240 fclose(fid);    % close file
241
242 %------------------------------------------------------------------------
243 % if temperature ymin can't devided by 5 subtract rest
244 if mod(ymin_temp,5)
245     ymin_temp = ymin_temp - mod(ymin_temp,5);
246 end;
247 % if temperature ymax can't devided by 5 add rest
248 if mod(ymax_temp,5)
249     ymax_temp = ymax_temp + (5-mod(ymax_temp,5));
250 end;
```

```
%-------------------------------------------------------------------------
% if cell voltage ymin can't devided by 0.2 subtract rest
if mod(ymin_cell,0.2)
    ymin_cell = ymin_cell - mod(ymin_cell,0.2);
end;
% if cell voltage ymax can't devided by 0.2 add rest
if mod(ymax_cell,0.2)
    ymax_cell = ymax_cell + (0.2-mod(ymax_cell,0.2));
end;
%-------------------------------------------------------------------------
% if supply voltage ymin can't devided by 0.2 subtract rest
if mod(ymin_supply,0.2)
    ymin_supply = ymin_supply - mod(ymin_supply,0.2);
end;
% if supply voltage ymax can't devided by 0.2 add rest
if mod(ymax_supply,0.2)
    ymax_supply = ymax_supply + (0.2-mod(ymax_supply,0.2));
end;
%-------------------------------------------------------------------------
% configure y-achse of each sensor graph with the ymax and ymin values of
% each sensor data
for i = 1:max_sensor
    % set figure(i) temperature y-achse
    figure(i);
    subplot(3,1,1);
    % get 5 values between ymax and ymin
    YTick = linspace(ymin_temp, ymax_temp, 5);
    clear YTickLabel;
    for J = 1:5,
        YTickLabel(J,:) = {YTick(J)};   % set Label
    end
    % set y max and min values
    YLim = [ymin_temp ymax_temp];
    set(gca,'YLim',YLim,'YTick',YTick, 'YTickLabel',YTickLabel);
    legend('real','measured','calibrated','Location','NorthEast');

    %-------------------------------------------------------------------------
    % set figure(i) cell voltage y-achse
    subplot(3,1,2);
    % get 5 values between ymax and ymin
    YTick = linspace(ymin_cell, ymax_cell, 5);
    clear YTickLabel;
    for J = 1:5,
        YTickLabel(J,:) = {YTick(J)};   % set Label
    end
    % set y max and min values
    YLim = [ymin_cell ymax_cell];
    set(gca,'YLim',YLim,'YTick',YTick, 'YTickLabel',YTickLabel);

    %-------------------------------------------------------------------------
    % set figure(i) supply voltage y-achse
    subplot(3,1,3);
    % get 5 values between ymax and ymin
    YTick = linspace(ymin_supply, ymax_supply, 5);
    clear YTickLabel;
    for J = 1:5,
        YTickLabel(J,:) = {YTick(J)};   % set Label
    end
    % set y max and min values
    YLim = [ymin_supply ymax_supply];
    set(gca,'YLim',YLim,'YTick',YTick, 'YTickLabel',YTickLabel);
end
```

B.2.2. Auswertung der Messergebnisse

Listing B.17: Auswahlbildschirm

```
%-------------------------------------------------------------------------
% Project:      Diploma - display sensor data
% Date:         14/06/2008
% Last Update:  14/06/2008
% Author:       Stephan Plaschke
% Discreption:  Splash screen - chose your information source
%-------------------------------------------------------------------------

close all;
clear figure_title;
clear global date_sensor;
clear global sensor_addr;             % array with valid sensor addresses
clear global sensor_count;            % number of sensors
clear global data_sorted;             % temp, battery- and supply voltage values
clear global polynom;                 % offset function polynomes
clear global poly_order;              % offset function order
clear global radio_buttons_onoff;     % status variable for radiobuttons

% normal fonts
set(0,'defaultaxesfontsize',12);
set(0,'defaulttextfontsize',12);

screensize = get(0,'ScreenSize');
figure(4);
set(gcf,'Name','Wireless Battery Management Analysis Software', 'MenuBar','none');
set(gca,'Visible','off');
text(0.02,0.95, '=========================================' );
text(0.08,0.9,'Wireless Battery Managemant Analysis Software' );
text(0.02,0.85, '=========================================' );

% small fonts
set(0,'defaultaxesfontsize',10);
set(0,'defaulttextfontsize',10);
text(0, 0.7,'Two data sources are needed. The configuration file includes the information
    ');
text(0, 0.65,'about the sensor system. The name must be "sensorsystem.txt". The data file
    ');
text(0, 0.6,'includes the meassured data from each sensor in the system. Instead of an');
text(0, 0.55,'data file it is possible to transmitt the data directly from the
    controlling unit via');
text(0, 0.5,'RS232. If you are using the serial port be carefull. It is tested but did
    not work');
text(0, 0.45,'hundred percently.');
text(0, 0.3,'Writen and developed from Stephan Plaschke in May/June 2008.');

% normal fonts
set(0,'defaultaxesfontsize',12);
set(0,'defaulttextfontsize',12);
text(0.35,0.12,'Read data from:');


uicontrol(4,'style','pushbutton','Value',1,'string','Comport',...
          'units','normalized','position',[0.55 0.1 0.15 0.075],'callback','read_com();'
              );

uicontrol(4,'style','pushbutton','Value',1,'string','File',...
          'units','normalized','position',[0.30 0.1 0.15 0.075],'callback','read_file();
              ');
```

Listing B.18: Daten über serielle Schnittstelle einlesen

```
%--------------------------------------------------------------------------
% Project:      Diploma - display sensor data
% Date:         14/06/2008
% Last Update:  14/06/2008
% Author:       Stephan Plaschke
% Discreption:  Read data from comport
%--------------------------------------------------------------------------

function read_com
%--------------------------------------------------------------------------
% read data from control unit
close all;

com_port = sprintf('COM%s', input('Enter COM-PORT:','s'));
% open COM-PORT with 115200 baud and return as terminator
serial_port = serial(com_port,'BaudRate',115200,'Terminator','CR');
fopen(serial_port);

% send data request
fprintf(serial_port, 'mem');

% get data from serial port, if error close serial port and exit
% start error handler
try
i = 1;
% get data from serial port
temp = {fscanf(serial_port)};
temp_char = char(temp);
while (1)
    if (length(temp_char) < 3)
        error('Invalid row-data');
    end

    for j=1:length(temp_char)-1
        if (temp_char(1,j:j+1)=='OK' )
            ok_detected = 1;
            break;
        else
            ok_detected = 0;
        end
    end
    % when OK occurs exit while loop
    if (ok_detected)
        break;
    end

    data(i,:) = temp(1,:);
    i = i + 1;
    % get data from serial port
    temp = {fscanf(serial_port)};
    temp_char = char(temp);
end
% if error occured while receiving data, close port and exit
catch me
% close serial port
fclose(serial_port);
delete(serial_port);
clear serial_port;
rethrow(me);
end

% close serial port
fclose(serial_port);
delete(serial_port);
clear serial_port;
```

```
66
67 % convert cell to char-array
68 data_char = char(data);
69
70 % calculate array size
71 data_x = length(data_char);
72 data_y = numel(data_char)/data_x;
73
74 % error if not enough data
75 if (data_y < 3)
76 close all;
77 clear all;
78 error('Not enough valid row-data received');
79 end
80
81 % open or create file and write data in it
82 filename = sprintf('%s.txt',date);
83 fid = fopen(filename, 'a+');
84 fprintf(fid, '\r%s\r',datestr(now));
85
86 clear data;
87 clear temp;
88
89 % copy data from horizontal to vertical array
90 for i = 1:data_y-2
91 data((i)+((i-1)*10),:)   = {data_char(i+2,1:4)};
92 data((i+1)+((i-1)*10),:) = {data_char(i+2,6:16)};
93 data((i+2)+((i-1)*10),:) = {data_char(i+2,18:25)};
94 data((i+3)+((i-1)*10),:) = {data_char(i+2,27:28)};
95 data((i+4)+((i-1)*10),:) = {data_char(i+2,30:31)};
96 data((i+5)+((i-1)*10),:) = {data_char(i+2,33:34)};
97 data((i+6)+((i-1)*10),:) = {data_char(i+2,36:37)};
98 data((i+7)+((i-1)*10),:) = {data_char(i+2,39:40)};
99 data((i+8)+((i-1)*10),:) = {data_char(i+2,42:43)};
100 data((i+9)+((i-1)*10),:) = {data_char(i+2,45:46)};
101 data((i+10)+((i-1)*10),:)= {data_char(i+2,48:49)};
102
103 % write data to file
104 fprintf(fid, '%s',data_char(i,:));
105 end
106
107 fclose(fid);    % close file
108
109 main(data, filename);
```

Listing B.19: Daten aus einem File einlesen

```
1  %-------------------------------------------------------------------------
2  % Project:      Diploma - display sensor data
3  % Date:         14/06/2008
4  % Last Update:  14/06/2008
5  % Author:       Stephan Plaschke
6  % Discreption:  Read data from file
7  %-------------------------------------------------------------------------
8
9  function read_file
10 %-------------------------------------------------------------------------
11 % read data from *.txt file
12
13 [filename, pathname, filterindex] = uigetfile( ...
14 {'*.txt','test files (*.txt)'; ...
15 '*.dat','data files (*.dat)'; ...
16 '*.*',  'All Files (*.*)'}, ...
17 'Pick a datafile from a battery meassurement');
18 if isequal(filename,0) | isequal(pathname,0)
19     disp('Canceled by user in fileselect box');
```

