Eike Mense

Hard- und Softwareentwicklung für einen drahtlosen Batterie-Zellen-Sensor zur elektrochemischen Impedanzspektroskopie

Masterarbeit eingereicht im Rahmen der Masterpüfung
im gemeinsamen Studiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
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. Alfred Ebberg

Abgegeben am 28. November 2014

Eike Mense

Thema der Masterarbeit

Hard- und Softwareentwicklung für einen drahtlosen Batterie-Zellen-Sensor zur elektrochemischen Impedanzspektroskopie

Stichworte

Lithium-Eisen-Phosphat-Akkumulatoren, elektrochemischen Impedanzspektroskopie, Batterie-Zellen-Sensor, drahtlose Kommunikation, Mikrocontroller, CC430F5137, funksynchronisierte Messung, phasenrichtige Messung, Goertzel-Algorhytmus

Kurzzusammenfassung

Diese Arbeit behandelt die Entwicklung eines Messsystems, mit dem es möglich sein soll, Phasenunterschiede von Wechselspannungen kleiner Amplitude bei großem Offsetspannungen zu messen. Diese Messungen sollen für eine elektrochemischen Impedanzspektroskopie mit verringertem Hardwareaufwand genutzt werden. Dabei kommen drahtlose Zellen-Sensoren zum Einsatz, um für jede Zelle eines Lithium-Eisen-Phosphat-Akkumulators eigene Werte erfassen zu können.

Eike Mense

Title of the master thesis

Hard- and software development for a wireless battery cell sensor used for electrochemical impedance spectroscopy

Keywords

Lithium iron phosphate battery, electrochemical impedance spectroscopy, wireless communication, Microprocessor, CC430F5137, radio-synchronised measurement, phase measurement, Goertzel-Algorhytm

Abstract

This report discusses the development of a measuring system that is capable of measuring phase differences of small alternating voltages with high offset voltages. This measurements shall be used for an electrochemical impedance spectroscopy with decreased hardware costs. Therefore wireless communicating cell sensors are used to obtain values per each cell of a Lithium iron phosphate battery.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
ADC	Analog Digital Converter
APU	Auxilary Power Unit
ASK	Amplitude-Shift-Keyin
BMU	Battery Managment Unit
BATSEN	Forschungsprojekt Drahtlose Zellensensoren für Fahrzeugbatterien
CPE	Constant Phase Element
CRC	cyclic redundancy check
DCO	Digital Controlled Oscillator
DENIS	detailed electrochemistry and numerical impedance simulation
DFN	Dual-Flat No-Lead-Package
DFT	Discrete-Foruier-Transformation
EIS	Elektrochemische Impedanz Spektroskopie
ENOB	Effektive Number of Bits
ESB	Ersatzschaltbild
FIFO	First in - First Out
FIR	Finite Impuls Responce
FFT	Fast-Foruier-Transformation
HAW	Hochschule für angewandte Wissenschaften
I2C	Inter-Integrated Circuit
IC	Integrated-Circuit
ISM-Band	Industrial-, Scientific and Medical-Band
ISR	Interrupt Service Routine
$LiCoO_2$	Lithium-Cobalt(III)-oxid
$LiFePO_4$	Lithium-Eisenphosphat-Akkumulator
LiIon-Akku	Lithium-Ionen Akkumulator
LPM	Low Power Mode
OOK	On-Off-Keying
PCB	Printed Circuit Board
PWM	Pulsweiten-Modulation
QFN	Quad Flat No-Lead-Package

Abkürzung	Bedeutung
RAM	Random Access Memory
RF	Radio Frequency - dt.: Funkfrequenz
RFID	Radio-Frequency Identification
RX	Receive
SAR	Successive approximation register
SoC	State of Charge
SoH	State of Health
SOP	Small-Outline-Package
SOT	Small-Outline-Transistor
SPI	Serial Peripheral Interface
SPI-MISO	Serial Peripheral Interface - Master in Slave out
SPI-MOSI	Serial Peripheral Interface - Master out Slave in
SPI -SCLK	Serial Peripheral Interface - Serial Clock
TSSOP	Thin-Shrink Small-Outline-Package
TX	Transmitt
USART	Universal Synchronous/Asynchronous Receiver Transmitter
ZARC	Z-Arch $\Rightarrow$ Impedanz und Bogenelement

Verzeichnis der Begriffe und Definitionen

Akkumulator
Ein Akkumulator beschreibt eine Verschaltung galvanischer Zellen, der nach einer Entladung wiederaufladbar ist.

Analog-Digital-Umsetzer
Analog-Digital-Umsetzer (engl. analog-digital-converter (ADC)) generieren aus einer elektrischen Eingangsspannung einen digitalen Zahlenwert, der von einem Mikrocontroller aufgenommen werden kann. Dabei wird die Auflösung eines ADCs mit N Bit angegeben. Innerhalb dieser Arbeit werden verschiedene Arten von ADCs erwähn.

- **Sucessive Approximation Register ADC** (SAR) wenden eine Wägeverfahren an, um sich dem Spanungswert Schrittweise zu nähern. Sie benötigen einen N Bit DAC und einen Komparator sowie N Takte für die Umsetzung.
- **Flash-ADC**s wandeln den Eingangswert in einem Taktschritt und benötigen dafür N^2 Komparatoren.
- Δ-Σ**-ADCs** wenden eine Überabtastung und anschließende Dezimierung an. Sie benötigen einen 1 Bit-DAC , einen Komparator, einen Subtrahierer, einen Integrator und einen Dezimationsfilter.

Batterie
In dieser Arbeit wird mit dem Begriff Batterie die elektrische Verschaltung mehrerer galvanischer Zellen bezeichnet. Der Begriff wir teilweise analog zum englischen Sprachgebrauch auch für einen Verbund aus Primärzellen verwendet, da Sekundärzellen in dieser Arbeit nicht behandelt werden.

Elektrochemische Impedanzspektroskopie
EIS beschreibt die Aufnahme eines frequenzabhängigen Impedanzverlaufes durch Messung von Strom und Spannung bei verschiedenen Signalfrequenzen.

Package
Als Package wird im Rahmen dieser Arbeit das Gehäuse eines integrierten Schaltkreises (Integrated-Circuit IC). Dabei wird die englische Bezeichnung verwendet, da sie im üblichen Sprachgebrauch verankert ist und Teil der Bezeichnung der verschiedenen Gehäusetypen ist. Dabei beschreiben die Begriffe Quad-Flat-No-Lead-Package (QFN) und Dual-Flat-No-Lead-Package (DFN) kleine Packages ohne Lötfahnen außerhalb der Gehäusefläche. Da die Lötflächen unterhalb des Gehäuses liegen, lassen diese sich nicht mit einem regulären Lötkolben verarbeiten. Thin-Shrink Small-Outline-Package (TSSOP) und Small-Outline-Packag (SOP) bezeichnen Packages mit dicht nebeneinander-, aber außenliegenden Lötanschlüssen. Erstere haben dabei einen schmaleren Körper (4.4 mm) als zweitere (7.5 mm). Das Package Small-Outline-Transistor (SOT) ist für Transistoren gedacht, aufgrund seiner sehr schmalen Bauart (1.6 mm) jedoch auch für sehr kleine IC's geeignet. Die Zahl nach der Package-Bezeichnung gibt die Anzahl der PINs an.

Zelle
Als Zelle wird in dieser Arbeit ein einzelnes galvanisches Element samt seiner mechanischen Umfassung und den zugehörigen elektrischen Kontakten bezeichnet. Wiederaufladbare galvanische Zellen werden als Primärzelle bezeichnet. Nicht wiederaufladbare galvanische Zellen werden als Sekundärzelle bezeichnet.

1 Einleitung

Akkumulatoren auf Basis von Lithium-Ionen Interkalation haben einen festen Platz in der Verbraucherelektronik und anderen Anwendungen, die eine hohe Energiedichte bei mäßiger Leistungsdichte erfordern. In Elektrofahrzeugen werden derartige Akkumulatoren bereits erfolgreich eingesetzt, um die benötigten Leistungsdaten bei einem minimalen Gewicht erzielen zu können.

1.1 Motivation

Auch in den neuesten Generationen von zivilen Verkehrsflugzeugen kommen andere Batterie-Technologien zum Einsatz als die ansonsten heute noch in der Luftfahrt üblichen Nickel-Cadmium-Akkumulatoren. Diese Flugzeuggeneration zeichnet sich dadurch aus, dass die Entwicklung vom Ziel der Effizienzsteigerung getrieben wird. Das Mittel der Wahl ist neben der Entwicklung neuer effizienter Designs des Flugzeugkörpers und effizienterer Triebwerke vor allem die Gewichtsreduzierung. Hier konnten durch den Einsatz von Verbundwerkstoffen Einsparungen erzielt werden. Dennoch wird jede Baugruppe des Flugzeuges im Einzelnen auf eine mögliche Gewichtsreduzierung hin untersucht. Durch den gestiegenen Energiebedarf moderner Flugzeuge, insbesondere im Bereich der Unterhaltung, müssen Flugzeugbatterien trotz Effizienzsteigerung der Verbraucher heute gleich viel, oder mehr Energie liefern können als früher.

Abbildung 1.1 erklärt die möglichen Quellen für elektrische Energie in einer Boeing 787. Dabei dienen Flugzeugbatterien, anders als Batterien in Elektrofahrzeugen, niemals als Antrieb oder dauerhafte Versorgung von Komponenten, sondern lediglich als Puffer für Zeiten, in denen keine andere Energieversorgung möglich ist. Außerdem wird zum Starten der APU eine dedizierte APU-Batterie eingesetzt.

Um Gewicht zu sparen, ist es daher in jedem Fall wünschenswert, Akkumulatoren mit höherer Energiedichte einzusetzen als Nickel-Cadmium-Akkumulatoren (40-60 $^{Wh}/_{kg}$) [1]. Daher kommen zum Beispiel in der Boeing 787 Li-Ionen-Akkumulatoren mit einer 3- bis 3.5-fachen Energiedichte im Bereich von 120-210 $^{Wh}/_{kg}$ zum Einsatz. Diese Anwendungsgebiete erfordern, dass zum einen die Leistungsdichte der genutzten Akkumulatoren, im Vergleich zu

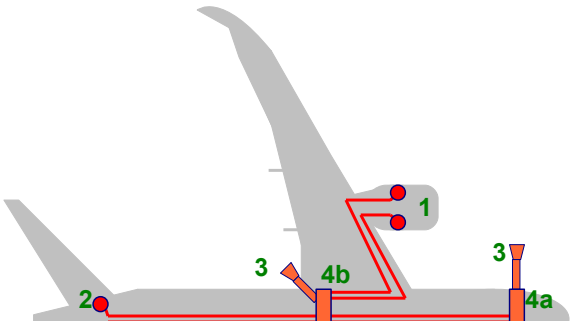


Abbildung 1.1: Quellen für elektrische Energie in einer Boeing 787. Bis auf die Einspeiseeinrichtung auf Steuerbord identisch.
1) Zwei Generatoren je Triebwerk, angetrieben über Zapfluft.
2) Generatoren an der Auxilary Power Unit APU, angetrieben über Zapfluft.
3) Anschluss für ein Bodenstromaggregat zur Versorgung von Aussen.
4a) Akkumulator für die Pufferversorgung der Flugzeugsysteme
4b) Akkumulator für den APU Start

solchen in Elektrokleingeräten verbauten, steigt, zum anderen werden deutlich größere Bauformen benötigt. Daraus resultiert eine gesteigerte Gesamtenergie, die in einem Akkumulator gespeichert wird, was im Fehlerfall die von einem System ausgehende Gefahr erhöht.

Die 787 ist der erste Flugzeugtyp, bei dem Batterien auf Lithium-Ionen Basis zum Einsatz kommen. Da die Zulassungskriterien für Luftfahrzeuge der für Boeing zuständigen US-amerikanischen Zulassungsbehörde Federal Aviation Agency (FAA) keine Li-Ionen-Akkumulatoren abdecken, wurde eine Reihe von verschiedenen Zusatzkriterien erlassen, die Boeing zu erfüllen hatte, um diesen Flugzeugtyp zuzulassen. Diese Zusatzkriterien sollten dafür sorgen, dass das von den neuen Akkumulatoren ausgehende Risiko in keinem Fall größer ist als das von bereits zulassungsfähigen Batterietypen [2]. Dennoch kam es am 07. und 16. Januar 2013 zu zwei sog. ‚ernsten Zwischenfällen' [3].

Am 07. Januar 2013 bemerkte das Wartungs- und Reinigungspersonal einer am Flughafen Boston, Massachusetts abgestellten Boeing 787 der Japan Airlines einen Rauchgeruch in der Kabine. Zeitgleich wurde festgestellt, dass sich die APU abgeschaltet hatte. Es stellte sich heraus, dass die für den Start der APU zuständige Batterie im hinteren Elektronik-Kompartment stark überhitzt war und Rauch ausstoß. Nach Beendigung der Löscharbeiten wurde festgestellt, dass 7 von 8 vorhandenen Lithium-Ionen-Zellen einen internen Kurzschluss erlitten hatten.

Die von Boeing eingeführten Sicherheitsmaßnamen zielten Laut FAA hauptsächlich auf eine Verhinderung von Überspannung und Überladung ab, da dies die Fehlerfälle seien, bei denen es zu Gas- und Flüssigkeitsaustritt und Flammenbildung kommen könne. Obwohl eine abschließende Beurteilung des Vorfalls noch nicht erfolgt ist, so gibt es Hinweise, dass die Bildung von Dendriten zu einem internen Kurzschluss in einer der Zellen geführt hat [4].

Als Dendrit wird eine baumartig wachsende Struktur eines Kristalls bezeichnet. Wächst eine solche Struktur aus dem Kristallgitter der Kathode eines Li-Ionen-Akkus und durchdringt den Separator, so kann dies zu einem internen Kurzchluss der Elektroden führen [5].

Am 16. Januar entschied sich die Besatzung einer Boeing 787 der Al-Nippon-Airways zu einer Notlandung, nachdem mehrere Warnungen vom Batterie-Management-System im Cockpit eingetroffen waren und ein Rauchgeruch in der Kabine wahrzunehmen war. Nach der Notlandung wurde festgestellt, dass die Hauptbatterie ausgebrannt war. Da das Flugzeug anders als die am Boden stehende Maschine der Japan Airlines nicht auf die Energieversorgung durch die Batterie angewiesen war, griffen die automatischen Sicherheitssysteme. Im Falle von Rauch im E&E-Kompartment (Electrical-Equipment-Kompartment) steuern sie die Belüftungsanlage so, dass der Rauch aus dem Flugzeug geleitet wird. So kam niemand bei dem Vorfall zu Schaden. Die ebenfalls noch nicht abgeschlossenen Ursachenermittlungen zeigen, dass es zu einer Überhitzung in der Zelle 6 der Hauptbatterie kam, deren Sicherheitsventil daraufhin zerbrach, um den Überdruck im Inneren abzubauen und eine Explosion zu verhindern. Austretendes Elektrolyt hat daraufhin zu einem Kurzschluss zwischen einem Zellverbinder und der, mit der Flugzeugmasse verbundenen, Zellwand geführt, wodurch die übrigen Zellen durch die auftretenden großen Ströme beschädigt wurden.

In den folgenden Abschnitten soll dargelegt werden, wie die genannten Ereignisse mit den in dieser Arbeit behandelten Aufgaben in Verbindung stehen.

1.2 Sicherheit von Lithium-Ionen Batterien in Flugzeugen

Es existieren zwei Gesichtspunkte, um Vorfällen wie die im vergangenen Kapitel genannten Batteriebrände in der Boeing 787 ihre Brisanz zu nehmen:

- Wie ist es möglich, zukünftig das Auftreten von schädlichen Veränderungen der Batteriezellen besser vorherzusagen?
- Wie können die Auswirkungen eines Zellkurzschlusses vermindert werden?

Die Batterien der Boeing 787 sind mit einem Batterie Managment Unit (BMU) System ausgestattet. Dieses besteht aus 4 redundanten Subsystemen, welche die Zellspannungen, die Batteriespannung, den Batteriestrom sowie die Temperatur der Batterie aufnimmt und auswertet. Im Fehlerfall hat eines der Subsysteme die Möglichkeit die Batterie durch Trennen des Rückleiters stromlos zu schalten. Diese BMU kommuniziert mit den Sicherheitssystemen des Flugzeuges, um so den Piloten im Fehlerfall zu informieren. Abbildung 1.2 zeigt die Batterie mit dem BMU. Gut zu erkennen ist, dass jede Zelle einzeln per Messleitung überwacht wird. Auch zu erkennen ist, dass je zwei der Subsysteme der BMU auf einem PCB untergebracht sind und dass jedes PCB unabhängige Messleitungen für die Zellspannungen besitzt. Schlecht zu erkennen ist der Hall-Sensor im blauen Gehäuse für die Strommessung im unteren Bereich der Batterie.

Trotz dieses Sicherheitssystems, das sogar die Möglichkeit besitzt die Batterie zu trennen, kam es zu den beschriebenen Vorfällen. Dies liegt daran, dass man aus einer einfachen Messung von Batteriestrom und Zellspannung nur schwer Rückschlüsse auf eine Änderung der Kristallstruktur in einer der Zellen schließen kann. Zwar wird sich eine Veränderung der Kristallstruktur auf den Innenwiderstand der Zelle auswirken, allerdings ist über die vorhandenen Mittel nur eine Messung des absoluten Innenwiderstandes bei einer nicht zu bestimmenden Frequenzmischung möglich. Wie jedoch zum Beispiel in [6] gezeigt, ist die Impedanz zum einen eine komplexe Größe, zum anderen in bedeutendem Maße frequenzabhängig. Die Elektrochemische Impedanz Spektroskopie (EIS) bietet eine Möglichkeit zur Messung der komplexen und frequenzabhängigen Impedanz. Bei dieser lassen sich verschiedene Vorgänge in den Batteriezellen verschiedenen Messfrequenzen zuordnen, sodass hier viel deutlicher eine Veränderung des Innenwiderstandes durch Umbildung der Kristallstruktur zu erwarten ist. In [7] wurde nachgewiesen, dass es prinzipielle Möglichkeiten gibt eine EIS auch in situ durchzuführen.

Auch mit verbesserter Diagnosefunktion ist ein Fehlerfall mit Kurzschluss in einer Zelle nicht auszuschließen. Das Problem lässt sich weiterhin nicht durch Freischalten der Batterie lösen, da die thermische Zersetzung der Zelle von innen heraus weiter stattfindet. Dieser Vorgang wird thermisches Durchgehen genannt. Dies tritt auf, da die eingesetzten Akkumulatoren Lithium-Cobalt-Oxid als Kathodenmaterial verwenden, das bei Hitzeeinwirkung Sauer-

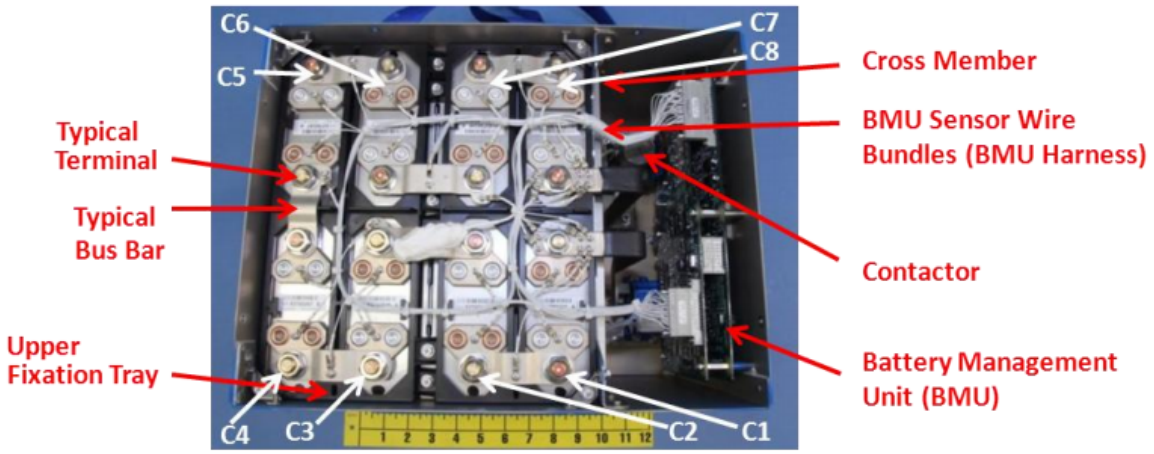


Abbildung 1.2: Boeing 787 Batterie mit Batterie Managment Unit (BMU) entnommen aus [3]. C1 bis C8 markiert die einzelnen Zellen.

stoff freisetzt, was zu zusätzlicher Oxidation und damit steigender Wärmeentwicklung führen kann. Schon zur Zeit der Entwicklung der Boeing 787 ab dem Jahr 2003 standen alternative Kathodenmaterialien zur Verfügung, die bei Hitzeeinwirkung keinen Sauerstoff freisetzen und ohne toxische Bestandteile wie Cobalt auskommen, beispielsweise Lithium-Eisenphosphat. Auch bei diesem Akkumulatortyp ist ein ‚Thermal Runaway' nicht ausgeschlossen, aber seine Folgen sind deutlich weniger gefährlich. Allerdings waren diese Materialien erst seit Kurzem auf dem Markt verfügbar und somit die Erfahrung mit diesem schwerer einzuschätzen als mit Lithium-Cobalt-Oxid. Da die Tendenz in der Luftfahrt dahin geht, ausgereifte Techniken einzusetzen, hat man sich zu dem Zeitpunkt auf das eingesetzte Kathodenmaterial geeinigt. Mit Blick auf die aufgetretenen Vorfälle stellt sich die Frage, ob Batterien aus anderen Materialien in Zukunft den Vorrang erhalten sollten.

1.3 Forschungsfragen

Aus den im vergangenen Abschnitt dargelegten Erkenntnissen lassen sich eine Reihe von Forschungsfragen ableiten, die bereits Teil verschiedener Veröffentlichungen sind und in dieser Arbeit betrachtet werden sollen.

- Kann ein Akkumulator auf Basis von Lithium-Interkalation mit Eisen-Phosphat als Kathodenmaterial die aktuell verwendeten Akkumulatoren mit Cobalt-Oxid als Kathodenmaterial ersetzen?
- Vermindert ein Lithium-Eisenphosphat-Akkumulator die Auswirkungen eines Zellkurzschlusses?
- Inwieweit ist die Elektrochemische Impedanz Spektroskopie dazu geeignet, Veränderungen der Kristallstruktur einer Lithium-Ionen-Akkumulatorzelle zu erkennen?
- Ist es möglich und wirtschaftlich Elektrochemische Impedanz Spektroskopie in situ in einem Flugzeug durchzuführen?

Zur Beantwortung der ersten drei Fragestellungen muss auf Literaturarbeit zurückgegriffen werden. Es ist mit den im BATSEN-Labor zur Verfügung stehenden Mitteln nicht möglich den Brand eines Lithium-Eisenphosphat-Akkumulators zu simulieren oder sogar real zu vermessen. Genauso wenig lässt sich eine Batterie gezielt dahingehend verändern, dass sie voraussagbar Dendriten ausbildet, sodass die Veränderungen schrittweise vermessen werden können.

Lediglich der Nachweis einer Durchführbarkeit von EIS mit kostengünstigen Sensoren lässt sich im BATSEN-Labor gut realisieren.

1.4 Technische Aufgabenstellung

Angold hat in [6] dargestellt, warum die aktuelle Generation der BATSEN-Sensoren nicht in der Lage ist, für eine EIS eingesetzt zu werden. Aus seinen Erkenntnissen ergeben sich die notwendigen Voraussetzungen, um eine solche Messung erfolgreich in einem System durchzuführen, das später auch wirtschaftlich zusammen mit jeder Zelle betrieben werden kann.

- Aufnahme der Wechselspannungsanteile des EIS-Messsignales mit ausreichender Auflösung trotz des großen Gleichspannungsanteils für jede Zelle einzeln.
- Frequenz- und phasensynchrone Messung der Zellenspannung und des Batteriestromes.
- Auswertung der Messergebnisse im Frequenzbereich auf dem Batteriesteuergerät.
- Kosten verhältnismäßig klein im Vergleich zu Batteriekosten.

1.5 Zusammenfassung

In dieser Arbeit soll eine Abschätzung der Anforderungen geschehen, die eine aufwandsreduzierte Elektrochemische Impedanzspektroskopie an einen drahtlosen Zellsensor stellt. Ein solcher Sensor soll auf Basis der Arbeiten aus [8] und [9] entwickelt und realisiert werden. Im Anschluss soll mit diesem neu entwickelten Sensor exemplarisch die Durchführbarkeit einer aufwandsreduzierten Elektrochemischen Impedanzspektroskopie nachgewiesen werden. Eine detailliertere Auflistung der einzelnen Komponenten der Aufgabenstellung ist in Anhang A angefügt.

2 Stand der Forschung und Technik

In diesem Abschnitt sollen relevante Grundlagenkenntnisse vermittelt sowie ein genauerer Überblick über den Stand der Technik von bereits geleisteten Vorarbeiten aus dem BATSEN-Projekt gegeben werden. Zunächst wird auf die Technik der Lithium-Eisenphosphat Akkumulatoren eingegangen. Im Anschluss soll die Theorie der Elektrochemischen Impedanzspektroskopie erläutert werden. Abschließend werden die drahtlosen Zellsensoren der Klasse 3 von Durdaut [8] und Sassano [9] vorgestellt, die als Grundlage für den in dieser Arbeit zu erstellenden Sensor dienen.

2.1 Aufbau eines Lithium-Eisenphosphat Akkumulators

Es existieren eine Reihe von Akkumulatortypen, die auf Grundlage der Interkalation von Lithium-Ionen in ein Kristallgitter arbeiten. Für sie alle ist der Begriff Lithium-Ionen-Akkumulator (Li-Ion-Akku) zutreffend. Heute wird unter dieser Bezeichnung jedoch meist ein Akkumulator mit Lithium-Cobalt(III)-oxid ($LiCoO_2$) als Anodenmaterial gemeint, da diese die ersten kommerziell verfügbaren LiIon-Akku waren [10]. Heute sind andere Stoffe als Anodenmaterial bekannt, wodurch sich die Akkumulatortypen an ihren Einsatzbereich anpassen lassen.

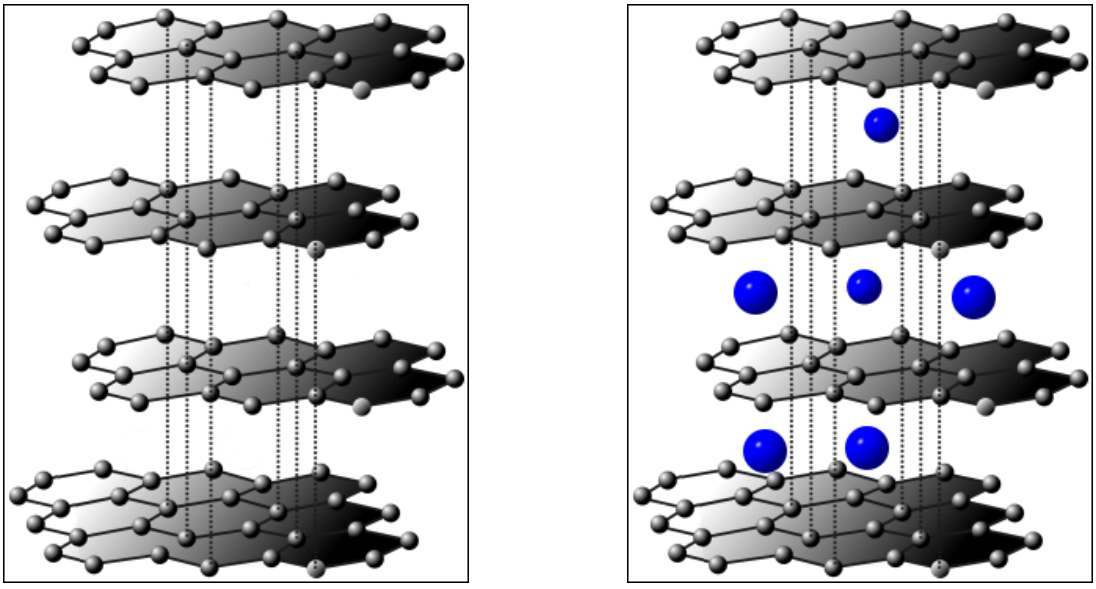
Beim Lithium-Eisenphosphat-Akkumulator ($LiFePO_4$-Akku) dient Lithium-Eisenphosphat als Anodenmaterial und wie bei allen anderen LiIon-Akkus eine Graphitelektrode als Kathode[1].

[1] Die Bezeichnung der Elektroden bezieht sich hier und in der gesamten Arbeit, wenn nicht anders angegeben, auf die Entladung der Zelle. Beim Laden der Zelle tauschen die Elektroden funktional ihre Bezeichnung.

2.1.1 Chemischer Aufbau

Graphit-Elektrode

Die Graphitelektrode ist aus gestapelten sogenannten Graphenschichten zusammengesetzt. Beim Graphen handelt es sich um eine 2-dimensionale Modifikation des Kohlenstoffes, bei dem jedes Kohlenstoffatom in einem Winkel von je 120° in einer Ebene mit drei anderen Kohlenstoffatomen verbunden ist. Da Kohlenstoff vierwertig ist, kommt es durch Bindung des übrigen Elektrons mit einem aus der darüber liegenden Schicht zu einer 3-dimensionalen Struktur aus Graphenschichten. Eine solche Struktur ist in Abbildung 2.1a dargestellt. Dabei sind die Wechselwirkungskräfte innerhalb der Schichten um Größenordnungen stärker als zwischen den Schichten. Die nicht an der Bildung der Schichten beteiligten Elektronen haben eine Aufenthaltswahrscheinlichkeitsfunktion, die sich hantelförmig senkrecht zu den Graphitschichten erstreckt, die sogenannten p-Orbitale. Mit diesen können positiv geladene Ionen, die in das Kristallgitter eingebracht werden, in Wechselwirkung treten und so stabil in das Kristallgitter interkaliert (d.h. eingelagert) werden.



(a) Graphen im ungeladenen Zustand.

(b) Interkalierte Lithium-Ionen an den Orten der p-Orbitale der Graphenschichten.