```
    return
else
    filename_compl = fullfile(pathname, filename);
    disp(' ');
    disp(['User has selected the file:']);
    disp(filename_compl);
    disp(' ');
end

% read data from file
data=textread(filename_compl,'%s');

main(data, filename);
```

Listing B.20: Hauptprgramm der Analysesoftware

```
%-------------------------------------------------------------------------
% Project:      Diploma - display sensor data
% Date:         11/12/2007
% Last Update:  14/06/2008
% Author:       Stephan Plaschke
% Discreption:  Read cell-sensor data
%-------------------------------------------------------------------------

function main(data, filename)

% clear all variables and close all windows
close all;

clear global figure_title;

% switch all warnings off
warning('off');

% small fonts
set(0,'defaultaxesfontsize',8);
set(0,'defaulttextfontsize',8);

% PDF print size, print without radiobutton in cm
myfigureprintsize = [-1,0,24,14];
mypapersize=[20.984 15];
% 15 is 14-0 in config above myfigureprintsize change both parameters together

global ssw;                     % Array for switching sensor graphs on/off
ssw=ones(3,40);

%global figure_title;    % by cell variables use global after use
global sensor_addr;             % array with valid sensor addresses
global sensor_count;            % number of sensors
global data_sorted;             % temp, battery- and supply voltage values
global polynom;                 % offset function polynomes
global poly_order;              % offset function order
global radio_buttons_onoff;     % status variable for radiobuttons
radio_buttons_onoff(1,1:3) = ones;

screensize = get(0,'ScreenSize');   % get screensize

% create sensor address array, later you will find here each used sensor
% address
sensor_addr = zeros(1,40);
index_data = ones(1,40);                % create index array

% calculate array size, divided by eleven because of length of information
data_length = (length(data)/11);
% reshape array from [x][1] to [11][x]
data = reshape(data, length(data)/data_length, data_length);
```

```
51
52 % convert char to dec
53 for i = 1:8
54     temp(i,:) = hex2dec(data((i+3),:));
55 end
56
57 % calculate 16bit values from two 8bit values
58 B(1,:) = temp(1,:);
59 for i = 2:4
60     B(i,:)= 256 * temp((i+(i-2)*1),:) + temp((i+(i-1)*1),:);
61 end
62
63 % read polynomevalues for offset function
64 fid = fopen('sensorsystem.txt','r');
65 sensor_system = fscanf(fid, '%g',[1 inf]);
66 fclose(fid);
67
68 sensor_count = sensor_system(1);                    % number of sensors
69 for i=1:sensor_count
70     valid_sensor_address(i) = sensor_system(i+1);   % address of each sensor
71 end
72
73 poly_order = sensor_system(2+sensor_count);         % offset function order
74 for i=1:(sensor_count*poly_order)
75     polynom(i) = sensor_system(i+2+sensor_count);   % offset function polynomes
76 end
77
78 % sort data
79 %-------------------------------------------------------------------------
80 for i=1:data_length
81     j = 1;                          % counter
82     while B(1,i) ~= valid_sensor_address(j)
83         j = j + 1;
84         if j == sensor_count+1
85             error('Invalid sensor address!');
86         end
87     end
88
89     if sensor_addr(j) == 0
90         sensor_addr(j) = B(1,i);
91     end
92
93     % write timestamp into cell array, for cell use {}!!
94     date_sensor(j, index_data(j)) = {sprintf('%s %s',char(data(2,i)),....
95         char(data(3,i)))};
96
97     % write temperaturevalue into array
98     data_sorted((j*3-2), index_data(j))= B(2,i);
99
100    if sensor_addr(j) == 11
101        % write batteryvoltage value into array
102        data_sorted((j*3-1), index_data(j))= B(3,i) - 410;
103        % write supplyvoltage value into array
104        data_sorted((j*3),   index_data(j))= B(4,i) - 410;
105    else
106        % write batteryvoltage value into array
107        data_sorted((j*3-1), index_data(j))= B(3,i);
108        % write supplyvoltage value into array
109        data_sorted((j*3),   index_data(j))= B(4,i);
110    end
111
112    % increment index value
113    index_data(j) = index_data(j) + 1;
114
115 end
116
117 global date_sensor;     % by cell use global after first use
```

```
118 %------------------------------------------------------------------------
119
120 % find the first maximum on x-achse, just in case
121 [countmax_x,countmax_y] = max(index_data);
122
123 % figures to display summarized temperatures, cell voltages and supply voltages
124 figuresize1=[screensize(3)*0.05+0, screensize(4)*0.05+20,screensize(3)*0.8+0,...
125     screensize(4)*0.8+20];
126 figuresize2=[screensize(3)*0.05+25, screensize(4)*0.05+10,screensize(3)*0.8+25,...
127     screensize(4)*0.8+10];
128 figuresize3=[screensize(3)*0.05+50, screensize(4)*0.05+0,screensize(3)*0.8+50,...
129     screensize(4)*0.8+0];
130
131 figure_caption =' from File: "%s", Start: %s, End: %s, \n Sensoraddresses: %s';
132
133 figure(1);
134 %set(gcf,'Name','Temperatures', 'Position',[screensize]);
135 set(gcf,'Name','Temperatures', 'Position',[figuresize1]);
136 set(gcf,'MenuBar','figure','units','normalized','Toolbar','figure');
137 set(gcf, 'PaperPosition', myfigureprintsize,'PaperSize',mypapersize);
138 figure_title(1) = {sprintf(strcat('Temperatures', figure_caption),filename,....
139        char(date_sensor(countmax_y,1)), char(date_sensor(countmax_y,end)), ...
140        strcat(num2str(sensor_addr(1,1:sensor_count))))
141 }; % callback as string defined
142
143 figure(2);
144 %set(gcf,'Name','Cell Voltages', 'Position',[screensize]);
145 set(gcf,'Name','Cell Voltages', 'Position',[figuresize2]);
146 set(gcf,'MenuBar','figure','units','normalized','Toolbar','figure');
147 set(gcf, 'PaperPosition', myfigureprintsize,'PaperSize',mypapersize);
148 figure_title(2) = {sprintf(strcat('Cell Voltages', figure_caption),filename,....
149        char(date_sensor(countmax_y,1)), char(date_sensor(countmax_y,end)),...
150        strcat(num2str(sensor_addr(1,1:sensor_count))));
151 }; % callback as string defined
152
153 figure(3);
154 %set(gcf,'Name','Supply Voltages', 'Position',[screensize]);
155 set(gcf,'Name','Supply Voltages', 'Position',[figuresize3]);
156 set(gcf,'MenuBar','figure','units','normalized','Toolbar','figure');
157 set(gcf, 'PaperPosition', myfigureprintsize,'PaperSize',mypapersize);
158 figure_title(3) = {sprintf(strcat('Supply Voltages', figure_caption),filename,....
159        char(date_sensor(countmax_y,1)), char(date_sensor(countmax_y,end)), ...
160        strcat(num2str(sensor_addr(1,1:sensor_count))));
161 }; % callback as string defined
162
163 global figure_title;
164
165 % ---- the matlab user interface functions with radiobutton (windows style)
166 for i=1:sensor_count
167    % radionbuttons on temperature window 1
168    %set(gca,'position',[.05 .1 .8 .8])
169    uicontrol(1,'style','radiobutton','Value',1,'string',strcat('Sensor ',...
170                ': ',num2str(sensor_addr(1,i))),...
171                'units','normalized','position',[.92 .9-i*.05 .07 .03],...
172                'callback',strcat('global ssw; ssw(1,',num2str(i),')=not(ssw(1,',...
173                num2str(i),')); plot_graph(1)'));
174    % radionbuttons on cell_voltage window 2
175    %set(gca,'position',[.05 .1 .8 .8])
176    uicontrol(2,'style','radiobutton','Value',1,'string',strcat('Sensor ',...
177                ': ',num2str(sensor_addr(1,i))),...
178                 'units','normalized','position',[.92 .9-i*.05 .07 .03],...
179                'callback',strcat('global ssw;  ssw(2,',num2str(i),')=not(ssw(2,',...
180                num2str(i),')); plot_graph(2)'));
181     % radionbuttons on supply_voltage window 3
182    %set(gca,'position',[.05 .1 .8 .8])
183    uicontrol(3,'style','radiobutton','Value',1,'string',strcat('Sensor ',...
184                ': ',num2str(sensor_addr(1,i))),...
```

```
185                 'units','normalized','position',[.92 .9-i*.05 .07 .03],...
186                 'callback',strcat('global ssw;  ssw(3,',num2str(i),')=not(ssw(3,',...
187                 num2str(i),')); plot_graph(3)'));
188 end
189
190 % radio buttons to switch all graphs on/off
191 uicontrol(1,'style','radiobutton','Value',1,'string','All on/off',...
192             'units','normalized','position',[.92 .2 .07 .03],...
193             'callback','allgraph_onoff(1);');
194 uicontrol(2,'style','radiobutton','Value',1,'string','All on/off',...
195             'units','normalized','position',[.92 .2 .07 .03],...
196             'callback','allgraph_onoff(2);');
197 uicontrol(3,'style','radiobutton','Value',1,'string','All on/off',...
198             'units','normalized','position',[.92 .2 .07 .03],...
199             'callback','allgraph_onoff(3);');
200
201 % plot graphs
202 plot_graph(1);  % temperature;
203 plot_graph(2);  % cell_voltage;
204 plot_graph(3);  % supply_voltage;
205
206 % switch all warnings on
207 warning('on');
```

Listing B.21: Graphen darstellen

```
1  %--------------------------------------------------------------------------
2  % Project:      Diploma - display sensor data
3  % Date:         13/06/2008
4  % Last Update:  13/06/2008
5  % Author:       Stephan Plaschke
6  % Discreption:  Plot sensor graphs
7  %--------------------------------------------------------------------------
8
9  function plot_graph(selector)
10
11 global figure_title;    % different figure title inside the window
12 global date_sensor;     % array with date values
13 global sensor_addr;     % array with valid sensor addresses
14 global sensor_count;    % number of sensors
15 global data_sorted;     % temp, battery- and supply voltage values
16 global ssw;             % Array for switching sensor graphs on/off
17 global polynom;         % offset function polynomes
18 global poly_order;      % offset function order
19
20 yaxislabel(1)={'Temperature / °C'};
21 yaxislabel(2)={'Cell Voltage / V'};
22 yaxislabel(3)={'Supply Voltage / V'};
23 y_min = 100;
24 y_max = 0;
25 offloadvoltage_start = 0;
26 offloadvoltage_end = 0;
27
28 % colormap for 40 different colours
29 cmap =[
30       1.0000    1.0000    1.0000; % white not  visible
31       1.0000         0         0;
32       0         1.0000         0;
33       0              0    1.0000;
34            0    1.0000    1.0000;
35       1.0000         0    1.0000;
36       0.7500         0         0;
37       0         0.7500         0;
38       0              0    0.7500;
39       0.7500    0.7500         0;
40       0         0.7500    0.7500;
```

```
41        0.7500         0    0.7500;
42        0.5000         0         0;
43        0         0.5000         0;
44        0              0    0.5000;
45        0.5000    0.5000         0;
46        0         0.5000    0.5000;
47        0.5000         0    0.5000;
48        1.0000    0.7500         0;
49        0         1.0000    0.7500;
50        0.7500         0    1.0000;
51        1.0000    1.0000    0.7500;
52        0.7500    1.0000    1.0000;
53        1.0000    0.7500    1.0000;
54        0.7500    1.0000         0;
55        0         0.7500    1.0000;
56        1.0000         0    0.7500;
57        0.7500    0.7500    1.0000;
58        1.0000    0.7500    0.7500;
59        0.7500    1.0000    0.7500;
60        0.5000    0.7500         0;
61        0         0.5000    0.7500;
62        0.7500         0    0.5000;
63        0.5000    0.5000    0.7500;
64        0.7500    0.5000    0.5000;
65        0.5000    0.7500    0.5000;
66        1.0000    0.5000         0;
67        0         1.0000    0.5000;
68        0.5000         0    1.0000;
69        1.0000    1.0000    0.5000;
70        0.5000    1.0000    1.0000;
71        ];
72
73  % Symbols at plot lines
74  MATLAB_markers= ['<' 'o' 'v' '+' '*' 's' 'd' 'x' '^' '.' '>' 'p' 'h'] ;
75
76  % set markers for plot, MATLAB only has 13 different markers
77  for i=1:sensor_count
78      j = mod(i,13);
79      markers(i) = MATLAB_markers(j);
80  end
81
82  figure(selector);
83  hold off;
84
85  for i = 1:sensor_count
86      clear x;
87      clear y;
88      x_data = find(data_sorted((i*3-3+selector),:));
89      x_len = length(x_data);
90      % get X-Axis values, datenum() is needed for timestamp
91      x(1:x_len) = datenum(date_sensor(i,1:x_len),'dd-mmm-yyyy HH:MM:SS')';
92      % get Y-axis values, divide by 8 for temperature else by 4096 for voltages
93      if (selector==1)
94          % get temperature values
95          y(1:x_len) = data_sorted(i*3-3+selector,1:x_len)/8;
96          % copy polynome values
97          for j=1:poly_order
98              p(j) = polynom(i+j-1+(i-1)*2);
99          end
100         % calculate difference between real and measured value
101         y_trendline = polyval(p,y);
102         % calculate real graph
103         y(1,:) = y_trendline(1,:) + y(1,:);
104     else
105         % get cellvoltage and supplyvoltage values
106         y(1:x_len) = data_sorted(i*3-3+selector,1:x_len)/4096;
107     end
```

```
108
109     %calculate off-load-voltage
110     if (selector == 2)
111         offloadvoltage_start = offloadvoltage_start + y(1,1);
112         offloadvoltage_end = offloadvoltage_end + y(1,end);
113     end
114
115     % get max and min y-axis value
116     if (min(y) < y_min)
117         y_min = min(y);
118     end
119     if (max(y) > y_max)
120         y_max = max(y);
121     end
122
123     % show graph if radiobutton switched on, delete if switched off
124     if (ssw(selector,i)==0)
125         % invisible plot, only to have an handle for the legend function
126         h=plot(0,0);  hold on;
127         set(h, 'Color', cmap(1,:), 'Marker',markers(i));
128     else
129         h=plot(x,y);  hold on ; zoom on;
130         set(h, 'Color', cmap(i+1,:), 'Marker',markers(i));
131     end
132 end % for
133
134 % show off-load-voltage
135 if (selector == 2)
136     offloadvoltage_start
137     offloadvoltage_end
138 end
139
140 % organize x-axis (date)
141 XTick = linspace(min(x), max(x), 5);
142 XTickLabel = datestr(XTick, 'dd-mmm-yyyy HH:MM:SS');
143 set(gca,'XTick',XTick, 'XTickLabel',XTickLabel);
144
145 % organize y-axis
146 % round grid it to approbiate value
147 if (selector==1)
148     round_value=2;    % Temperature to steps of 2
149 else
150     round_value=0.1;  % Voltages steps of 0.1 V = 100 m V
151 end
152
153 % round max and min values for y-achse
154 if mod(y_min,round_value)
155     y_min = y_min - mod(y_min,round_value);
156 end;
157 if mod(y_max,round_value)
158     y_max = y_max + (round_value-mod(y_max,round_value));
159 end;
160
161 % get 9 values between ymax and ymin
162 YTick = linspace(y_min,y_max, 9);
163 clear YTickLabel;
164 for J = 1:9,
165         YTickLabel(J,:) = {YTick(J)};   % set Label
166 end
167
168 YLim = [y_min, y_max];
169 set(gca,'YLim',YLim,'YTick',YTick, 'YTickLabel',YTickLabel);
170
171 % lables, title and legend
172 ylabel(yaxislabel(selector));
173 title(figure_title(selector));
174 legend(num2str(sensor_addr(1,1:sensor_count)'));
```

```
175
176 zoom on;
177 grid on;
```

Listing B.22: Alle Graphen ein oder aus schalten

```
1  %-------------------------------------------------------------------------
2  % Project:       Diploma - display sensor data
3  % Date:          13/06/2008
4  % Last Update:   13/06/2008
5  % Author:        Stephan Plaschke
6  % Discreption:   Switch all graphs on/off
7  %-------------------------------------------------------------------------
8
9  function allgraph_onoff(figure_select)
10
11 global ssw;                      % Array for switching sensor graphs on/off
12 global sensor_addr;              % array with valid sensor addresses
13 global sensor_count;             % number of sensors
14 global radio_buttons_onoff;      % status variable for radiobuttons
15
16
17
18 if radio_buttons_onoff(figure_select) == 1
19     for i=1:sensor_count
20         uicontrol(figure_select,'style','radiobutton','Value',0,'string',...
21             strcat('Sensor ',': ',num2str(sensor_addr(1,i))),...
22             'units','normalized','position',[.92 .9-i*.05 .07 .03],...
23             'callback',strcat('global ssw; ssw(',num2str(figure_select),...
24             ',',num2str(i),')=not(ssw(',num2str(figure_select),',',...
25             num2str(i),'));plot_graph(', num2str(figure_select),');'));
26     end
27
28     uicontrol(figure_select,'style','radiobutton','Value',0,'string',...
29            'All on/off','units','normalized','position',[.92 .2 .07 .03],...
30            'callback',strcat('allgraph_onoff(',num2str(figure_select),');'));
31     ssw(figure_select,:) = zeros;
32     radio_buttons_onoff(figure_select) = 0;
33 else
34     for i=1:sensor_count
35         uicontrol(figure_select,'style','radiobutton','Value',1,'string',...
36             strcat('Sensor ',': ',num2str(sensor_addr(1,i))),...
37             'units','normalized','position',[.92 .9-i*.05 .07 .03],...
38             'callback',strcat('global ssw; ssw(',num2str(figure_select),...
39             ',',num2str(i),')=not(ssw(',num2str(figure_select),',',...
40             num2str(i),')); plot_graph(', num2str(figure_select),');'));
41     end
42     uicontrol(figure_select,'style','radiobutton','Value',1,'string','All on/off',...
43            'units','normalized','position',[.92 .2 .07 .03],...
44            'callback',strcat('allgraph_onoff(',num2str(figure_select),');'));
45     ssw(figure_select,:) = ones;
46     radio_buttons_onoff(figure_select) = 1;
47 end
48
49 plot_graph(figure_select);
```

Listing B.23: Konfigurationsfile