Abbildung 2.1: Graphit aus Graphenschichten wie es als Kathodenmaterial in $LiFePO_4$-Akkumulatoren zum Einsatz kommt. Entnommen und teilweise modifiziert aus [11]

Separator und Elektrolyt

Zwischen den Elektroden befindet sich ein elektrisch isolierender Separator, der für Lithium-Ionen durchlässig ist. Im Falle der in dieser Arbeit besprochenen Zellen der Firma ECC Repenning, besteht der Separator aus je einer äußeren Schicht Polypropylen und einer inneren Schicht Polyethylen. Der Separator wird von einem in Lösungsmittel gelösten Elektrolyt umgeben, das für die elektrische Verbindung zwischen den Elektroden durch Ionen-Transport sorgt.

Die in dieser Arbeit besprochenen $LiFePO_4$-Akkumulatoren der Firma Repenning verwenden einen nicht näher bezeichneten Elektrolyten aus Lithium-Salz und organischen Carbonaten [12]. Die ebenfalls in dieser Arbeit besprochenen Zellen der Firma A123 Systems [13] gebrauchen als Elektrolyten Lithiumhexafluorophosphat ($LiPF_6$), das sich in einem Lösungsmittel in Li^+ und ${PF_6}^-$ aufteilt. Der Elektrolyt darf nicht in einer wässrige Lösung gelöst sein, da Lithium mit Wasser stark exotherm reagiert, daher muss ein aproptisches Lösungsmittel benutzt werden.

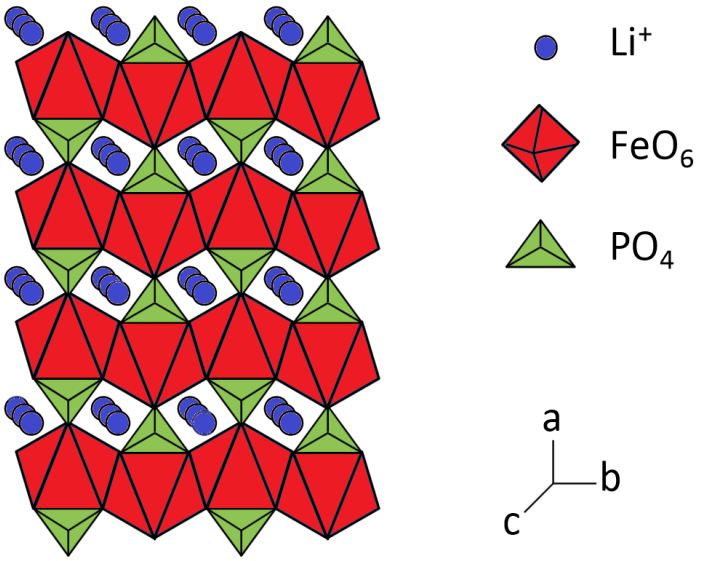


Abbildung 2.2: Kristallstruktur des Eisenphosphates einer $LiFePO_4$-Zelle. Darstellung verändert nach [16].

Eisenphosphat-Elektrode

Das Kathodenmaterial Eisen-Phosphat liegt in einer sog. Olivinschicht vor [14]. Das bedeutet, dass der Phosphor von vier Sauerstoffatomen so umgeben ist, dass ein Tetraeder mit der Summenformel PO_4^{3-} (Phosphatanion) entsteht. Jedes Eisenatom ist von sechs dieser Sauerstoffatomen in einer Oktaederform umschlossen (FeO_6) (vgl. [15]). Dabei sind mehrere Oktaeder in zu einer Reihe zusammengeschlossen. Diese Reihen werden von den Tetraedern der Phosphor-Sauerstoffverbindung wie in Abbildung 2.2 gezeigt zu einer Ebene verbunden. Aus solchen Ebenen setzt sich eine dreidimensionale Struktur zusammen, die zwischen den Phosphor-Sauerstoffverbindungen Röhren aufweist, in denen Lithium interkaliert werden kann. Dabei besteht das Kathodenmaterial produktionsbedingt nicht aus einer homogenen Kristallstruktur, sondern vielmehr aus Partikeln der Kristallsruktur, die nebeneinander und in Schichten auf eine Folie aufgebracht werden. Dabei kann die Größe der Partikel Einfluss auf die Elektrochemie haben, wie in Abschnitt 2.1.4 auf Seite 30 gezeigt wird.

Durch die Bindung der Sauerstoffatome an das Phosphor-Atom neigt Lithium-Eisenphosphat anders als andere Kathodenmaterialien von Lithium-Interkalations Akkumulatoren nicht dazu bei zu hohen Temperaturen Sauerstoff frei zu setzen [17]. Eine solche Freisetzung von Sauerstoff führt zum sogenannten 'Thermal Runaway', da der freiwerdende Sauerstoff für exotherme Reaktion mit dem Elektrolyten zur Verfügung steht, wodurch zusätzliche Wärme produziert wird, die wiederum mehr Sauerstoff freisetzt. Das bedeutet, dass LiFePO Akkumulatoren zwar nicht besser gegen Überstromereignisse oder innere Kurzschlüße geschützt sind, als andere Lithium-Interkalations Akkumulatoren, dass aber die Gefahr, die durch ein solches Ereignis ausgelöst wird, verringert werden kann.

2.1.2 Elektrochemie

Im vollständig entladenen Zustand befinden sich keine Lithium-Atome im Gitter der Graphitelektrode. Durch anlegen eines Äußeren Feldes können Lithium-Atome im Lithium-Eisenphosphat Gitter ein Elektron abgeben und als Lithium-Ion in den Elektrolyten eintreten. Dabei tritt gleichzeitig ein Lithium-Ion aus dem Elektrolyten in das Gitter der Graphitelektrode ein und rekombiniert dort mit einem Elektron, das über den Außenleiter von der Kathode zur Anode gelangt ist. Dadurch entsteht eine erhöhte Lithium-Konzentration am Rand der Eisen-Phosphat Elektrode und eine niedrigere Lithium-Konzentration an der Grenze der Graphitelektrode. Durch diese Konzentrationsdifferenz entsteht durch Diffussion ein Ionenstrom innerhalb des Elektrolyten.

Innerhalb der Graphitelektrode werden die Lithium-Atome in den senkrecht zu den hexagonalen Graphenschichten stehenden p-Orbitalen eingelagert und so in die Graphitschicht interkaliert zu werden (siehe Abbildung 2.1b, vgl. [18]). Wird nun die äußere Spannung entfernt, indem die Batterie vom Ladegerät getrennt wird, so kann das re-kombinierte Lithium den Separator nicht mehr passieren, sodass die eingespeiste Energie erhalten bleibt, bis eine äußere Last angeschlossen wird. Über diese können die Elektronen abfließen, während die nun entstehenden Lithium-Ionen den Separator passieren.

Beim Entladen stellt sich entsprechend ein Ionenstrom durch den Elektrolyten von der Graphit-Elektrode zur Eisen-Phosphat-Elektrode ein, während die Elektronen über den Außenleiter und die angeschlossene Last von der Anode zur Kathode fließen. Dieser Vorgang ist in Abbildung 2.3 auf der nächsten Seite dargestellt.

Innerhalb des Eisen-Phosphat Materials der Kathode werden die Lithium-Atome entlang eindimensionaler Kanäle innerhalb der einzelnen Partikel des Materials eingelagert. Diese Kanäle entstehen durch die oben beschriebene Olivin-Struktur und führen zu einem Effekt, der in Abschnitt 2.1.4 auf Seite 30 näher beschrieben ist.

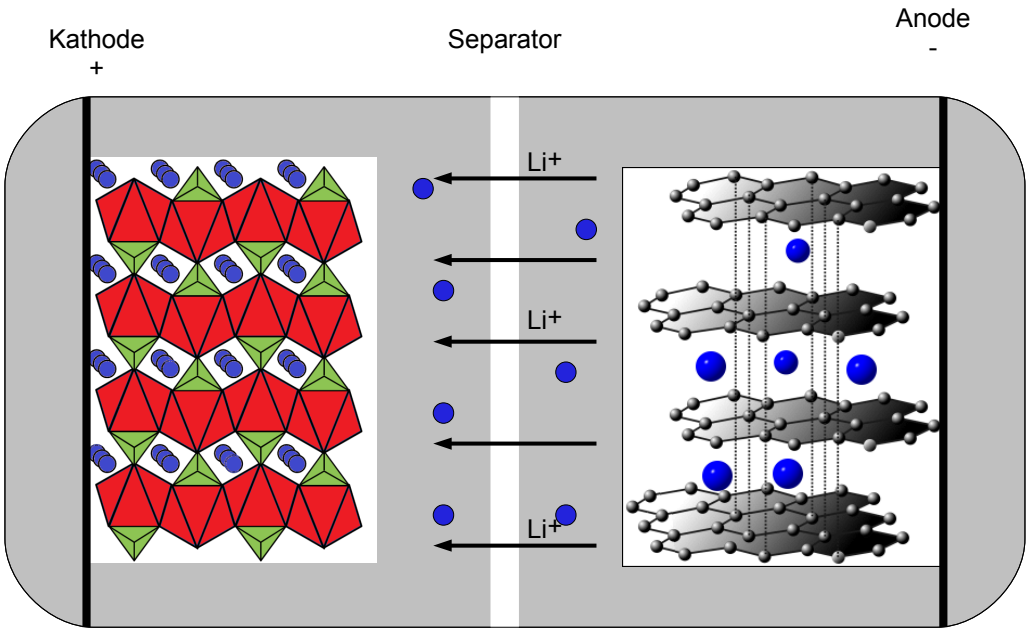


Abbildung 2.3: Ionentransport bei der Entladung eines LiFePo4 Akkumulators. Elektronen wandern von der Anode durch den Verbraucher zur Kathode. Gleichzeitig können Lithium-Ionen aus der Graphitelektrode durch den Separator in die Eisenphosphat-Elektrode diffundieren.

2.1.3 Technischer Aufbau

Abbildung 2.4 auf der nächsten Seite zeigt den inneren Aufbau einer $LiFePO_4$ - Zelle.

Die Graphitschichten werden dabei auf eine Kupferfolie aufgebracht, die Eisenphosphat-Kristalle auf einer Aluminium-Folie. Diese Folien dienen gleichsam der elektrischen Anbindung an den Zellen-Polen. Dazu wird zwischen beiden Folien eine Separator-Folie aus Polypropylen und Polyethylen gelegt, sodass auf je einer Seite eine der beiden Elektrodenfolien über den Separator hinausragt, die andere nicht. Eine weitere Separator-Folie wird hinter die Kupferfolie gelegt, damit das so entstandene Folienpaket zu einem Zylinder aufgewickelt werden kann, ohne dass es zu einer Verbindung von Anode und Kathode kommen kann. Der Zylinder wird in einem Metallgehäuse untergebracht, das anschließend mit dem Elektrolyten befüllt wird.

Zur elektrischen Anbindung wird die Aluminium-Folie mit dem Aluminum-Gehäuse und die Kupferfolie mit einem elektrisch vom Gehäuse isoliertem Anschluss im Deckel verbunden. Das Gehäuse, welches als Grundlage für die Abmessungsbestimmungen der Zellsensoren der Klasse-3 im BATSEN-Projekt dient, ist in Abbildung 2.5 auf der nächsten Seite dargestellt.

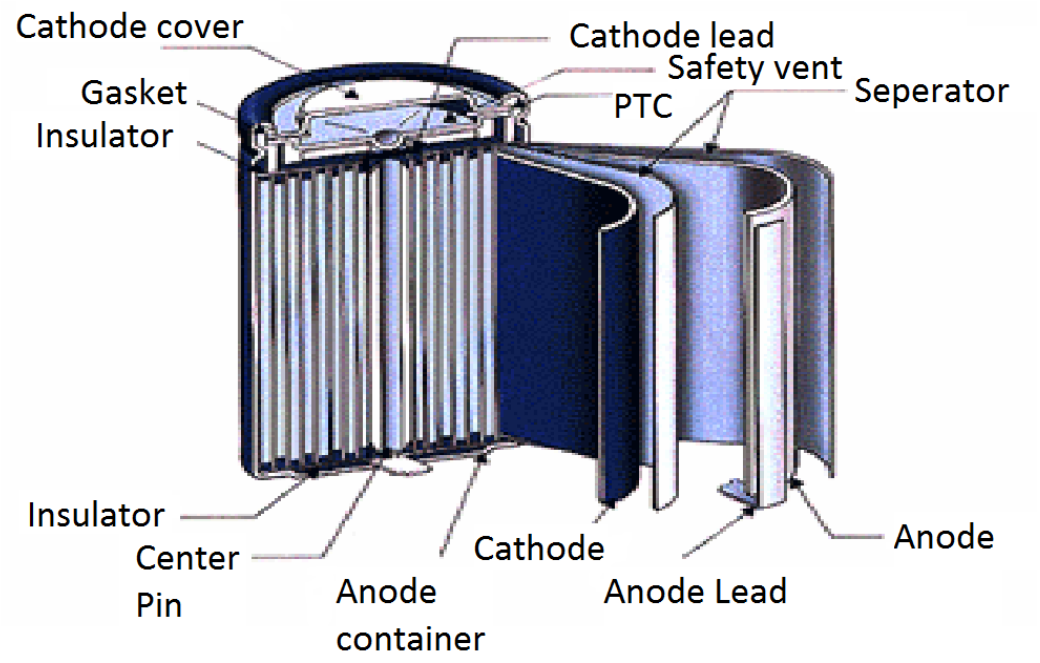


Abbildung 2.4: Technischer Aufbau einer Lithium-Eisenphosphat-Zelle nach [12]

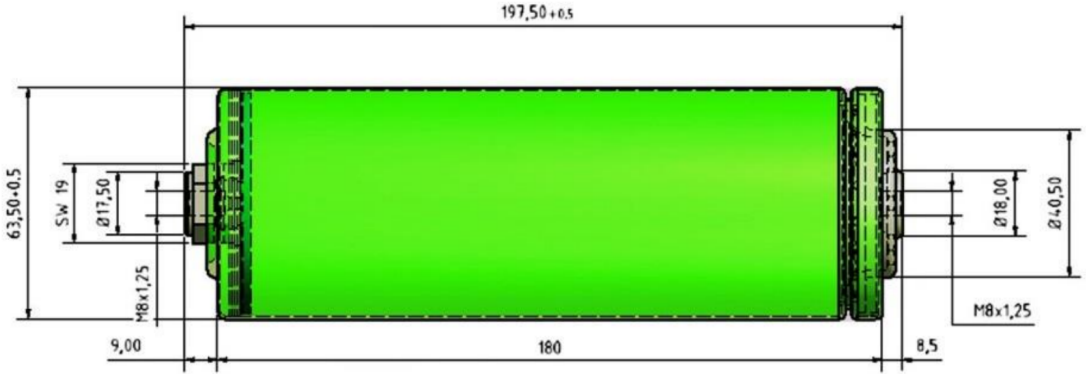


Abbildung 2.5: Abmessung der 42 Ah $LiFePO_4$-Akkummulatorzellen der Firma Repenning nach [8]

2.1.4 Eigenschaften und Modellierung

Für die Betrachtung der elektrochemischen Eigenschaften von Lithium-Eisenphosphat Akkumulatoren kommen wie bei jeder System-Analyse zwei Ansätze in Frage: eine Top-Down Betrachtung oder eine Bottom-Up-Betrachtung. Bei letzterer wird auf der kleinsten für die Eigenschaften relevanten Ebene mit der Analyse des Systems begonnen. Im Falle eines Lithium-Interkalations-Akkumulators also etwa auf der Ebene der am Ladungstransport beteiligten Elektronen oder den Atomen, die sich im Kristallgitter befinden oder als Ionen im Elektrolyten vorhanden sind.