```
1 % This file is created by the MATLAB programm temp_kalibrierung.m
2 % written by Stephan Plaschke
3
4 12              % Total amount of active sensors
5 2               % Address of sensor 1
6 3               % Address of sensor 2
7 4               % Address of sensor 3
```

```
5               % Address of sensor 4
6               % Address of sensor 5
7               % Address of sensor 6
8               % Address of sensor 7
9               % Address of sensor 8
10              % Address of sensor 9
11              % Address of sensor 10
12              % Address of sensor 11
14              % Address of sensor 12
3               % Difference function order+1, only for temperature graph
-3.144925e-003  % 3 polynomes for one sensorgraph, sensor 1
2.286090e-001
1.539427e+000
-7.833739e-005  % sensor 2
1.750993e-002
4.223026e-001
-3.073221e-003  % sensor 3
1.928886e-001
8.206266e-001
-2.662026e-003  % sensor 4
1.762518e-001
1.771485e+000
-5.299826e-003  % sensor 5
4.369695e-001
-7.451265e+000
-2.663839e-003  % sensor 6
1.681840e-001
-2.276351e-001
-2.597427e-003  % sensor 7
1.663005e-001
8.324122e-001
-2.857149e-003  % sensor 8
1.666055e-001
-1.865120e+000
-1.747754e-003  % sensor 9
1.309077e-001
-4.117300e+000
-3.338203e-003  % sensor 10
1.504583e-001
1.070548e+001
-5.880626e-004  % sensor 11
5.140346e-002
4.949305e+000
-1.487335e-003  % sensor 12
9.924465e-002
-2.255723e+000
```

B.2.3. Darstellen eines Oszilloskop Ausdrucks

Listing B.24: M-File für die Darstellung eines Oszilloskop Ausdrucks

```
%-------------------------------------------------------------------------
% Project:      Diploma
% Date:         05/11/2007
% Last Update:  27/05/2008
% Author:       Stephan Plaschke
% Discreption:  Read oscilloscop data from ascii-file
%-------------------------------------------------------------------------

function show_osc_data(figure1)

% clear all variables and close all windows
clear all;
close all;
```

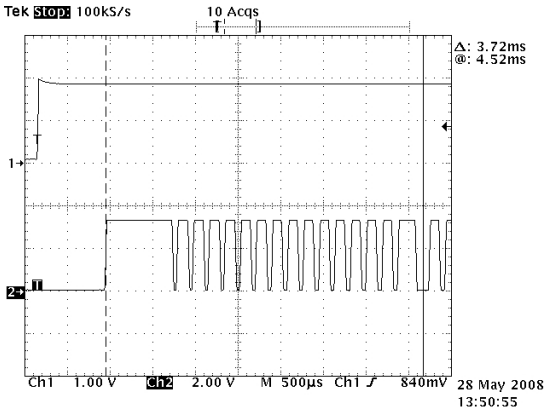
```
14
15 disp('------------------------------------------------------------------------')
16 disp('Read and plot oscilloskop-data')
17 disp('Use only *.asc files as input')
18 disp('As input use only filename with no extension')
19 disp('Author:   Stephan Plaschke')
20 disp('Date:     17.05.2008')
21 disp('------------------------------------------------------------------------')
22
23 % enter name of file
24 filename = input('Please enter the filename:','s');
25 filename = sprintf('%s.asc', filename);
26
27 % read data from file
28 data = load(filename, 'ascii');
29 data_length = length(data);             % get length of data
30 data_maxquarter = max(data) / 4;        % calculate
31 data_endval = (data_length-1) * 0.04;   % calclate endval in ms
32
33 % look for runin-code
34 for i=1:data_length
35     if ((data(i) >= data_maxquarter))
36         % if logical one occurs 60 times minimum, runin-code detected
37         for j=i:(i+60)
38             if (data(j) < data_maxquarter)
39                 break;
40             end
41         end
42         if (j == (i+60))
43             data_start = i;
44             break;
45         end
46     end
47 end
48
49 % look for end of CRC-byte
50 for i=data_start:data_length
51     if ((data(i) <= data_maxquarter))
52         % if logical zero occurs 60 times minimum, end of CRC-byte detected
53         for j=i:(i+60)
54             if (data(j) > data_maxquarter)
55                 break;
56             end
57         end
58         if (j == (i+60))
59             data_end = i;
60             break;
61         end
62     end
63     % if logical zero isn't occured 60 times until end of data, set
64     % data_end to last element
65     if (i == (data_length-60))
66         data_end = data_length;
67         break;
68     end
69 end
70
71 % clear data array where no data detected
72 data(1:data_start) = 0;
73 data(data_end:data_length) = 0;
74
75 % calculate real x-values
76 x_max = (data_end + 100) * 0.04;
77 x_min = (data_start - 100) * 0.04;
78
79 x=(0:0.04:data_endval);
80
```

```
% change data values to logical values
for i=1:data_length
    if (data(i) > data_maxquarter)
        data(i) = 1;
    else
        data(i) = 0;
    end
end

% create figure
figurename = sprintf('Osziloskopausdruck, Dateiname: %s', filename);
screensize = get(0,'ScreenSize');
figure('Name',figurename, 'NumberTitle','off', 'Position',[screensize]);

% plot data
plot(x,data)

% configure axes
ylabel('Logik-Pegel');
xlabel('Time/ms');
axis([x_min, x_max, 0,1.5])

% set Y-axis
YTick = [0,1,1.5];
YTickLabel = ['0';'1';' '];
set(gca,'YTick',YTick, 'YTickLabel',YTickLabel);


% vertical lines
annotation('line',[0.143 0.143],[0.654 0.72]);    %1er vertical line
annotation('line',[0.282 0.282],[0.654 0.72]);    %2er vertical line
annotation('line',[0.338 0.338],[0.654 0.72]);    %3er vertical line
annotation('line',[0.393 0.393],[0.654 0.72]);    %4er vertical line
annotation('line',[0.448 0.448],[0.654 0.72]);    %5er vertical line
annotation('line',[0.504 0.504],[0.654 0.72]);    %6er vertical line
annotation('line',[0.560 0.560],[0.654 0.72]);    %7er vertical line
annotation('line',[0.614 0.614],[0.654 0.72]);    %8er vertical line
annotation('line',[0.670 0.670],[0.654 0.72]);    %9er vertical line
annotation('line',[0.725 0.725],[0.654 0.72]);    %10er vertical line
annotation('line',[0.780 0.780],[0.654 0.72]);    %11er vertical line
annotation('line',[0.8355 0.8355],[0.654 0.72]);  %12er vertical line
annotation('line',[0.891 0.891],[0.654 0.72]);    %13er vertical line

% horizontal arrows
annotation('doublearrow',[0.143 0.282],[0.7 0.7]);   % runin
annotation('doublearrow',[0.282 0.338],[0.7 0.7]);   % addresse1
annotation('doublearrow',[0.338 0.393],[0.7 0.7]);   % addresse2
annotation('doublearrow',[0.393 0.448],[0.7 0.7]);   % addresse3
annotation('doublearrow',[0.448 0.504],[0.7 0.7]);   % addresse4
annotation('doublearrow',[0.504 0.560],[0.7 0.7]);   % data1
annotation('doublearrow',[0.560 0.614],[0.7 0.7]);   % data2
annotation('doublearrow',[0.614 0.670],[0.7 0.7]);   % data3
annotation('doublearrow',[0.670 0.725],[0.7 0.7]);   % data4
annotation('doublearrow',[0.725 0.780],[0.7 0.7]);   % data5
annotation('doublearrow',[0.780 0.8355],[0.7 0.7]);  % data6
annotation('doublearrow',[0.8355 0.891],[0.7 0.7]);  % CRC