In der Literatur, die sich im Speziellen mit der Modellierung von Lithium-Eisenphosphat Akkumulatoren befasst, kommen vornehmlich mehrdimensionale Simulationen zum Einsatz, welche diesen Bottom-Up Ansatz verfolgen und die physikalischen und chemischen Vorgänge innerhalb der Zelle betrachten. Als Beispiele seien das Modellierungswerkzeug 'detailed electrochemistry and numerical impedance simulation (DENIS)' der Universität Heidelberg [19] oder die 'Mikrostrukturmodellierung von Lithiumbatterie-Elektroden' des Karlsruher Institut für Technologie [20] genannt. Für eine Implementierung in einem in-Situ Sensor sind solche sehr rechenintensiven Modelle aktuell noch unattraktiv. Im Gegensatz dazu bieten Ersatzschaltbildmodelle eine Näherung des von außen messbaren Verhaltens der Zelle ab, was dem Top-Down Ansatz entspricht. Hier wird eher das dynamische Verhalten eines Systems von außen betrachtet. Über diese Modelle lassen sich keine Rückschlüsse auf physikalische Vorgänge innerhalb der Batterie machen [21], entsprechend lässt sich auch die Bildung von Dendriten nicht nachzuvollziehen [22].

Für die Bestimmung von SoC und SoH sind Ersatzschaltbild-Modelle ausreichend geeignet.

Im Folgenden werden verschiedene Modelierungsansätze aus beiden Ansatzrichtungen vorgestellt. Dabei ist das Ziel nicht, die Verwendung dieser Modelle innerhalb dieser Arbeit. Stattdessen soll herausgestellt werden, welche Parameter durch eine Elektrochemische Impedanzspektroskopie ermittelt werden können, um mit diesen Erkenntnissen ein Anforderungsprofil für die eigene EIS-Messung zu erstellen.

Zwei-phasentransformation

Abbildung 2.6 auf der nächsten Seite zeigt, dass im Vergleich zu anderen Akkumulator-Technologien der Ruhespannungsverlauf über der Entladetiefe sehr flach ist. Es zeigt sich aber auch, dass es an den Randbereichen der Entladung, also bei fast vollständig geladener oder fast vollständig entladener Zelle, zu Abweichungen vom ansonsten fast linearen Verlauf kommt. Die Ursache für dieses Verhalten ist in der Art begründet, wie Lithium im Kristallgitter der Eisen-Phosphat Elektrode eingelagert wird.

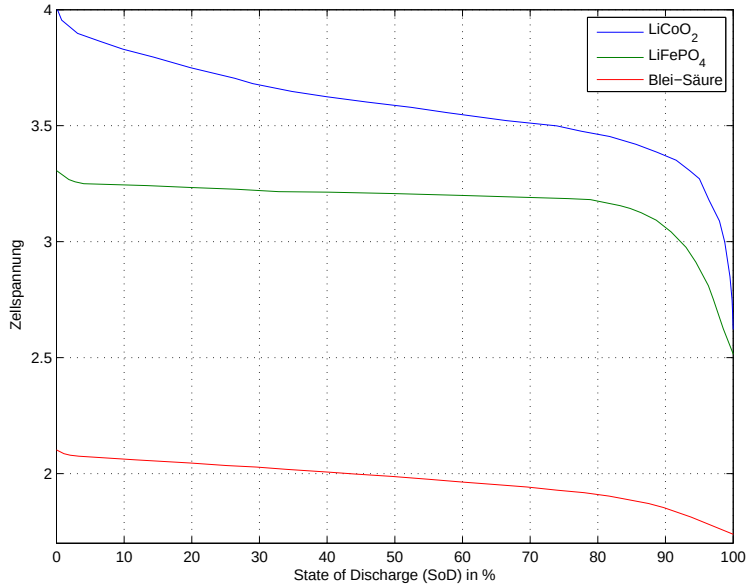
Wie in Abschnitt 2.1.1 auf Seite 23 beschrieben, findet die Interkalation der Lithium Ionen in den Kanälen des Kristalgitters statt. Dabei kommt es nicht zu einer homogenen Einlagerung der Ionen in die Kanäle [24]. Stattdessen existieren sog. Grenzkonzentrationen, bei denen die Anziehungskräfte zwischen Wirtsgitter und eingelagerten Lithium-Ionen zu energetisch günstigen Zuständen führen. Eine weitere Einlagerung von Lithium-Ionen geschieht nur durch den Übergang eines Bereiches mit niedrigerer Grenzkonzentration zur höheren Grenzkonzentration. Für Lithium-Eisenphosphat werden in der Literatur übereinstimmend zwei Grenzkonzentrationen angegeben, deren absoluter Wert aber unterschiedlich angegeben wird [25], [26].

Ausgehend von einem anfangs völlig verarmten Kristallgitter, also einer Lithiumkonzentration nahe 0, geschieht bei der Interkalation nur Folgendes: Bis zum Erreichen der unteren Grenzkonzentration verändert sich die Lithiumkonzentration homogen. Danach bilden sich nach [27] an den Oberflächen ausreichend großer Partikel des Elektrodenmaterials Phasen mit der höheren Grenzkonzentration. Diese 'wachsen' mit zunehmender Lithium-Interkalation in die Partikel hinein. Es verbleibt ein Kern mit der niedrigeren Grenzenergie im inneren der Partikel, der mit zunehmender Lithium Einlagerung kleiner wird.

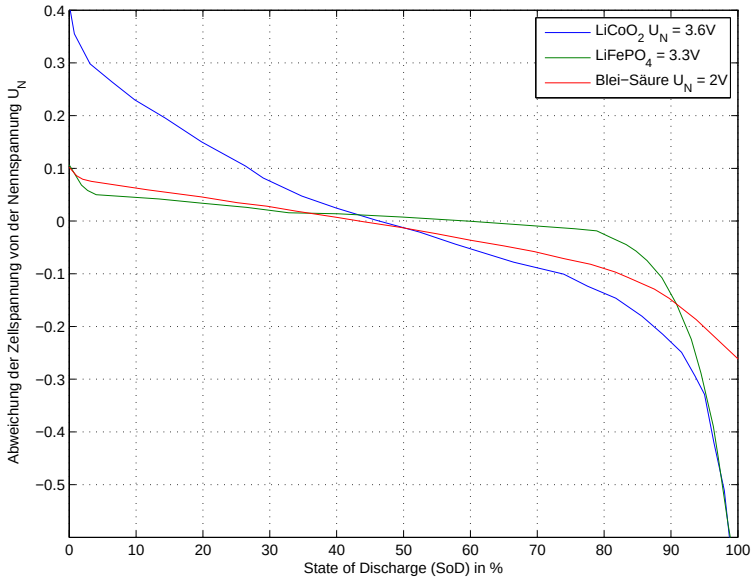
Ist ein Partikel vollständig mit der höheren Grenzkonzentration durchsetzt, so beginnt wieder eine Phase homogener Konzentrationssteigerung. Da für das Potential der Elektrode maßgeblich die Lithiumkonzentration an der Oberfläche der Partikel relevant ist, bleibt das Ruhespannungsniveau über einen weiten Bereich des SoC konstant, da in diesem Bereich das interkalierte Lithium ins Zentrum der Partikel wandert, ohne Einfluss auf die Konzentration an der Oberfläche der Partikel zu nehmen.

Abbildung 2.7 auf Seite 32 zeigt schematisch den Verlauf der Ruhespannung über den State of Charge sowie zusätzlich die Lithiumkonzentration in einem beispielhaften Partikel des Eisen-Phosphat-Materials, welches außerhalb der unteren und oberen Grenzkonzentration eine homogene Konzentrationsänderung erfährt und dazwischen die sog. Zwei-Phasentransformation durchläuft. Während dieser ist die Änderung der Ruhespannung über dem SoC noch deutlich flacher als in den Randbereichen.

In keinem Bereich steigt die Änderung der Ruhespannung über dem SoC über 2 $^{mV}/_{\%}$, wodurch eine zuverlässige Ladezustandserkennung über den SoC stark erschwert wird. Zwar



(a) Absoluter Verlauf der Zellenspannung



(b) Verlauf der Zellenspannung bezogen auf die Nennspannung

Abbildung 2.6: Beispielhafte Entladekurven verschiedener Batterietechnologien nach Daten aus [23]. Angegeben ist die Zellspannung, links auf einer absoluten Skala, rechts jeweils bezogen auf die Nennspannung. Deutlich zu erkennen ist der sehr flache Verlauf beim Lithium-Eisenphosphat-Akkumulator, außer an den Randbereichen.

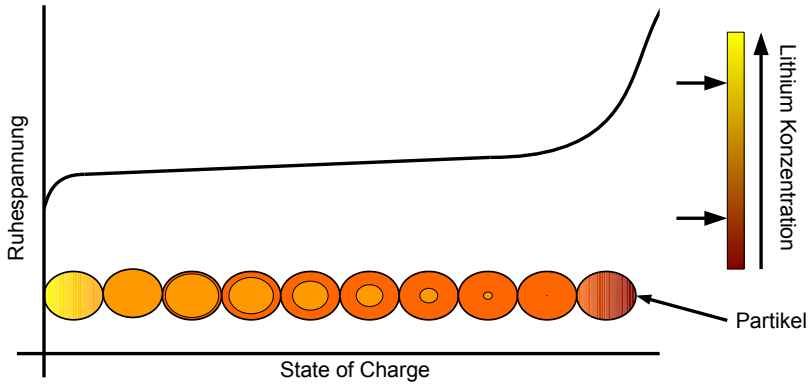


Abbildung 2.7: Verlauf der Ruhespannung über verschiedene Lithiumkonzentrationen im Kristalgitter. Auf der Farbskala eingezeichnet sind die zwei Grenzkonzentrationen.

sind Messungen so geringer Spannungsschwankungen möglich, jedoch dauert das Abklingen der dynamischen Spannungsänderung zur Ruhespannung nach einer Belastung bis zu mehreren Stunden, während dieser Zeit liegt die Abweichung von der Ruhespannung deutlich über 2 mV .

Aus der Anschauung der zwei-phasigen Entladung lässt sich ein weiteres Problem erklären, das die Ermittlung des SoC aus der Ruhespannung erschwert. In [22] wird dargestellt, dass selbst nach sehr langen Abklingzeiten, (so groß, dass die Selbstentladung der Zellen nicht mehr vernachlässigbar ist), Unterschiede in der Ruhespannung bestehen bleiben. Dabei besteht eine Abhängigkeit der Abweichung vom erreichten SoC und ob dieser durch Aufladen oder Entladen erreicht wurde.

Abbildung 2.8 auf der nächsten Seite zeigt, dass die Ruhespannung nach dem Entladen mehrere Millivolt unterhalb der Ruhespannung des entsprechenden SoC nach der Ladung liegt.

Dies ist bedingt durch die Lithiumkonzentration, die jeweils nach dem Laden und Entladen an den Oberflächen der Partikel zu finden ist. Dabei muss beachtet werden, dass das Eisen-Phosphat-Material beim Laden Lithium abgibt und beim Entladen aufnimmt.

Beim Entladen lagern sich das eintreffende Lithium an der Oberfläche der Partikel an. Es entsteht eine Phase mit der höheren Grenzkonzentration um einen Kern, der die niedrige Grenzkonzentration an Lithium enthält. Dieser Kern schrumpft mit zunehmender Ladung. Wird zu einem bestimmten Zeitpunkt das Entladen unterbrochen, so wird die Lithiumkonzentration in den Partikeln aufgrund der Grenzkonzentrationen nicht homogenisiert, sondern verbleibt in der Konstellation mit Kern und Mantel auf verschiedenen Konzentrationsstufen. Bei der Ladung bildet sich bei Erreichen der oberen Grenzkonzentration an den Oberflächen der Partikel eine Phase mit der niedrigen Grenzkonzentration, die mit zunehmender Ladung wächst und den Kern aus höher konzentriertem Lithium schrumpfen lässt.

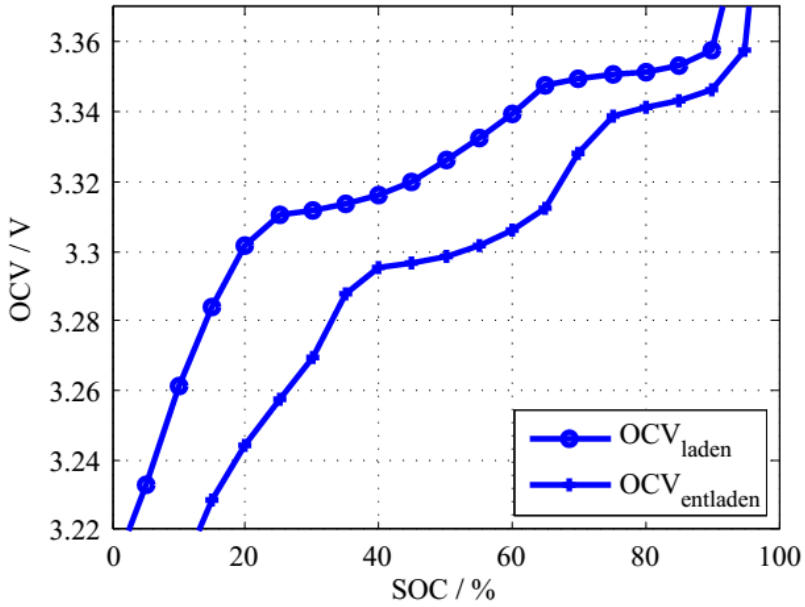


Abbildung 2.8: Hysteresse der Ruhepspannung einer $LiFePO_4$ Zelle in Abhängigkeit der Stromflussrichtung zum Erreichen eines Ladezustandes. Entnommen aus [22].

Stoppt man in diesem Fall die Entladung beim gleichen SoC wie oben die Ladung, so ist dieselbe Anzahl der Lithium-Atome in den Partikeln vorhanden, allerdings befindet sich nun eine Phase mit niedrig konzentrierten Lithium-Ionen an der Oberfläche. Dies stellt sich nach außen in einer niedrigeren Ruhespannung dar, da die Lithiumkonzentration an der Partikeloberfläche einen stärkeren Einfluss auf die Ruhespannung hat als die im Kern vorhandene. Der Vorgang von Laden und Entladen und die entsprechenden Konzentrationsverteilungen in einem Partikel sind in Abbildung 2.9 dargestellt.

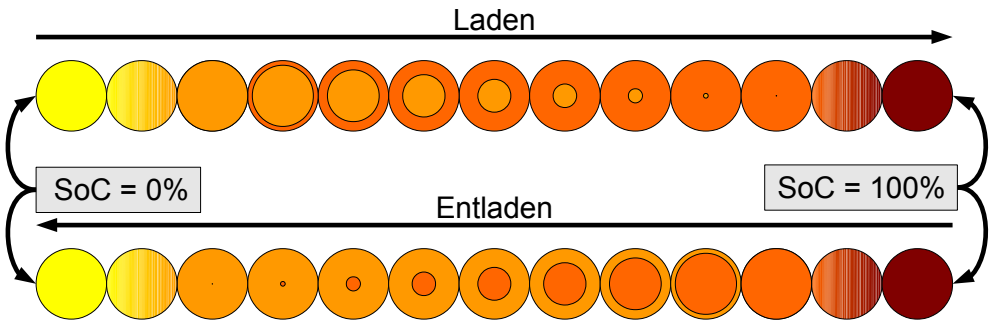


Abbildung 2.9: Lithium-Konzentrationsverteilung in einzelnem Eisen-Phosphatpartikel beim Laden und Entladen.