% print text boxes
annotation('textbox',[0.1776 0.7232 0.07478 0.03256],'Interpreter','none',...
    'String',{'Runin code'},'FitBoxToText','on','LineStyle','none');
annotation('textbox',[0.3487 0.7232 0.08884 0.03256],'Interpreter','none',...
    'String',{'4 Address bytes'},'FitBoxToText','off','LineStyle','none');
annotation('textbox',[0.50 0.7232 0.06697 0.03256],'Interpreter','none',...
    'String',{'Databyte 1'},'FitBoxToText','off','LineStyle','none');
annotation('textbox',[0.555 0.7232 0.06697 0.03256],'Interpreter','none',...
    'String',{'Databyte 2'},'FitBoxToText','off','LineStyle','none');
annotation('textbox',[0.61 0.7232 0.06697 0.03256],'Interpreter','none',...
```

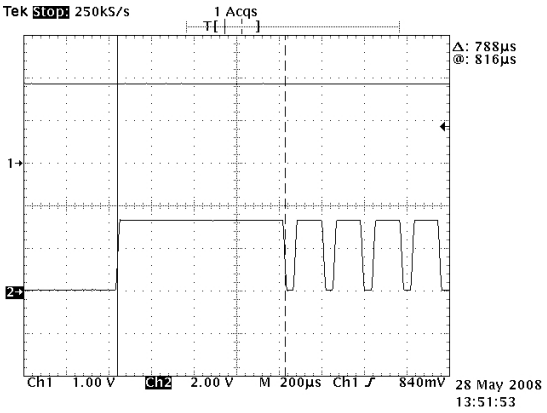
```
148     'String',{'Databyte 3'},'FitBoxToText','off','LineStyle','none');
149 annotation('textbox',[0.665 0.7232 0.06697 0.03256],'Interpreter','none',...
150     'String',{'Databyte 4'},'FitBoxToText','off','LineStyle','none');
151 annotation('textbox',[0.72 0.7232 0.06697 0.03256],'Interpreter','none',...
152     'String',{'Databyte 5'},'FitBoxToText','off','LineStyle','none');
153 annotation('textbox',[0.775 0.7232 0.06697 0.03256],'Interpreter','none',...
154     'String',{'Databyte 6'},'FitBoxToText','off','LineStyle','none');
155 annotation('textbox',[0.835 0.7232 0.06697 0.03256],'Interpreter','none',...
156     'String',{'CRC Byte'},'FitBoxToText','off','LineStyle','none');
```

C. Zeichnungen

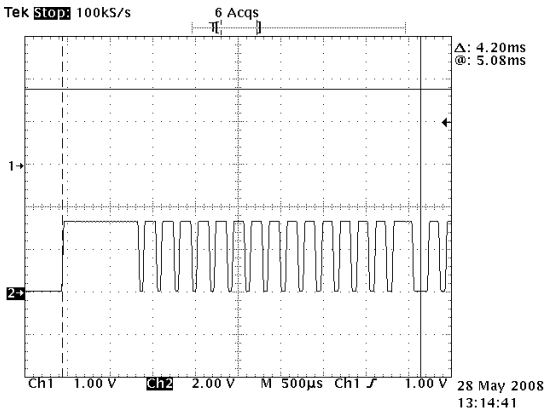
C.1. Zeitmessung der Run-In Sequenz



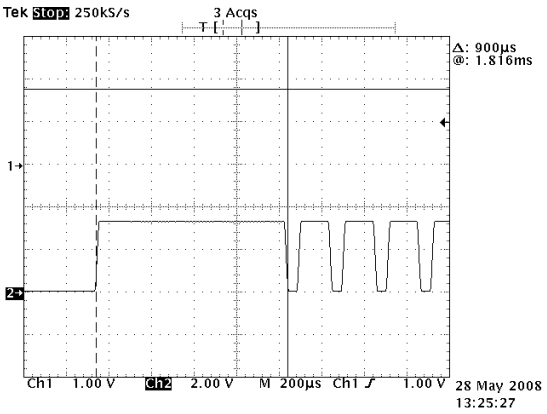
(a) Run-In-Sequenz bei -15°C



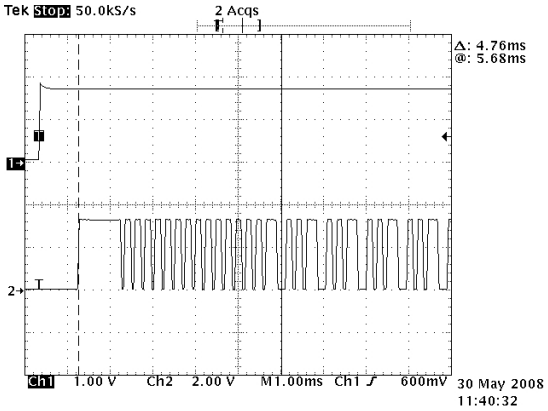
(b) Run-In-Block bei -15°C



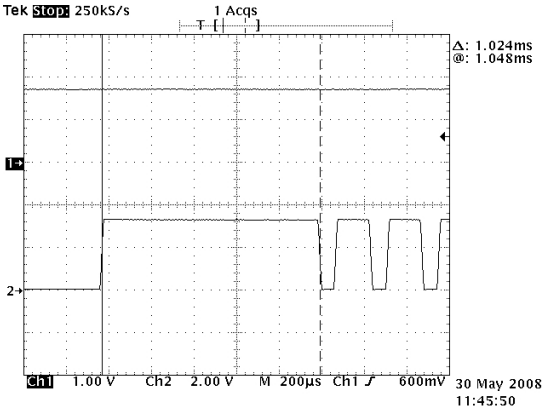
(c) Run-In-Sequenz bei 23°C



(d) Run-In-Block bei 23°C



(e) Run-In-Sequenz bei 60°C



(f) Run-In-Block bei 60°C

Bild C.1.: Zeitmessung der Run-In Sequenz bei verschiedenen Temperaturen, Auswertung in Kapitel 2.2.4

C.2. Darstellung des Übertragungsprotokolls

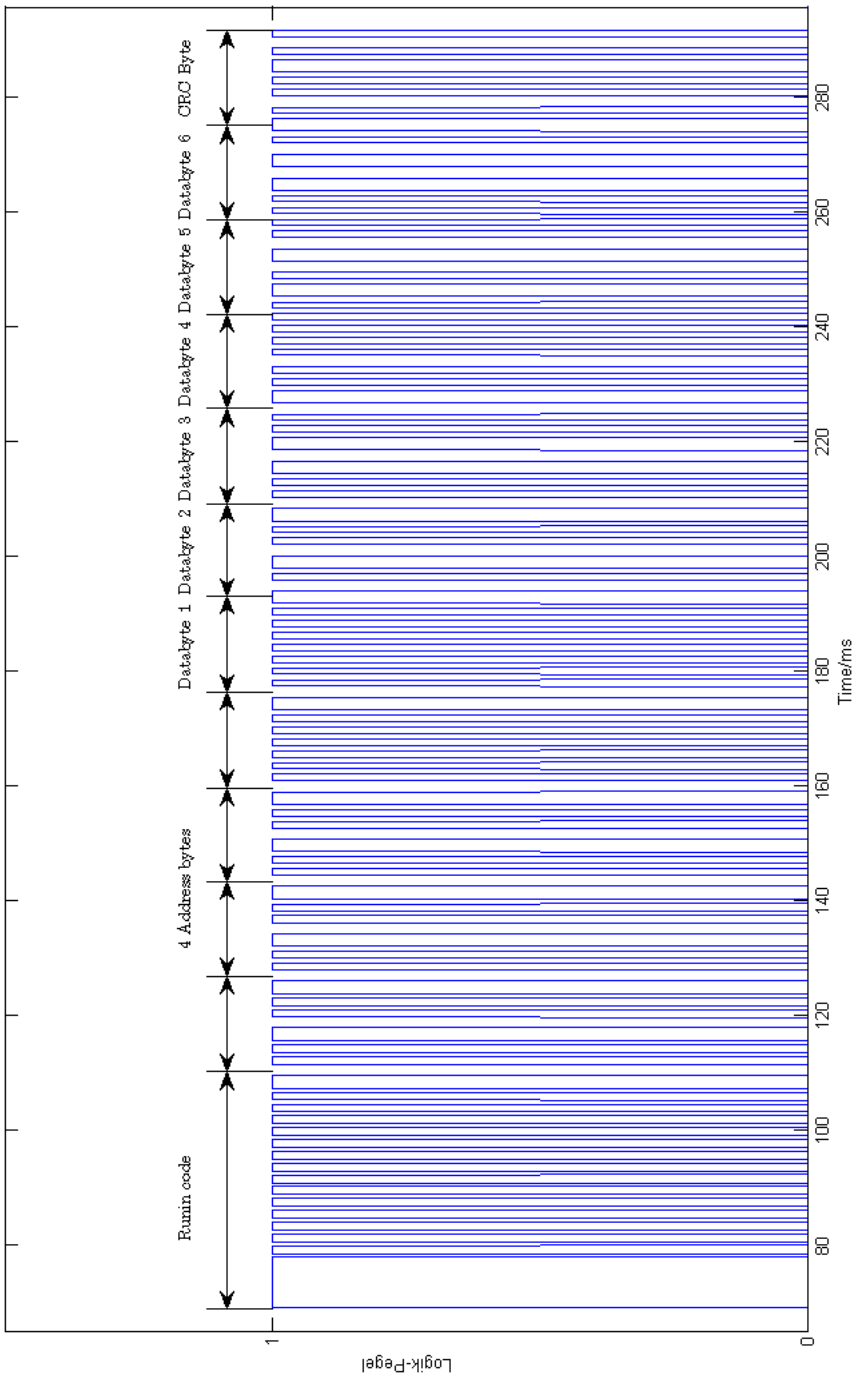


Bild C.2.: Darstellung des Übertragungsprotokolls (MATLAB Ausdruck)

D. Hardwaredesign

D.1. Schaltpläne

D.1.1. Schaltplan: Receiver

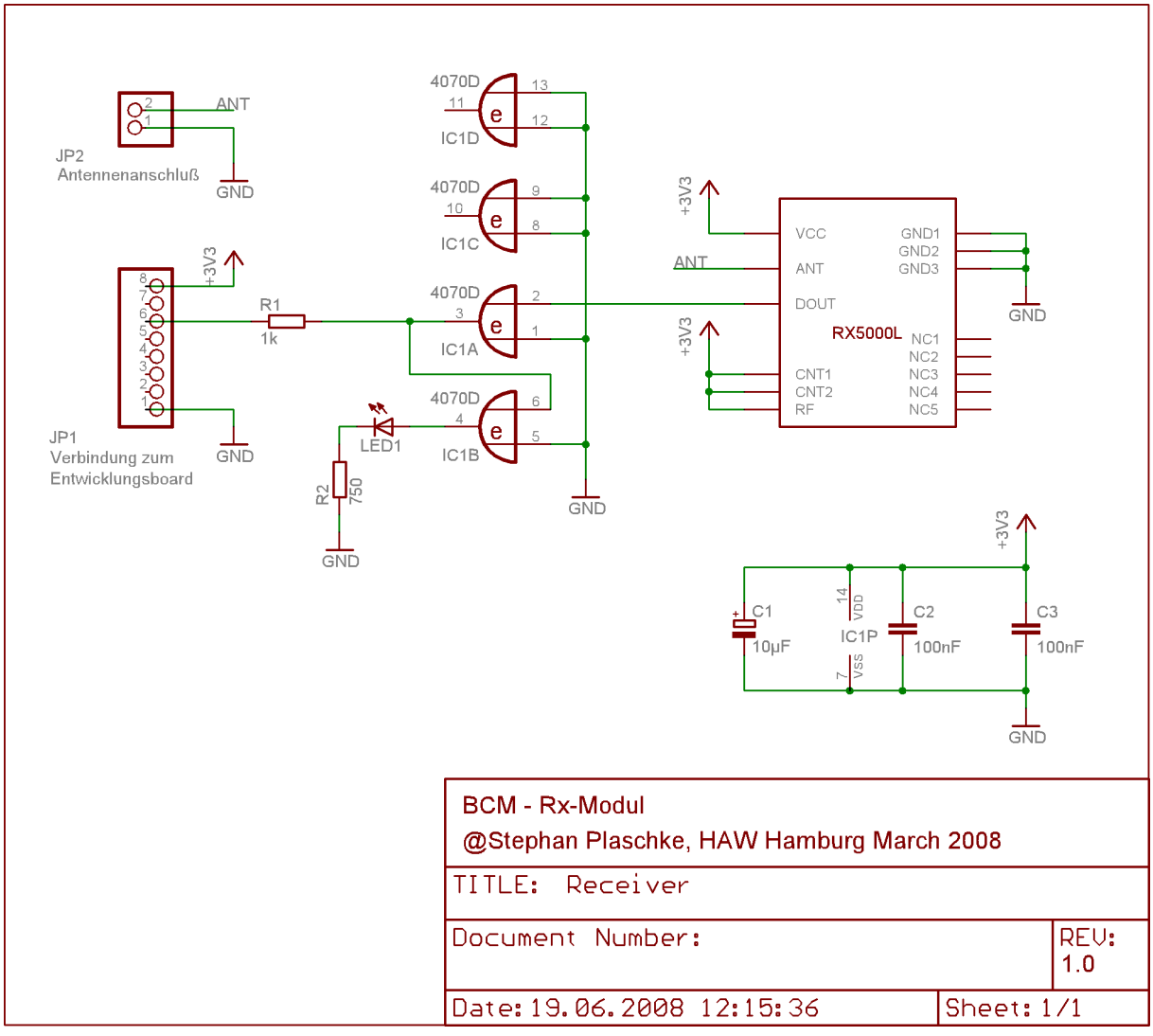


Bild D.1.: Schaltplan des Receiverboards

D.1.2. Schaltplan: Entwicklungsboard

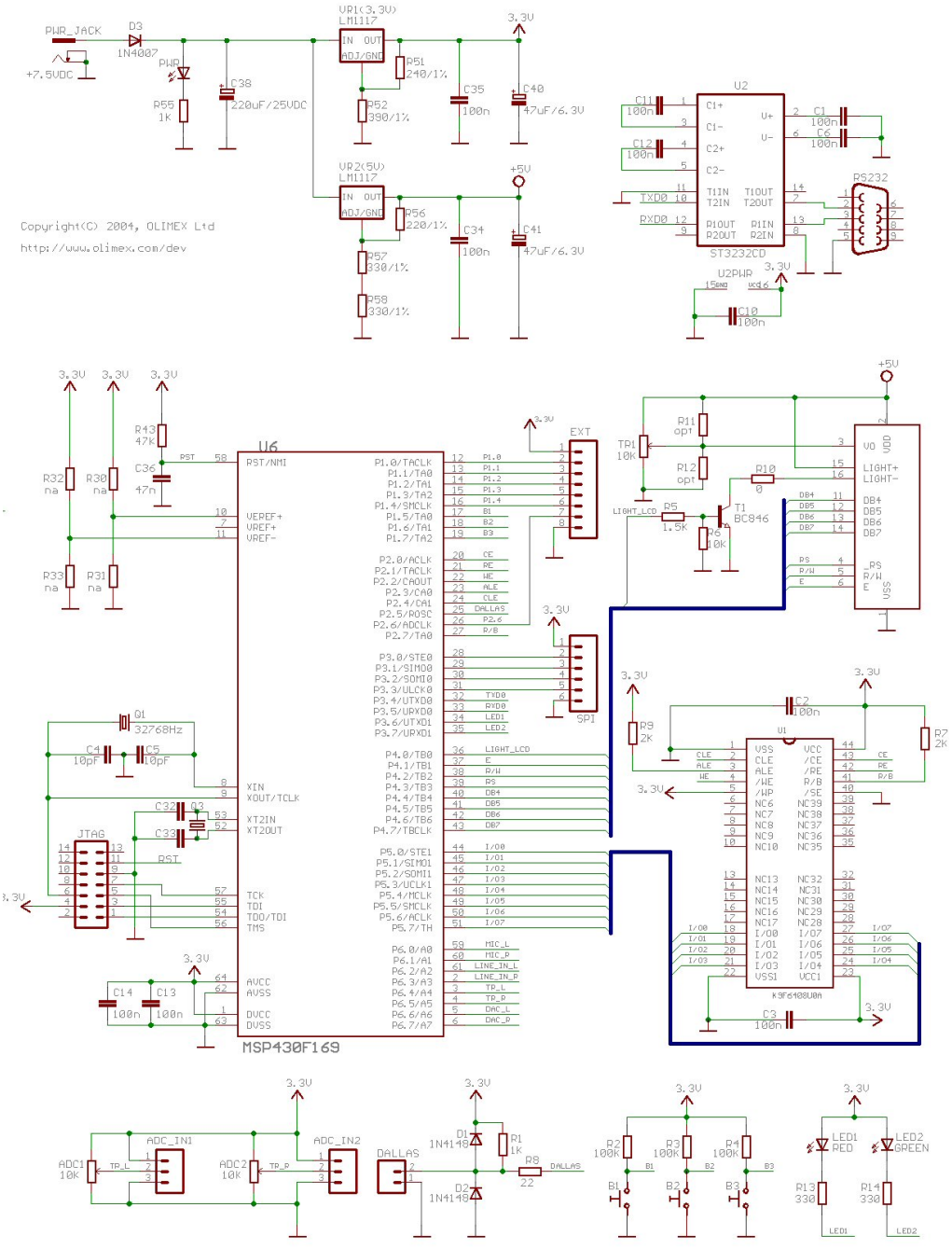


Bild D.2.: Erster Schaltplanteil des Olimex Entwicklungsboards [18]

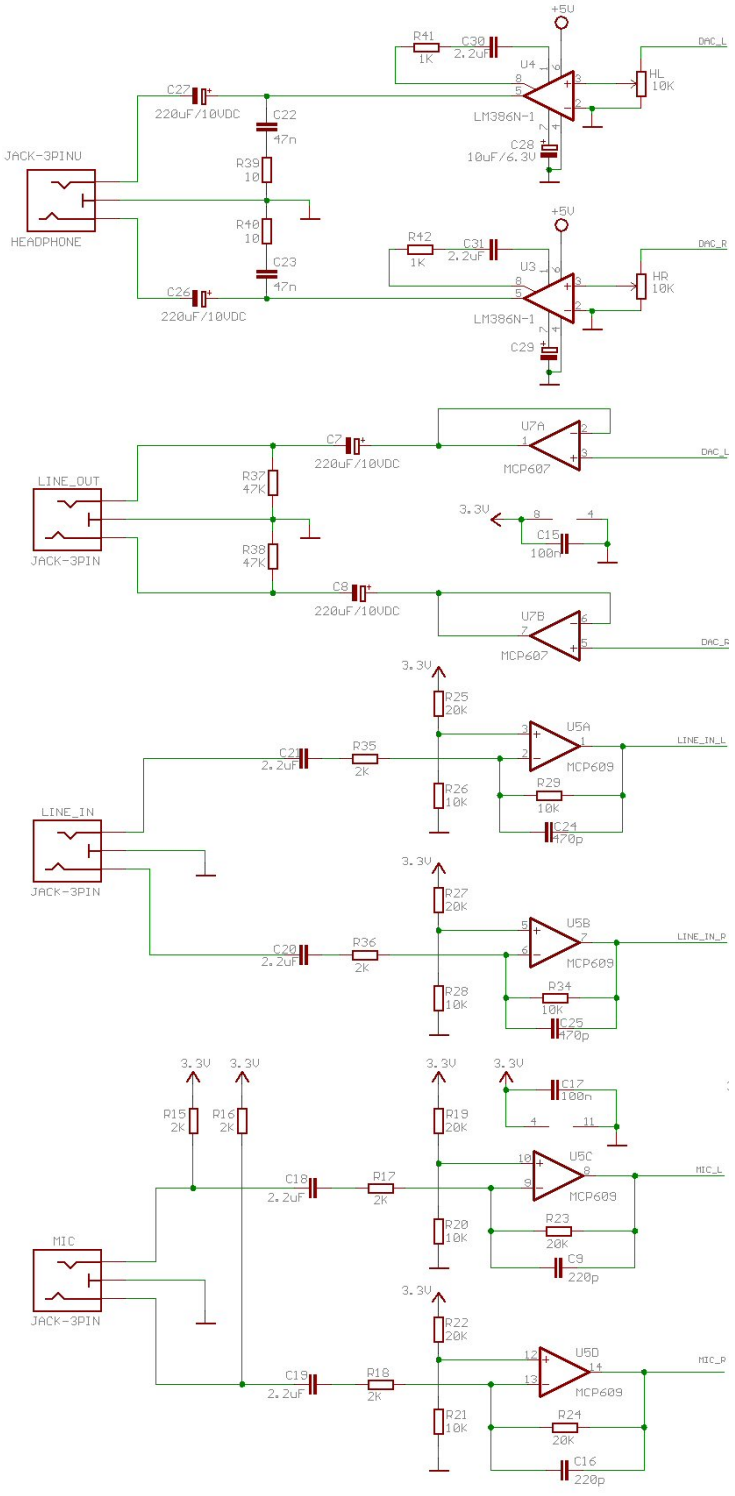


Bild D.3.: Zweiter Schaltplanteil des Olimex Entwicklungsboards [18]

D.1.3. Schaltplan: Sensor

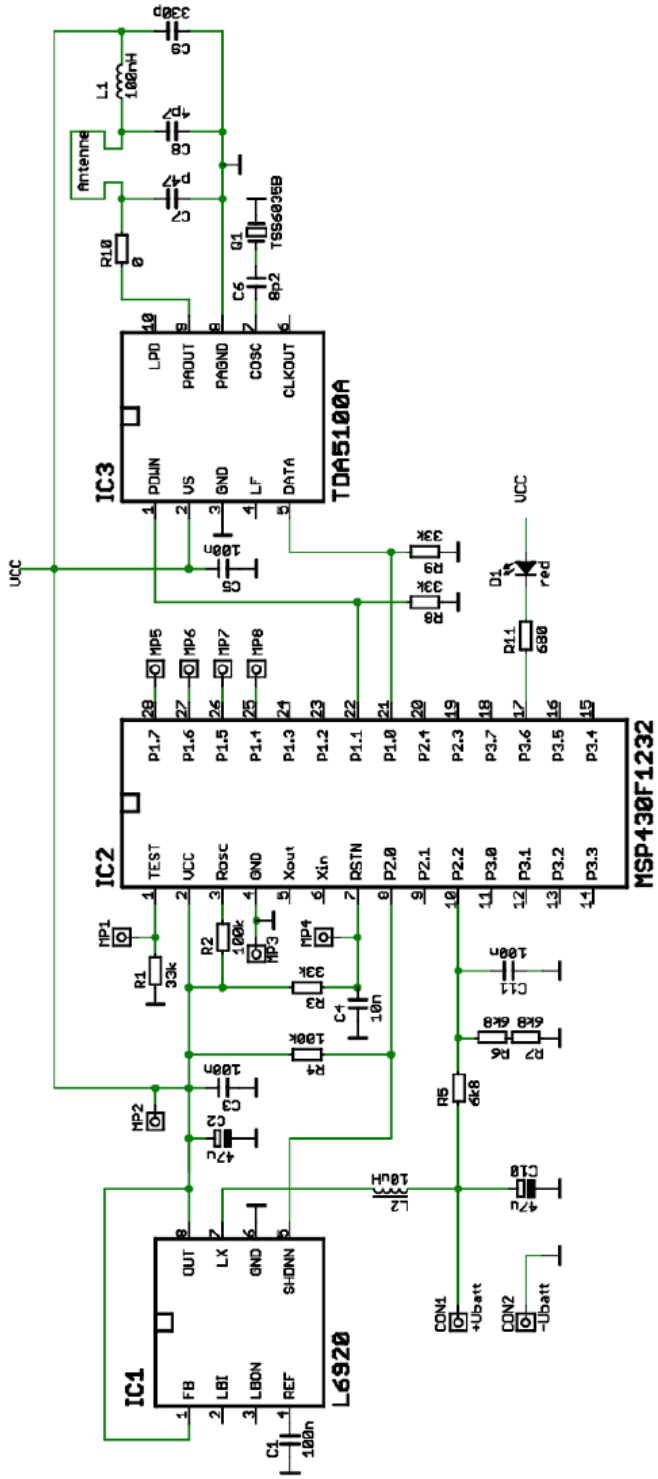


Bild D.4.: Schaltplan des Sensor-Systems [21]

D.2. Platinenlayout

D.2.1. Platinenlayout: Receiver

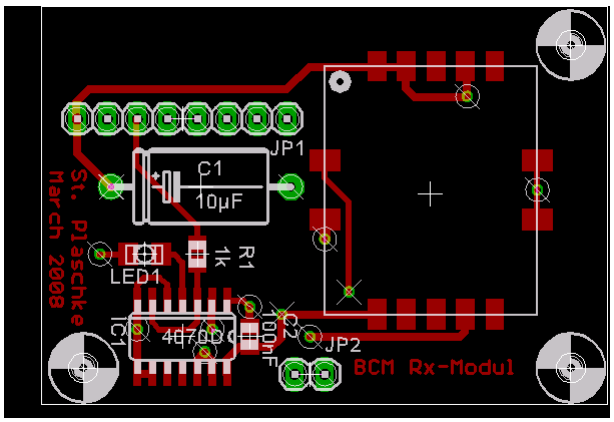


Bild D.5.: Platinenlayout der Empfängerplatine (top layer)

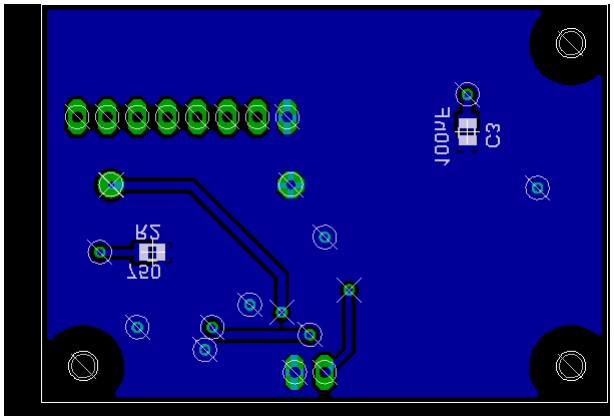


Bild D.6.: Platinenlayout der Empfängerplatine (bottom layer)

E. Messwerte

E.1. Kalibrierungsmessung

Listing E.1: Aufgenommene Messwerte zur Temperaturkalibrierung

```
1  WED 04-Jun-2008 12:30:00  6  0 73 2d a5 34 76 ae
2  WED 04-Jun-2008 12:30:00  a  0 69 2d 78 33 eb ff
3  WED 04-Jun-2008 12:30:00  b  0  0 2f  d 36 1c  3
4  WED 04-Jun-2008 12:30:00  9  0 55 2d e1 34 62 d7
5  WED 04-Jun-2008 12:30:00  8  0 42 2d ff 34 8b 36
6  WED 04-Jun-2008 12:30:00  4  0 3e 2e 3b 35 16 1d
7  WED 04-Jun-2008 12:30:00  c  0 27 2d d2 34 14 f4
8  WED 04-Jun-2008 12:30:00  5  0 3b 2e 68 34 b4 e9
9  WED 04-Jun-2008 12:30:00  2  0 3b 2d ff 34 76 b8
10 WED 04-Jun-2008 12:30:00  3  0 48 2d 78 33 eb d7
11 WED 04-Jun-2008 12:30:00  e  0 5f 2e 68 34 76 55
12 WED 04-Jun-2008 12:30:00  7  0 4c 2e  e 34 ed a3
13 WED 04-Jun-2008 12:38:00  c  0 52 2d e1 33 d7 76
14 WED 04-Jun-2008 12:38:00  e  0 7d 2e 95 35 2b d6
15 WED 04-Jun-2008 12:38:00  4  0 66 2e 3b 34 8b d9
16 WED 04-Jun-2008 12:38:00  a  0 8a 2d 69 33 74 92
17 WED 04-Jun-2008 12:38:00  9  0 7a 2d f0 33 d7 5b
18 WED 04-Jun-2008 12:38:00  8  0 69 2e  e 34 4d 29
19 WED 04-Jun-2008 12:38:00  b  0 1a 2f 1c 36  8 1c
20 WED 04-Jun-2008 12:38:00  2  0 5f 2e  e 34 8b d3
21 WED 04-Jun-2008 12:38:00  3  0 76 2d b4 33 60 ae
22 WED 04-Jun-2008 12:38:00  7  0 70 2e 1d 34 9f fe
23 WED 04-Jun-2008 12:38:00  6  0 80 2d b4 34 4d 77
24 WED 04-Jun-2008 12:38:00  5  0 5f 2e 68 35  2 3a
25 WED 04-Jun-2008 12:48:00  2  0 7a 2d ff 34 76 f9
26 WED 04-Jun-2008 12:48:00  7  0 91 2e 2c 34 c8 79
27 WED 04-Jun-2008 12:48:00  5  0 80 2e 68 34 8b 6d
28 WED 04-Jun-2008 12:48:00  3  0 a1 2d c3 33 eb 85
29 WED 04-Jun-2008 12:48:00  a  0 b5 2d 78 33 eb 23
30 WED 04-Jun-2008 12:48:00  e  0 a8 2e 77 34 76 bd
31 WED 04-Jun-2008 12:48:00  4  0 87 2e 3b 34 dd 6e
32 WED 04-Jun-2008 12:48:00  b  0 45 2f 1c 36 41  a
33 WED 04-Jun-2008 12:48:00  8  0 87 2e  e 34 39 b3
34 WED 04-Jun-2008 12:48:00  6  0 a1 2d b4 34  0 1b
35 WED 04-Jun-2008 12:48:00  9  0 9e 2d f0 34  0 6f
36 WED 04-Jun-2008 12:48:00  c  0 76 2d f0 34 39 aa
37 WED 04-Jun-2008 12:53:00  2  0 9b 2e  e 34 4d d1
38 WED 04-Jun-2008 12:53:00  9  0 bf 2d f0 34 62 2c
39 WED 04-Jun-2008 12:53:00  a  0 d0 2d 78 33 c2 6f
40 WED 04-Jun-2008 12:53:00  7  0 a8 2e 2c 34 ed 65
41 WED 04-Jun-2008 12:53:00  6  0 bf 2d b4 34 62 67
42 WED 04-Jun-2008 12:53:00  b  0 66 2f 1c 36  8 60
43 WED 04-Jun-2008 12:53:00  3  0 bf 2d c3 33 c2 b2
44 WED 04-Jun-2008 12:53:00  8  0 a8 2e  e 34 4d e8
45 WED 04-Jun-2008 12:53:00  5  0 a1 2e 77 34 9f 47
46 WED 04-Jun-2008 12:53:00  c  0 91 2d e1 34 4d 28
47 WED 04-Jun-2008 12:53:00  4  0 a8 2e 3b 34 76 ea
48 WED 04-Jun-2008 12:53:00  e  0 c9 2e 77 34 76 dc
49 WED 04-Jun-2008 13:00:00  7  0 da 2e 3b 34 ed  0
50 WED 04-Jun-2008 13:00:00  9  0 ea 2d ff 33 d7 c4
```