Ersatzschaltbilder

Die im letzten Abschnitt beschriebenen elektrochemischen Modelle liefern zwar aufgrund ihrer Nachbildung der realen Vorgänge in der Zelle Erklärungen für die beobachtbaren Phänomene, allerdings wäre für die Simulation einer vollständigen Batteriezelle der Rechenaufwand extrem hoch. Um modellbasiert die Veränderungen innerhalb einer Zelle, insbesondere den SoC und SoH, nachzuverfolgen, ohne enorme Rechenleistung zur Verfügung zu stellen, nutzt man daher elektrische Ersatzschaltbilder (ESB) für die einzelnen Zellen. Diese Art der Top-Down Modellbildung löst sich von der Grundlage der Zellchemie und betrachtet stattdessen das Verhalten einer Zelle auf eine bestimmte Anregung. Dieses Verhalten wird dann mit einer minimalen Anzahl an diskreten Bauelementen nachgebildet.

Für das Gleichstromverhalten gelten dabei die Gesetzmäßigkeiten des Thevenin- oder Norton-Therorems [28], die besagen, dass ein System aus einer beliebigen Anzahl von Widerständen, Spannungsquellen und Stromquellen entweder beschrieben werden kann als Spannungsquelle mit Reihenwiderstand oder Stromquelle mit Parallelwiderstand. Dabei existiert zwischen den Größen im Thevenin-ESB U_{Th} und R_{Th} und den Größen im Norton-ESB I_N und R_N der Zusammenhang:

$$R_{Th} = R_N$$
$$U_{Th} = I_N \cdot R_N$$

Abbildung 2.10 auf der nächsten Seite zeigt beispielhaft wie eine derartige Umwandlung aussieht.

Gleichzeitig kann an dieser Abbildung der große Nachteil der Methodik der Ersatzschaltbilder verdeutlicht werden. Im Bezug auf die Klemmen verhalten sich beide Ersatzschaltbilder vollkommen äquivalent, intern sind die Vorgänge aber unterschiedlich. Das Thevenin-ESB mit der Spannungsquelle setzt im Leerlauf keine Leistung um, da kein Strom fließt, dafür wird im Kurzschlussfall die Leistung $P_{Th,max} = {U_{Th}^2}/{R_{Th}}$ umgesetzt. Im Norton-ESB mit der Stromquelle wird im Leerlauf die maximale Leistung von $P_{N,max} = I_N^2 \cdot R_N$ umgesetzt, dafür bleibt das System im Kurzschlussfall ohne Verlustleistung. Obwohl also beide Modelle dem Top-Down Ansatz entsprechen, da sie das Klemmenverhalten korrekt abbilden, liefert das dennoch keine Aussage auf das interne Verhalten des beschriebenen Systems.

Im Wechselstromfall gilt analog dasselbe, wobei als Bauteile der zu vereinfachenden Systeme zusätzlich ideale Induktivitäten und ideale Kapazitäten zugelassen werden. Dann ergibt sich ein entsprechendes Ersatzschaltbild, das statt des realen Widerstandes R eine komplexe Impedanz $\underline{Z}$ als Serien-, bzw. Parallelimpedanz besitzt. Allerdings beschränken sich auch diese Modelle darauf, lineare Vorgänge zu modellieren. Da die meisten realen Vorgänge, wie

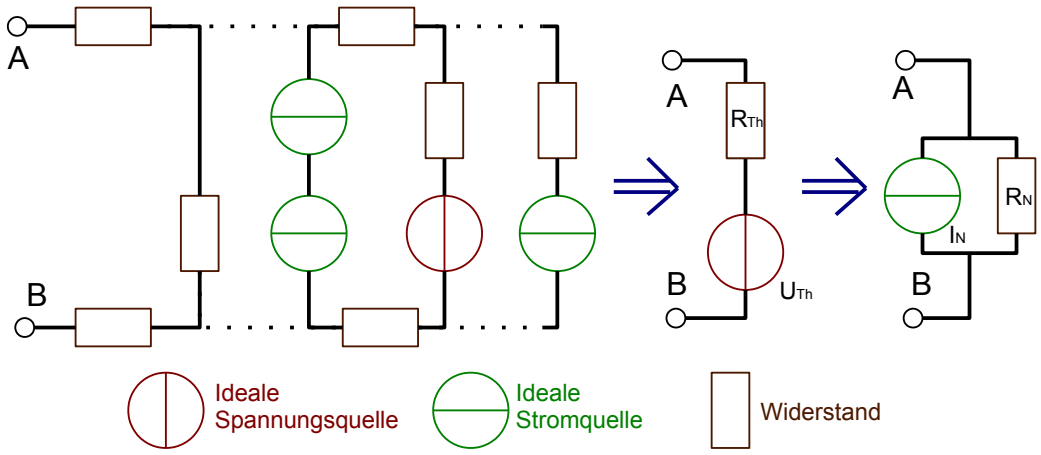


Abbildung 2.10: Transformation eines Gleichstromnetzwerkes mit vielen Elementen zu einem einfachen Netzwerk nach Thevenin oder Norton.

auch das Verhalten einer Batteriezelle, nicht-lineare Anteil haben, sind Modelle die sich auf diese einfachen Ersatzschaltbilder stützen, immer fehlerbehaftet.

Ersatzschaltbilder werden parametrisiert, sodass ihre Reaktion auf eine bestimmte Anregung der Reaktion des zu modellierenden Systems entspricht. Daraus folgt, dass ein Modell, welches bei einer Anregung korrekte Simulationsergebnisse liefert, bei stark abweichender Anregung nicht zwangsläufig auch sinnvolle Ergebnisse liefern muss. Bei der Erstellung eines Modells muss daher darauf geachtet werden, dass das erstellte Modell zur späteren Anwendung und der damit verbundenen Anregung passt.

Im Falle von Akkumulatoren kann eine erste einfache Fall Unterscheidung wie folgt aussehen.

- Im **Betriebsfall** versorgt eine Batterie einen oder mehrere Verbraucher oder wird selbst geladen und dient eventuell als Puffer vor den Verbrauchern. Dabei werden konstante oder nicht-deterministisch schwankende Ströme über Zeiträume im Bereich von Sekunden über Minuten bis Stunden abgegeben oder aufgenommen. Es findet eine relevante Ent- und Aufladung der Zelle statt, die Einfluss auf den Ladungsinhalt der Batterie und somit die Ruhespannung hat. Vorgänge innerhalb des Akkumulators mit Zeitkonstanten unterhalb der genannten Zeiträume sind für ein Modell das dieses Verhalten nachbilden soll, eher zweitrangig.
- Im Fall der **Messung des Kleinsignalverhaltens**, wie es bei einer EIS vorkommt, werden kleine Amplituden bei Frequenzen von wenigen mHz bis einigen kHz in das System eingeprägt. Hier spielen die Komponenten des Systems mit sehr kurzen Zeitkonstanten eine deutlich größere Rolle. Der SoC sollte während einer EIS-Messung

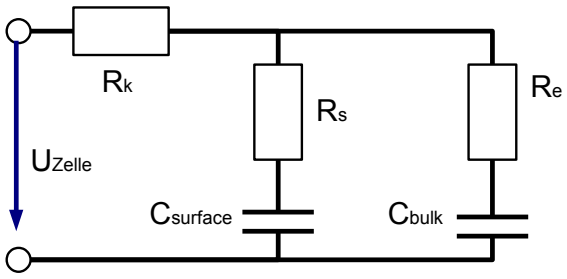


Abbildung 2.11: Einfaches Ersatzschaltbild einer Blei-Säure-Batterie nach [29].

nahezu konstant bleiben, weshalb die Ruhespannungsänderung im Modell eher untergeordnet zu realisieren ist.

Beide Ansätze verfolgen gegensätzliche Schwerpunkte, sodass es unwahrscheinlich erscheint, dass ein Modell gefunden werden kann, das beide Anregungsfälle gleichsam präzise beschreibt.

Innerhalb des BATSEN-Projektes hat sich Li [29] mit dem Modellieren von Blei-Säure-Batterien für die Implementation eines Kalman-Filters beschäftigt. Da dieses Filter genutzt werden sollte, um im laufenden Betrieb aus der dynamischen Zellspannung und dem Strom die Ruhespannung zu ermitteln, basieren seine Modelle auf dem Betriebsverhalten der Batterie und nicht auf dem Kleinsignalverhalten. Als einfache Näherung eines Batteriemodels dient dabei das in Abbildung 2.11 gezeigte Batteriemodell. Es basiert auf zwei Kapazitäten mit unterschiedlich großen Serienwiderständen. Die Kapazität C_{bulk} übersteigt dabei um Größenordnungen die Kapazität $C_{surface}$. Erstere stellt den Hauptenergiespeicher der Zelle dar. Sein Energieinhalt (und daher seine Spannung) ist die Zielgröße, die vom Kalman-Filter beobachtet werden soll. Alle drei Widerstände zusammen sind ursächlich für den Spannungseinbruch beim Entladen und die Spannungsüberhöhung beim Laden. Der zweite Kondensator beschreibt die abklingende Spannungsantwort nach einem Stromsprung.

Messungen zeigen, dass dieses Modell den realen Verlauf der Spannung bei einem Vorgegeben Strom nicht abbilden kann. Zum einen zeigt sich, dass die Spannungsüberhöhung beim Laden geringer ist als der Spannungsabfall beim Entladen und dass beide Vorgänge unterschiedliche Zeitkonstanten haben. Außerdem zeigt sich beim erstmaligen Entladen eine Spannungsüberhöhung, die auf einen induktiven Anteil hinweist. Daraufhin wurde das in Abbildung 2.11 gezeigte Modell entwickelt, das deutlich komplexer ist und nun auch Nichtlinearitäten, namentlich Dioden, enthält, die notwendig sind, um die unterschiedliche Reaktion in Abhängigkeit der Stromflussrichtung beim Laden und Entladen zu simulieren.

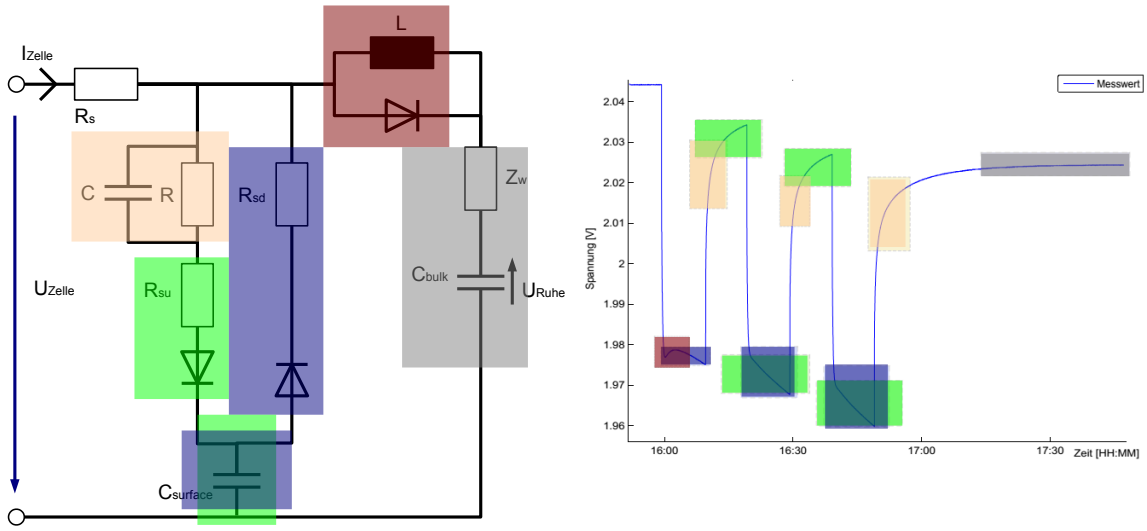


Abbildung 2.12: Ersatschaltbild für eine Blei-Säure-Batterie und Auswirkungen im Signal nach [29].

Da dieses komplexere Modell aufgrund der deutlichen Nichtlinearitäten in einem klassischen Kalman-Filter nicht zu implementieren ist, hat Li trotz der deutlichen Abweichungen bei der direkten Simulation das erste Modell gewählt. Ist der Modellfehler nicht zu groß, so kann ein Kalman-Filter die Abweichungen der Simulation von der Messung ausreichend gut ausgleichen, um dennoch eine Vorhersage des Energie-Inhaltes der Bulk-Kapazität treffen zu können. Seine Ergebnisse zeigen, dass eine plausible laufende Bestimmung der Ruhespannung im Betrieb möglich ist, auch wenn diese durch deutliche dynamische Spannungen überlagert wird.

In Abbildung 2.6 auf Seite 31 wurde gezeigt, dass der Verlauf der Zellspannung einer Blei-Säure Batterie über den Ladungsinhalt nahezu linear ist und lediglich kurz vor dem Bereich der vollständigen Entladung von diesem Verlauf abweicht. Da Blei-Säure-Batterien in der Praxis nicht derart tief entladen werden sollten, ist die Annäherung der Ruhespannung mit einer Kapazität zulässig. Bei dieser sinkt die Ruhespannung linear über die entnommene Ladung. Lithium-Interkalations-Akkumulatoren können, ohne übermäßig Schaden zu nehmen, über ihren gesamten SoC und Spannungsbereich verwendet werden. Dabei weisen sie neben einem sehr flachen Verlauf der Ruhespannung über den SoC einige deutliche Abweichungen in den Randbereichen auf, deren Ursprung, wie im letzten Abschnitt erklärt, in der Zwei-Phasentransformation des Lithiums im Eisen-Phosphat und Graphit-Gitter liegt. Daher ist ein Kondensator als Ladungsspeicher schlecht geeignet, um die Ruhespannung einer Zelle in einem Ersatzschaltbild darzustellen. Roscher [22] verwendete daher eine einstellbare Spannungsquelle für die Ruhespannung. Der Spannungswert dieser Quelle entsteht mathematisch aus dem durch das Modell bestimmten SoC und der im Vorfeld ermittelten

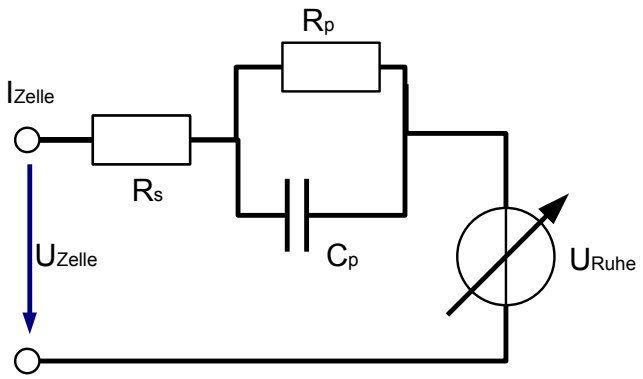


Abbildung 2.13: Ersatzschaltbild des Ladezustandsbeobachters nach Roscher [22].

Ruhespannungskurve (vgl: Abbildung 2.13). Für die Modellierung des dynamischen Verhaltens wird zum einen der serielle Klemmenwiderstand R_S und dazu in Reihe eine Kapazität C_p mit parallelgeschaltetem Widerstand R_p verwendet. Er stellt fest, das für ausreichend lange Ladungs- und Endladungszeiten dieses Modell brauchbare Ergebnisse liefert, wenn die Beschreibung der Ruhespannung über den durch Integration des Stroms gewonnenen Ladezustand ausreichend gut ist.

Roscher bestimmt für dieses Modell den Impedanzverlauf über verschiedene Frequenzen und ermittelt dabei, dass eine signifikante Abweichung im qualitativen und quantitativen Verlauf der Impedanz-Frequenzganges von typischen Lithium-Eisenphosphat-Batterien und dem von ihm vorgestellten Modell besteht (vgl. Abbildung 2.14 auf der nächsten Seite). Das Modell eignet sich, um den Betriebsfall darzustellen, nicht aber, um die Parameter für das Modell über eine EIS zu gewinnen.

Die Abbildung Abbildung 2.14 auf der nächsten Seite zeigt eine sogenannte Nyquist-Darstellung des Impedanzspektrums einer Batterie, die in Abschnitt 2.2 auf Seite 44 näher vorgestellt wird. Es ist hier der Imaginärteil über dem Realteil einer Messung in einem bestimmten Punkt aufgetragen. Man kann feststellen, dass die Ordinate invertiert dargestellt ist. Das ist bei Impedanzspektroskopien von Akkumulatoren üblich, da der Großteil des Verhaltens kapazitiven Charakter hat und sich deshalb bei negativen Imaginärteilen abspielt.

Aus der Abbildung geht nicht hervor, zu welcher Frequenz die einzelnen Messpunkte gehören. Die Darstellung beginnt bei kleinem Realteil und positivem Imaginärteil bei hohen Frequenzen im Bereich mehrerer kHz und endet bei großem Realteil und großem negativen Imaginärteil bei kleinen Frequenzen weniger mHz.

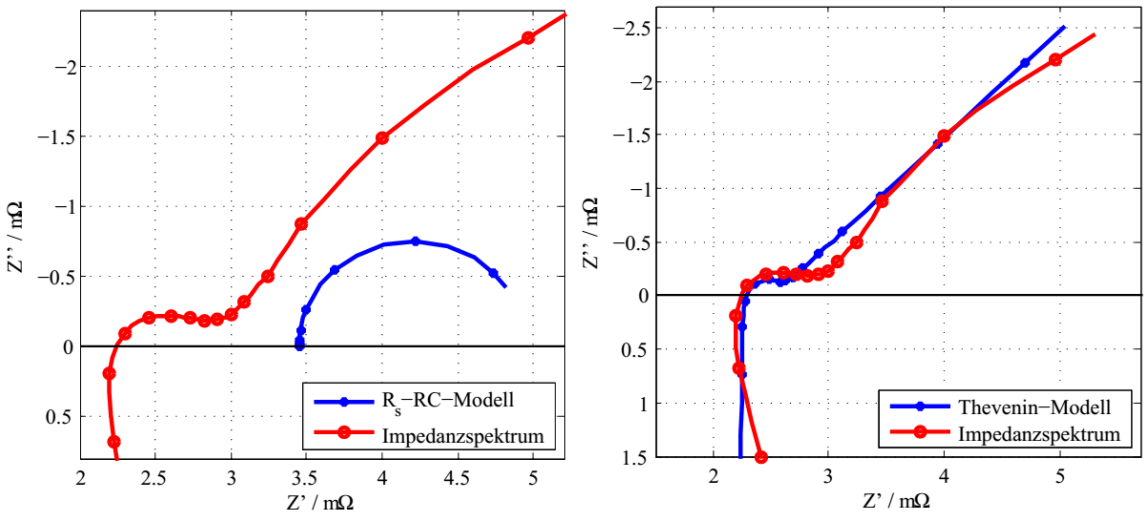


Abbildung 2.14: Impedanzspektren der verschiedenen Batteriemodelle nach Roscher. Links das Spektrum des RsRC-Modells aus Abbildung 2.13 auf der vorherigen Seite. Rechts das Spektrum des Thevenin-Modells mit ZARC-Element und Warburg-Impedanz aus Abbildung 2.16 auf Seite 41. Übernommen aus [22].

Man kann dem Bild entnehmen, dass bei hohen Frequenzen ein induktiver Anteil wirksam ist. Da die Kurve nicht bei 0 Ohm Realteil die Abszisse schneidet, muss auch ein dauerhaft wirksamer ohmscher Widerstand vorhanden sein. Es schließt sich ein bogenförmiger Verlauf an, wie er z.B bei der Parallelschaltung von Kapazität und Widerstand im ersten Modell nach Roscher entstanden ist. Abschließend bildet der Verlauf entweder eine Biegung hin zu einem linearen Verlauf ab oder einen zweiten Bogen mit sehr großem Radius.

Roscher entscheidet sich ein zweites Modell mit einem linearen Ausklingen des Verlaufs mittels Warburg-Impedanz zu realisieren. Außerdem verwendet er für den ersten Bogen ein sogenanntes ZARC-Element. Beide Elemente werden nun beide kurz vorgestellt.

Constant Phase Element Um ein ZARC-Element zu erhalten, benötigt man zunächst ein sogenanntes 'Constant Phase Element'. Dieses skaliert frequenzabhängig den Betrag des Eingangssignals mit Q und addiert eine über alle Frequenzen konstante Phase, die vom Exponenten n bestimmt wird. Für n = 1 entspricht die CPE einem idealen Kondensator. Die Übertragungsfunktion lautet:

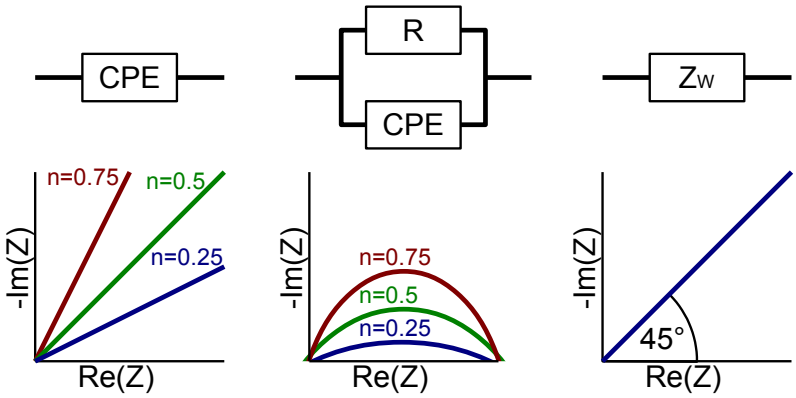


Abbildung 2.15: Impedanzspektren von CPE, ZARC und Warburg-Impedanz von links nach rechts. Für CPE und ZARC sind jeweils 3 Fälle dargestellt. Beim ZARC verlassen die Kurven den Nullpunkt unter den beim CPE ersichtlichen Winkeln.

$$
\begin{aligned}
H_{CPE} &= \frac{1}{Q(j\omega)^n} \\
H_{CPE}|_{n=1} &= \frac{1}{Q(j\omega)^1} \\
&= \frac{1}{Q(j\omega)} \qquad\qquad \Rightarrow \frac{1}{C(j\omega)}
\end{aligned}
$$

ZARC Das ZARC-Element entsteht durch das Parallelschalten eines CPE mit einem Widerstand. Die Bezeichnung rührt aus der Kombination von Z für die Impedanz des Widerstandes und ARC für die Bogenform des Ausgangssignals her. Für kleine Frequenzen schließt das CPE den Widerstand kurz und das Ausgangssignal liegt im Nullpunkt. Mit steigender Frequenz steigt der Widerstand des CPE und der parallele Widerstand R fällt mehr in Gewicht. Hin zu großen Frequenzen bildet das CPE einen Leerlauf und der Bogen endet auf der realen Achse beim Wert des parallelen Widerstandes. Eine Parallelschaltung von Kapazität und Widerstand erzeugt ebenfalls einen Bogen, der allerdings immer einen Halbkreis abbildet, da er mit 90 ° aus dem Nullpunkt startet. Durch Verwendung des CPE gewinnt man also einen Freiheitsgrad in der Darstellung.

Warburg-Impedanz Bei einer Warburg-Impedanz handelt es sich um den Sonderfall eines CPE mit $n = 0.5$. Also um einen frequenzabhängigen Widerstand, der dem Signal immer eine Phasendrehung von 45 ° hinzufügt. Die Impedanzspektren aller drei einzelnen Elemente sind in Abbildung 2.15 dargestellt.

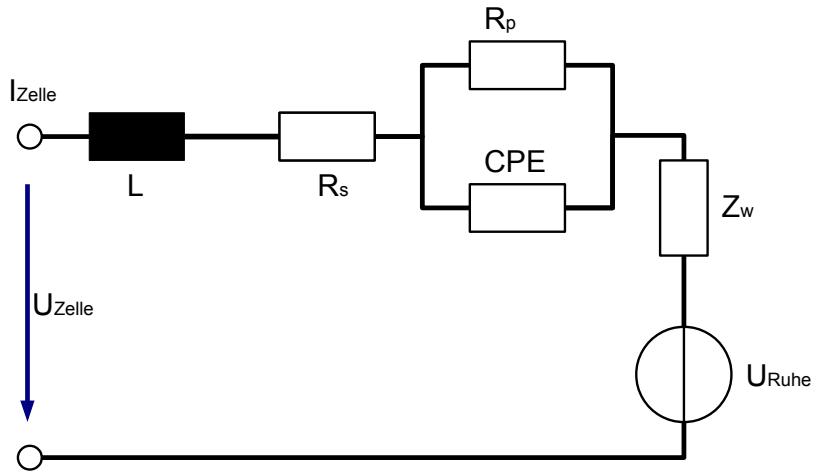


Abbildung 2.16: Ersatzschaltbild der Impedanzspektroskopie nach Roscher [22].

Diese Elemente bilden einen Teil des in Abbildung 2.16 gezeigten ESB. Die Parameter für die Bauteile werden über ein Fehlerminimierungsverfahren ermittelt. Die Impedanzspektren dieses Modells und des ursprünglichen Modells sind Abbildung 2.14 auf Seite 39 abgebildet. Man erkennt, dass der prinzipielle Verlauf vom zweiten Modell nachvollzogen wird. Es existieren jedoch teilweise erhebliche Abweichungen.

Beispielsweise lässt sich die angesprochene Krümmung der Kurve hin zu kleinen Frequenzen nicht mit einem einzelnen Warburg-Element abbilden, das einen linearen Verlauf hat. Der gekrümmte Verlauf der Impedanz hin zu sehr hohen Frequenzen kann ebenfalls nicht mit einer idealen Induktivität dargestellt werden. Insgesamt bietet das Modell trotz der Verwendung von ungewöhnlichen und aufwändig zu simulierenden Komponenten nicht das gewünschte Ergebnis.

Quantmeyer, Kießling und Liu-Henke [21] nutzen, statt komplizierter Elemente, eine einstellbare Spannungsquelle für die Ruhespannung, einen Serienwiderstand als Klemmenwiderstand und zusätzlich eine Kaskade von Kapazitäten mit parallel geschalteten Widerständen. Dieses Modell kann als erweitertes Modell des ersten Roscher Modells gesehen werden, das anstatt eines einzelnen Elementes aus paralleler Kapazität und parallelem Widerstand eine Reihenschaltung solcher Elemente verwendet. Wie bereits gesagt, lässt sich durch Parallelschalten von Kapazität und Widerstand ein Bogen erzeugen, der immer unter einem Winkel von 90 ° aus seinem Ursprung tritt. Durch in Reiheschalten mehrere solcher Elemente mit verschiedenen Zeitkonstanten lassen sich verschiedene Bögen im Spektrum erzeugen und die Form der einzelnen Bögen variieren. Mit der Anzahl dieser Elemente steigt der Simulationsaufwand dieser Modelle, allerdings ist auch eine Verbesserung der Systemabbildung zu erwarten. Abbildung 2.17 auf der nächsten Seite zeigt das ESB für eine beliebige Anzahl von kaskadierten Bogenelementen. Die Autoren nutzen die Methode der kleinsten Fehler-

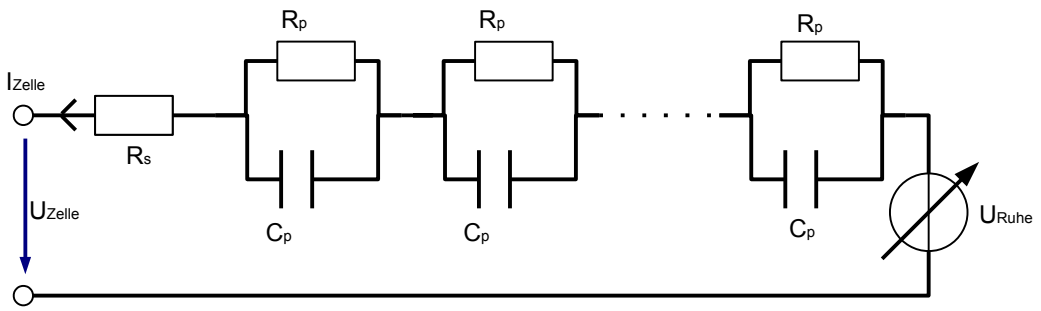
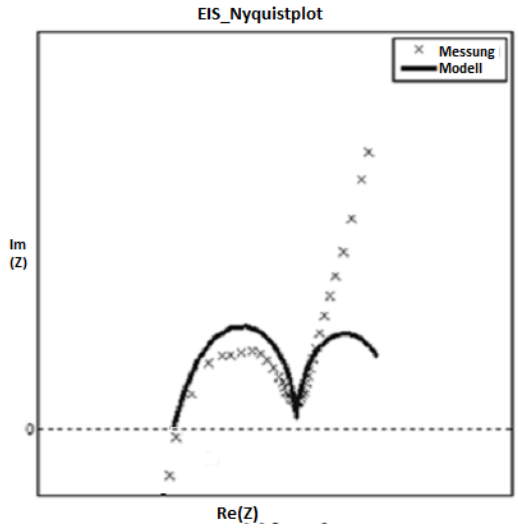


Abbildung 2.17: ESB mit variabler Spannungsquelle für die Ruhespannung nach [21].

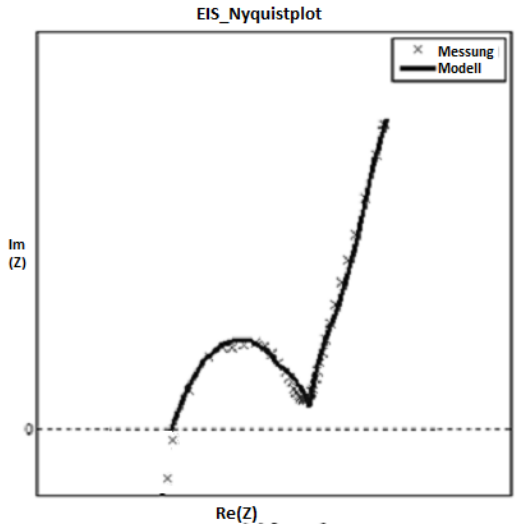
quadrate aus einer EIS-Messung, die besten Parameter für ein Modell mit einer gegebenen Anzahl von Bogenelementen zu ermitteln.