```
WED 04-Jun-2008 13:00:00  8  0 d0 2e 1d 34 9f 51
WED 04-Jun-2008 13:00:00  b  0 87 2f 3a 35 ca 66
WED 04-Jun-2008 13:00:00  4  0 d0 2e 4a 34 8b 1e
WED 04-Jun-2008 13:00:00  a  0 fe 2d 87 33 85 f9
WED 04-Jun-2008 13:00:00  5  0 c6 2e 59 34 76 e7
WED 04-Jun-2008 13:00:00  6  0 ea 2d c3 34 c8 ef
WED 04-Jun-2008 13:00:00  2  0 c2 2e  e 34 62 a7
WED 04-Jun-2008 13:00:00  e  0 f4 2e 68 35 53 da
WED 04-Jun-2008 13:00:00  3  0 e7 2d d2 33 74 4d
WED 04-Jun-2008 13:00:00  c  0 bc 2d f0 34 8b d2
WED 04-Jun-2008 13:08:00  7  1  1 2e 4a 34 62 24
WED 04-Jun-2008 13:08:00  3  1  b 2d e1 33 c2 25
WED 04-Jun-2008 13:08:00  e  1 1c 2e 86 34 8b  4
WED 04-Jun-2008 13:08:00  6  1  5 2d c3 34  0 c9
WED 04-Jun-2008 13:08:00  4  0 f7 2e 59 34 dd 7c
WED 04-Jun-2008 13:08:00  9  1 12 2e  e 34  0 1f
WED 04-Jun-2008 13:08:00  a  1 1c 2d 96 34  0 89
WED 04-Jun-2008 13:08:00  b  0 b2 2f 1c 36  8 b4
WED 04-Jun-2008 13:08:00  5  0 f1 2e 77 34 ed 65
WED 04-Jun-2008 13:08:00  c  0 e7 2d f0 34 76 74
WED 04-Jun-2008 13:08:00  8  0 f7 2e 2c 34 b4 6c
WED 04-Jun-2008 13:08:00  2  0 e7 2e  e 35  2 e3
WED 04-Jun-2008 13:19:00  2  1 15 2e 3b 34 62 44
WED 04-Jun-2008 13:19:00  3  1 36 2d e1 34 14 c9
WED 04-Jun-2008 13:19:00  9  1 3d 2e  e 34 9f af
WED 04-Jun-2008 13:19:00  a  1 4d 2d a5 34 39 d2
WED 04-Jun-2008 13:19:00  5  1 22 2e 95 35 3f 86
WED 04-Jun-2008 13:19:00  4  1 26 2e 77 34 8b d4
WED 04-Jun-2008 13:19:00  b  0 e0 2f 58 35 df 76
WED 04-Jun-2008 13:19:00  6  1 39 2d d2 34 4d a9
WED 04-Jun-2008 13:19:00  e  1 47 2e 86 34 8b 5f
WED 04-Jun-2008 13:19:00  7  1 30 2e 59 34 8b ef
WED 04-Jun-2008 13:19:00  c  1 12 2e  e 34  0  b
WED 04-Jun-2008 13:19:00  8  1 29 2e 1d 34 39  f
WED 04-Jun-2008 13:30:00  4  1 4a 2e 86 34 b4 76
WED 04-Jun-2008 13:30:00  c  1 40 2e  e 34 b4 ed
WED 04-Jun-2008 13:30:00  e  1 6e 2e b3 34 9f 57
WED 04-Jun-2008 13:30:00  2  1 43 2e 59 34 62 70
WED 04-Jun-2008 13:30:00  5  1 43 2e 86 34 ed 27
WED 04-Jun-2008 13:30:00  b  1  8 2f 67 36 31 4d
WED 04-Jun-2008 13:30:00  a  1 85 2d f0 34 76  0
WED 04-Jun-2008 13:30:00  3  1 64 2e 1d 34 28 58
WED 04-Jun-2008 13:30:00  9  1 6b 2e 3b 33 eb bf
WED 04-Jun-2008 13:30:00  7  1 64 2e 68 35 16 16
WED 04-Jun-2008 13:30:00  8  1 54 2e 86 34 8b 5b
WED 04-Jun-2008 13:30:00  6  1 64 2d f0 34 28 b3
WED 04-Jun-2008 14:03:00  a  1 a3 2d ff 33 eb b3
WED 04-Jun-2008 14:03:00  2  1 6e 2e a4 34 76 b4
WED 04-Jun-2008 14:03:00  7  1 89 2e b3 34 c8 ff
WED 04-Jun-2008 14:03:00  9  1 a0 2e 77 34 b4 60
WED 04-Jun-2008 14:03:00  4  1 78 2e c2 34 9f 2b
WED 04-Jun-2008 14:03:00  5  1 75 2e ef 35 2b bf
WED 04-Jun-2008 14:03:00  8  1 82 2e 95 34 dd c8
WED 04-Jun-2008 14:03:00  e  1 a0 2e fe 34 ed a6
WED 04-Jun-2008 14:03:00  3  1 8c 2e 4a 33 85 4d
WED 04-Jun-2008 14:03:00  6  1 93 2e 3b 34 39 9d
WED 04-Jun-2008 14:03:00  b  1 40 2f a3 36 7e 8e
WED 04-Jun-2008 14:03:00  c  1 61 2e 68 34 dd c3
WED 04-Jun-2008 14:14:00  6  1 b0 2e 59 34 39 dc
WED 04-Jun-2008 14:14:00  9  1 c8 2e 95 34  0 5e
WED 04-Jun-2008 14:14:00  a  1 cb 2e 68 33 eb 4f
WED 04-Jun-2008 14:14:00  c  1 89 2e 68 34 9f 69
WED 04-Jun-2008 14:14:00  b  1 61 2f d0 36  8 aa
WED 04-Jun-2008 14:14:00  8  1 a3 2e a4 35 16 12
WED 04-Jun-2008 14:14:00  2  1 8c 2e a4 34 b4 94
WED 04-Jun-2008 14:14:00  e  1 c4 2e fe 34 b4 9b
WED 04-Jun-2008 14:14:00  3  1 ad 2e 68 33 85 4e
```