Mit einem solchen Modell lässt der induktive Anteil hin zu sehr schnellen Frequenzen nicht abbilden. Außerdem besteht die Gefahr, dass bei Modellen, die zu viele Parameter mit Hilfe der Methode kleinster Fehlerquadrate anzupassen versuchen, am Ende ein Modell entsteht, das zwar alle Punkte der Testmessung trifft, dazwischen aber deutlich schwingt. Daher ist es zum einen wichtig, eine deutlich größere Anzahl an Messpunkten als die Zahl der Parameter zu verwenden und im Anschluss zu überprüfen, ob das erstellte Modell plausibel ist. Außerdem ist klar, dass ein solches Modell nur innerhalb des mit der Testmessung abgedeckten Frequenzbereiches das System nachbilden kann, weshalb man darauf achten muss den gesamten relevanten Frequenzbereich des Systems in die Modellbildung mit einzubeziehen.

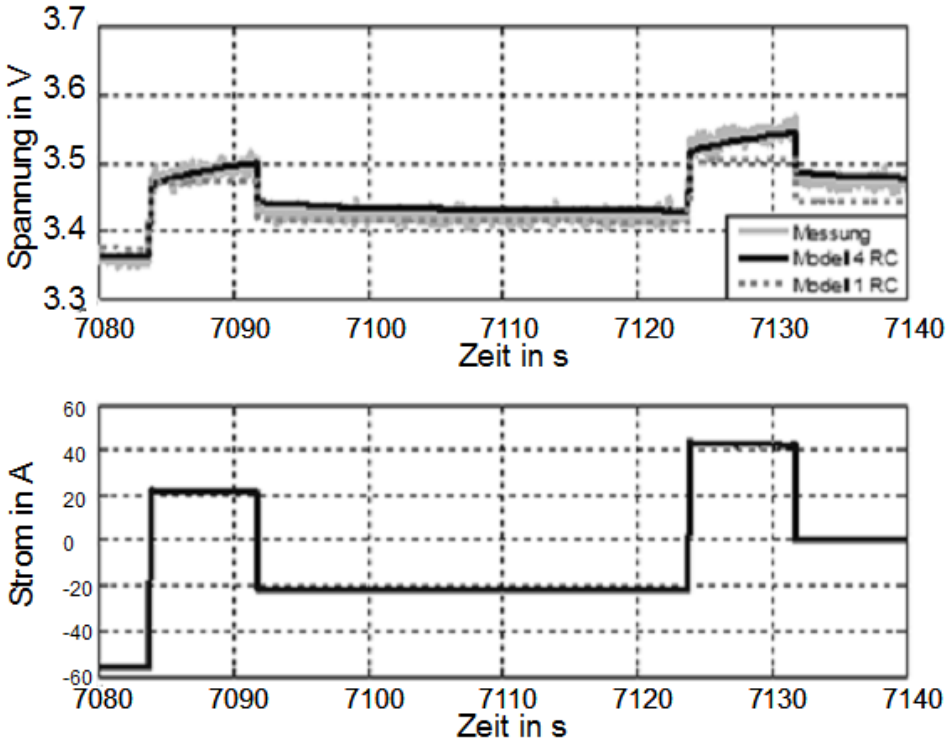
Die Autoren testen dieses Modell und stellen fest, dass für vier Bogenelemente bereits eine annehmbare Annäherung an das Impedanzverhaltens eines nicht näher genannten Lithium-Interkalations-Akkumulators im Frequnzbereich zwischen 50 mHz und 2 kHz erreichbar ist. Bei nur zwei Elementen weicht die Kurve noch deutlich ab. Die Verläufe sind in Abbildung 2.18 a) und b) dargestellt. Das parametrisierte Modell mit den vier Elementen verwenden sie, um den Verlauf einer Batteriespannung bei mehrsekündiger Ladung und Entladung mit großen Strömen zu simulieren. Abbildung 2.18 c) zeigt, dass der Verlauf der Kurve annähernd dem gemessenen Spannungsverlauf entspricht, obwohl die Parameter für das Modell über einen anderen Betriebsfall ermittelt wurden. Die Autoren machen keine Aussage über die quantitative Art des Fehlers der Simulation. Dennoch scheinen die Ergebnisse darauf hinzudeuten, dass hier ein Modell gefunden wurde, das sowohl Kleinsignalverhalten als auch das Betriebsverhalten abbilden kann.



(a) Impedanzspektrum des Modells mit zwei Bogenelementen.



(b) Impedanzspektrum des Modells mit vier Bogenelementen.



(c) Simulation eines Betriebsfalls mit sekundenlanger Ladung und Entladung bei verschiedenen Stromstufen.

Abbildung 2.18: Modelleigenschaften der Modelle nach Quantmeyer et al. mit zwei oder vier Bogenelementen, bestehend aus einer Kapazität mit parallelem Widerstand. Entnommen aus [21].

2.2 Elektrochemische Impedanzspektroskopie

Im vorherigen Abschnitt wurde gezeigt, dass es möglich ist, mit Hilfe von einfachen Ersatzschaltbild-Modellen Vorhersagen über den Ladezustand SoC und den allgemeinen Gesundheitszustand SoH von Akkumulatoren zu machen. Damit diese Modelle genutzt werden können, müssen ihre Parameter bestimmt werden. Da das Modell selber von verschiedenen Zuständen wie dem SoC und dem SoH abhängt, ist es notwendig, die Modellparameter laufend aktuell zu halten, wenn man verhindern will, dass deutliche Abweichungen zwischen Modellverhalten und Simulationsverhalten entstehen. Wie ebenfalls im vorherigen Abschnitt angesprochen, kann ein Spektrum des komplexen Innenwiderstandes der Batterie über verschiedene Frequenzen Aufschluss über solche Parameter geben. Das Erfassen eines solchen Spektrums wird 'Elektrochemische Impedanzspektroskopie (EIS)' genannt. Ziel dieser Arbeit ist es eine solche EIS mit den Sensoren des BATSEN-Projektes durchzuführen. Dazu wird der aktuelle Stand der Technik der EIS im Folgenden vorgestellt.

2.2.1 Begriffserklärung Impedanz

In diesem Abschnitt soll der Begriff Impedanz, der das Ergebnis einer Elektrochemischen Impedanzspektroskopie darstellt, erläutert werden. Außerdem werden verschiedene Methoden zu Visualisierung von Impedanzen vorgestellt.

Mathematische Betrachtung der Impedanz

Jedes lineare elektrische System reagiert auf eine Anregung mit einer Spannung $u(t) = \hat{u} \cdot sin(2\pi f \cdot t)$ mit der Signalfrequenz f durch einen Strom für den allgemein gilt: $i(t) = \hat{i} \cdot sin(2\pi f \cdot t + \phi)$. Dabei ist ϕ die Phasenverschiebung zwischen den sinusförmigen Signalen. Handelt es sich bei dem System um einen Widerstand R oder ein Netzwerk aus Widerständen, so ist $\phi = 0$ und für den Strom gilt $\hat{i} = \hat{u}/R$. Im Zeitbereich ist der Zusammenhang zwischen Strom und Spannung durch das Lösen von Differentialgleichungen herzustellen.

Dies ist für einfache Systeme die beispielsweise nur aus einer Induktivität L oder einer Kapazität C bestehen noch möglich, da hier die Phasendrehung immer $\pm 1/2\pi$ beträgt.

Hier gilt dann für die Induktivität:

$$
\begin{aligned}
u(t) &= \frac{di(t)}{dt} \cdot L \\
di(t) &= \frac{1}{L} u(t) dt \\
i(t) &= \frac{1}{L} \int_{-\infty}^{\infty} u(t) dt \\
i(t) &= \frac{1}{L} \int_{-\infty}^{\infty} (\hat{u} \cdot sin(2\pi f \cdot t)) dt \\
i(t) &= \frac{\hat{u}}{L} \int_{-\pi}^{\pi} sin(2\pi f \cdot t) dt \\
i(t) &= -\frac{\hat{u}}{L} \left[cos(2\pi f \cdot t) \cdot \frac{1}{2\pi f} \right] \\
i(t) &= \frac{\hat{u}}{2\pi f \cdot L} sin(2\pi f \cdot t - \frac{pi}{2}) \\
\Rightarrow & \qquad \hat{i} = \frac{\hat{u}}{2\pi f \cdot L} \qquad \phi = -\frac{pi}{2}
\end{aligned}
$$

Und für die Kapazität

$$
\begin{aligned}
i(t) &= \frac{dU(t)}{dt} \cdot C \\
i(t) &= \frac{d\hat{u} \cdot sin(2\pi f \cdot t)}{dt} \cdot C \\
i(t) &= \hat{u} \cdot cos(2\pi f \cdot t) \cdot C \cdot 2\pi f \\
i(t) &= \hat{u} C \cdot 2\pi f \cdot sin(2\pi f \cdot t + \frac{pi}{2}) \\
\Rightarrow & \qquad \hat{i} = \hat{u} C \cdot 2\pi f \qquad \phi = \frac{pi}{2}
\end{aligned}
$$

Besteht ein System aus mehreren Elementen, so ist das Suchen einer Lösung der Differentialgleichung im Zeitbereich oft nicht mehr sinnvoll. Stattdessen bietet es sich an, in den Frequenzbereich zu wechseln. Dazu werden alle Komponenten des Systems und alle Signale der Fourier-Transformation unterzogen. Mit dem Operator $\mathcal{F}\{\}$ für die Fourier-Transformation folgt:

$$\mathcal{F}\{u(t)\} = \underline{U}(j\omega)$$
$$\mathcal{F}\{i(t)\} = \underline{I}(j\omega)$$

Für Gleichstromanwendungen gilt das ohmschen Gesetz, das geschrieben werden kann als: $R = {}^{U}/_{I}$. Analog zum so bestimmten Gleichstromwiderstand kann im Zeitbereich für Wechselgrößen eine Impedanz bestimmt werden zu:

$$\underline{Z}(j\omega) = \frac{\underline{U}(j\omega)}{\underline{I}(j\omega)} \qquad = \frac{\mathcal{F}\{u(t)\}}{\mathcal{F}\{i(t)\}} \tag{2.1}$$

Auch beim Verzicht auf die Frequenzabhängigkeit von $\underline{Z}$ handelt es sich bei der Impedanz um eine komplexe Größe. Diese kann beschrieben werden durch einen Betrag $|\underline{Z}|$ und eine Phase ϕ, dann gilt:

$$\underline{Z} = |\underline{Z}| \cdot e^{j\omega\phi} \tag{2.2}$$

Außerdem kann eine komplexe Zahl durch einen Realteil $Re\{\underline{Z}\}$ und einen Imaginärteil $Im\{\underline{Z}\}$ beschrieben werden, sodass gilt:

$$\underline{Z} = \mathrm{Re}\{\underline{Z}\} + j \cdot Im\{\underline{Z}\} \tag{2.3}$$

wobei j die in der Elektrotechnik übliche Schreibweise der imaginären Einheit i ist.

Der Zusammenhang der Darstellungsarten komplexer Zahlen ergibt sich wie folgt:

$$\mathrm{Re}\{\underline{Z}\} = |\underline{Z}|cos(\phi) \qquad Im\{\underline{Z}\} = |\underline{Z}|sin(\phi) \tag{2.4}$$

$$|\underline{Z}| = \sqrt{\mathrm{Re}\{\underline{Z}\}^2 + Im\{\underline{Z}\}^2} \qquad \phi = tan^{-1}\left(\frac{Im\{\underline{Z}\}}{\mathrm{Re}\{\underline{Z}\}}\right) \tag{2.5}$$

Darstellungsformen

Die komplexe Ebene bietet eine gute Möglichkeit zur Darstellung komplexer Zahlen. Dazu wird auf der Abszisse der Realteil einer Zahl und auf der Ordinate der Imaginärteil aufgetragen.. Alternativ kann auch ein 'Zeiger' der Länge des Betrags der komplexen Zahl gezeichnet werden, der im Ursprung beginnt und mit der positiven Halbachse der Abszisse den Winkel ϕ aufspannt. Abbildung 2.19 zeigt, das für beide Darstellungsformen derselbe 'Zeiger' entsteht.

Die abgebildete Impedanz $\underline{Z}$ stellt den Wechselstromwiderstand für eine bestimmte Frequenz f dar. Die bei der Elektrochemischen Impedanzspektroskopie vermessenen Systeme haben einen frequenzabhängigen komplexen Widerstand. Das bedeutet, um eine Darstellung des Systemverhaltens zu erstellen, muss die Impedanz für verschiedene Frequenzen aufgetragen werden. Wie dieser Frequenzbereich zu wählen ist, hängt von den, im zu messenden System vorhandenen, Zeitkonstanten ab. In [30] wird dargestellt, dass bei Frequenzen zwischen 10 mHz und 100 mHz die Zeitkonstante der Diffusion der Ladungsträger im Elektrolyten liegt. Zwischen 1 Hz und 100 Hz liegt die Zeitkonstante des Ladungsdurchtritts ins Kristalgitter der Elektroden. Zwischen 100 Hz und etwa 1 kHz liegt die Zeitkonstante des Ionen-Durchtritts durch die Passivierungsschicht der Anode. Etwa bei 1 kHz wird die Impedanz der Zelle rein real und gibt hier den rein ohmschen Innenwiderstand der Zelle an. Anschließend zeigt die Zelle induktives Verhalten, das etwa bis in den hohen kHz Bereich beobachtet werden kann. Der für die Messung interessante Bereich erstreckt sich also von wenigen mHz bis zu einigen kHz.

Um die üblichen Darstellungsformen für Elektrochemische Impedanzspektroskopie vorstellen zu können, wurde das in Anhang G vorgestellte Batteriemodell simuliert, das im angegeben Frequenzbereich ein Verhalten zeigt, das an einen realen Akkumulator angelehnt ist.

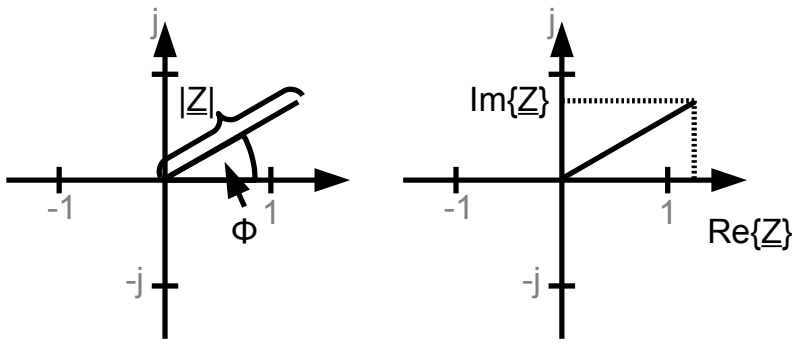


Abbildung 2.19: Darstellung einer komplexen Zahl in der komplexen Ebene. Links sind Betrag und Phase aufgetragen, rechts sind Realteil und Imaginärteil aufgetragen.

Bode-Diagram Beim Bode-Diagramm werden der Betrag der Impedanz und die Phase der Impedanz jeweils über der Frequenz aufgetragen. Dabei wird üblicherweise die Frequenzsachse in einer logarithmischen Einteilung dargestellt. Bei Frequenzgängen von Filtern ist es üblich, auch die Achse des Betrages logarithmisch darzustellen, da Betragsunterschiede über mehrere Dekaden üblich sind. Bei der EIS ist die Änderung des Impedanzbetrages über den betrachteten Frequenzbereich regelmäßig klein genug, sodass hier auf einer linearen Skala dargestellt werden kann. Diese Darstellung eignet sich besonders, um die Frequenzabhängigkeit darzustellen, den Einfluss der einzelnen Modellparameter kann man einer solchen Darstellung, wie sie in Abbildung 2.20 gezeigt ist, nicht entnehmen.