```
118 WED 04-Jun-2008 14:14:00  4  1 b4 2f  d 34 ed 5b
119 WED 04-Jun-2008 14:14:00  5  1 99 2e fe 35 2b 42
120 WED 04-Jun-2008 14:14:00  7  1 ad 2e d1 34 ed 9c
121 WED 04-Jun-2008 14:56:00  3  1 d5 2e a4 34 28 50
122 WED 04-Jun-2008 14:56:00  c  1 aa 2e b3 34 4d 43
123 WED 04-Jun-2008 14:56:00  a  1 ef 2e 4a 34 4d e8
124 WED 04-Jun-2008 14:56:00  9  1 ec 2e b3 34  0 5c
125 WED 04-Jun-2008 14:56:00  2  1 b4 2e e0 34 dd 81
126 WED 04-Jun-2008 14:56:00  8  1 cb 2e ef 34 c8 ee
127 WED 04-Jun-2008 14:56:00  7  1 d5 2f  d 34 76 a2
128 WED 04-Jun-2008 14:56:00  4  1 c4 2f  d 35 16 d1
129 WED 04-Jun-2008 14:56:00  6  1 d8 2e 77 34 39 9a
130 WED 04-Jun-2008 14:56:00  b  1 89 30  c 36 1c 95
131 WED 04-Jun-2008 14:56:00  5  1 c1 2f 2b 35 53 b6
132 WED 04-Jun-2008 14:56:00  e  1 e9 2f 49 35  2 b7
```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ort, Datum Unterschrift