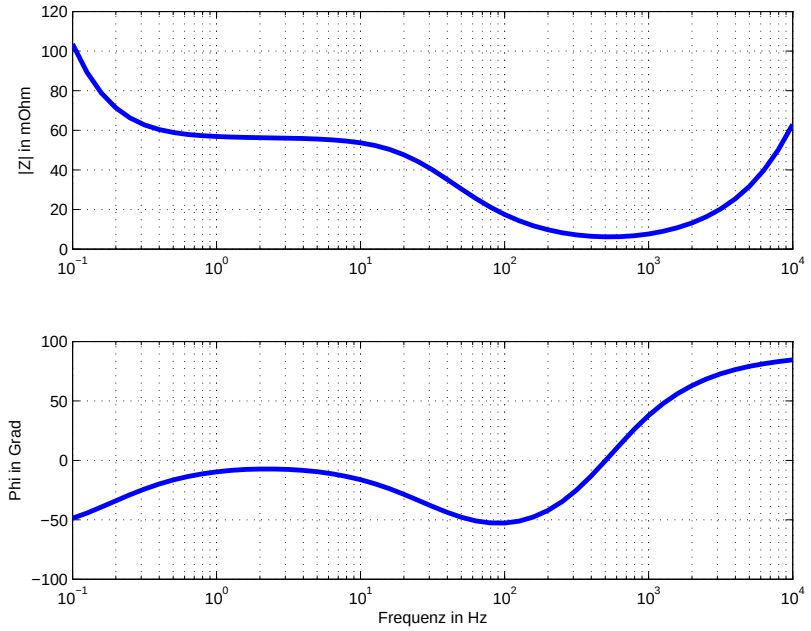


Abbildung 2.20: Bodediagramm des Spektrums eines Akkumulators. Die obere Abbildung zeigt den Betrag der komplexen Impedanz in $m\Omega$, die untere Abbildung zeigt den Betrag in $^\circ$

Nyquist-Diagram Bei einem Nyquist-Diagram wie es in Abbildung 2.21 dargestellt ist, wird der Imaginärteil über dem Realteil aufgetragen. Da die bei der EIS vermessenen Akkumulatoren im wesentlichen kapazitives Verhalten haben, wird dabei die Abszisse negiert, es werden also negative Imaginärteile, wie sie bei kapazitivem Verhalten auftreten, im ersten Quadranten dargestellt. Bei dieser Darstellungsart geht zunächst die Information über die Frequenzzuordnung der einzelnen Messpunkte verloren. Sollte hier nicht selbsterklärend sein, welche Punkte zu welchen Frequenzen gehören, so sollte dies im Diagramm vermerkt werden. Im gezeigten Diagramm finden sich die Impedanzen, die bei den kleinsten Frequenzen gemessen wurden, oben rechts. Sie haben den größten Realteil und den negativsten Imaginärteil. Die größten Messfrequenzen zeigen sich hier im Bereich unten rechts. Die gezeigte Kurve ist bijektiv (eineindeutig), das bedeutet jedem Frequenzwert ist genau eine komplexe Impedanz zugeordnet, dies ist jedoch keine allgemeine Eigenschaft eines Impedanzspektrums.

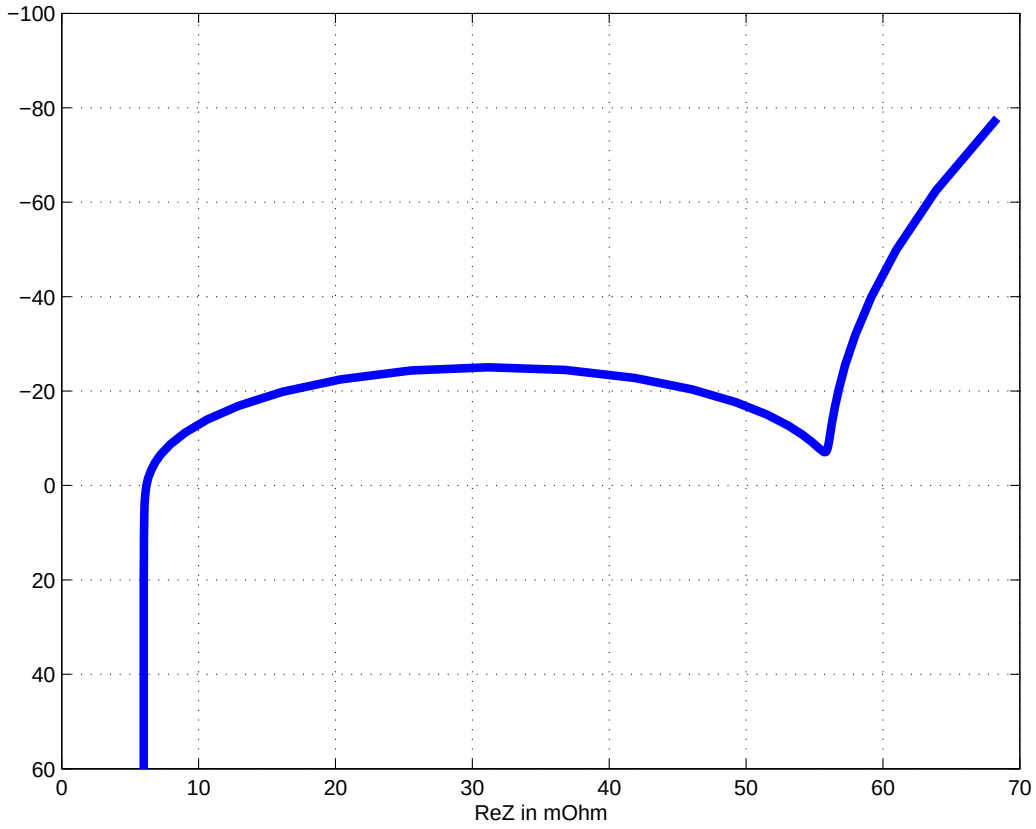


Abbildung 2.21: Nyquist-Diagram des Akkumulators.

Aus den einzelnen Abschnitten des Nyquist-Diagrams lässt sich der Aufbau des Modells, mit dem diese Messung erfolgt ist, nachvollziehen. Abbildung 2.22 zeigt erneut das Nyquist-Diagramm und zusätzlich das verwendete Ersatzschaltbild. Die zusammengehörenden Elemente und Kurvenabschnitte sind farblich markiert.

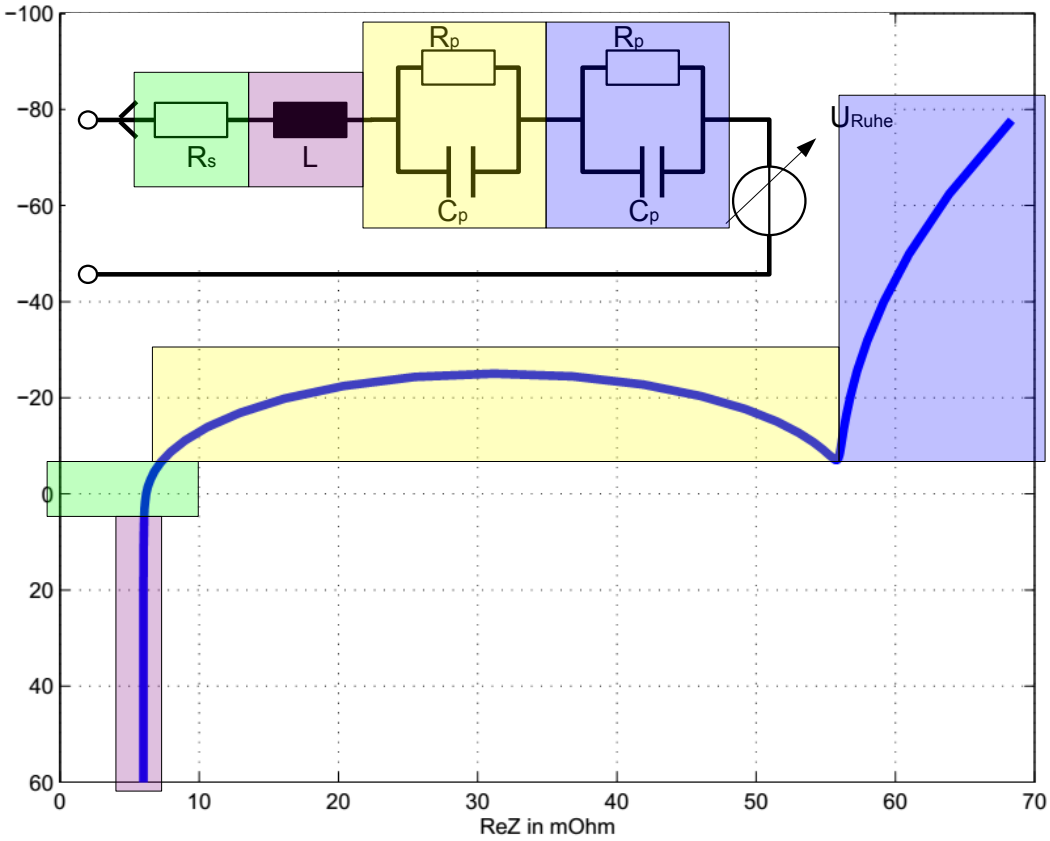


Abbildung 2.22: Nyquist-Diagram mit markiertem Verlauf entsprechend der beteiligten Schaltungselemente. Der Widerstand Rs kann direkt an der Schnittstelle der Ortskurve mit der x-Achse abgelesen werden. Die Induktivität L wirkt bei hohen Frequenzen und sorgt für eine mit der Frequenz wachsende Impedanz. Die beiden Bögen werden durch die in der Kapazität stark abweichenden Parallelschaltungen aus Kapazität und Widerstand erzeugt.

2.2.2 Messprinzip

In diesem Abschnitt soll das Prinzip der Elektrochemischen Impedanzspektroskopie erläutert werden.

Anregung

Um eine Impedanzspektroskopie an einem realen System durchzuführen, müssen die Impedanzen $\underline{Z}(j\omega)$ für verschiedene Frequenzen bestimmt werden. Im einfachsten Fall kann man dazu eine **monofrequente Anregung** verwenden. Dazu prägt man, für jede zu messende Frequenz, dem Prüfling ein sinusförmiges Signal ein. Dies kann entweder ein Strom oder eine Spannung sein. Wird ein Strom eingeprägt, so spricht man von einer galvanostatischen Messung, bei einer eingeprägten Spannung von einer potentiostatischen Messung. Abbildung 2.23 zeigt den Aufbau beider Anregungsschaltungen. Bei der potentiostatischen Regelung wird das Anregungssignal so ausgesteuert, das sich am Prüfling die gewünschte Spannungsschwankung einstellt. Der sich dabei einstellende Strom muss gemessen werden, um sowohl $u(t)$ als auch $i(t)$ zu kennen.

In Abschnitt 2.1.4 auf Seite 30 wurde gezeigt, dass Batteriezellen ein 'Gedächtnis' für den zuletzt geflossenen Strom besitzen, das sich sowohl auf die Ruhespannung als auch auf das dynamische Spannugsverhalten bei Belastung auswirkt [22]. Daher ist es sinnvoll, einen Akkumulator in verschiedenen Arbeitspunkten zu vermessen. Das heißt, dass dem Wechselanteil des Anregungssignals ein Gleichanteil hinzugefügt wird, der dafür sorgt, dass der Akkumulator entweder im Lade oder im Entladebetrieb getestet wird. Dies ist für die potentiostatische Messmethode nicht ohne Weiteres möglich, weshalb in dieser Arbeit ausschließlich die galvanostatische Messmethode Verwendung findet. Hierbei kann dem Anregungssignal $U(I_{soll})$ problemlos eine Gleichstromkomponente hinzugefügt werden.

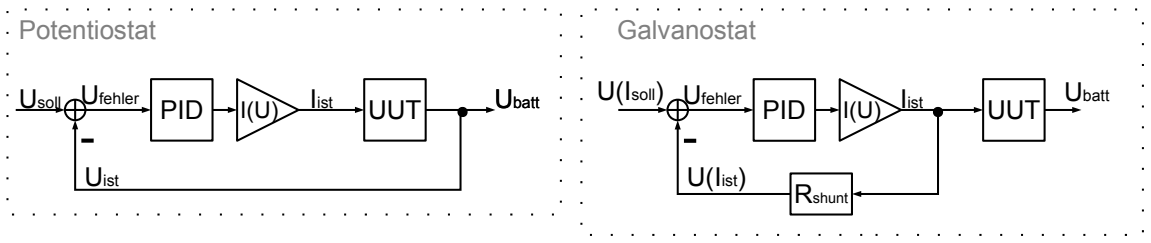


Abbildung 2.23: Die beiden Schaltbilder zeigen links die potentiostatische und rechts die galvanostatische Anregungsschaltung. Bei der galvanostatischen Messung erfolgt die Regelung auf den in die 'Unit under Test (UUT)' gegebenen Strom, die UUT selber ist nicht Teil der Regelstrecke.

Zur Erzeugung eines solchen Signals ist, wie in Abbildung 2.23 auf der vorherigen Seite, eine aktive Regelschaltung notwendig, um dem Strom zu einer Sinuskurve zu formen. Die oberste Reihe von Abbildung 2.24 auf der nächsten Seite zeigt ein solches Sinus-Signal mit einer Frequenz von 100 Hz, das durch einen Gleichanteil so verschoben ist, dass die gesamte Messung im Entladebetrieb durchgeführt wird. Durch das Variieren des Gleichanteils ließe sich nun der Arbeitspunkt der Messung einstellen.

Für aufwands-reduzierte Messungen kann die Verwendung einer vollwertigen Endstufe, wie sie für das Erzeugen eines Sinus-Signals notwendig ist ein limitierender Kostenfaktor sein. Eine einfacher zu erzeugende Anregung stellt die **Rechteckförmige Anregung** dar. Die unteren beiden Reihen von Abbildung 2.24 auf der nächsten Seite zeigen Signale, die mit weniger Hardwareaufwand erzeugt werden können. Das mittlere Signal stellt ein Rechteck-Signal dar, welches ansonsten mit den Eigenschaften des Sinus-Signals übereinstimmt. Es schwingt ebenfalls mit der Grundfrequenz 100 Hz und hat den identischen Gleichanteil und die gleiche maximale Auslenkung. In der rechten Spalte sind jeweils die Spektren des Signals bis zu einer Frequenz von 1 kHz dargestellt.

Während im Spektrum des Sinus-Signals lediglich die Grundfrequenz und der Gleichanteil wiederzufinden sind, sieht man beim Rechtecksignal weitere Frequenzanteile bei den ungeraden Harmonischen der Grundfrequenz. Die Amplituden dieser Frequenzanteile fallen mit zunehmender Frequenz ab. Der Verlauf der Amplituden folgt der Si-Funktion, die durch die Fouriertransformation des Rechtecks entsteht. Obwohl die Amplituden der Harmonischen mit der Si-Funktion kleiner werden, ist das Spektrum des periodischen Rechtecksignals nicht bandbegrenzt. Da das Signal für die EIS mit einer endlichen Abtastrate abgetastet werden muss, kommt es zum Aliasing bei der Messung, die nicht durch ein Anti-Aliasing-Tiefpass vor der Abtastung verhindert werden kann, da die Abtastrate jeweils an die EIS-Frequenz angepasst werden muss. In [7] wird gezeigt, dass der Einfluss des Aliasing auf die Signal-Amplitude der Grundwelle abhängt von der Rate der Überabtastung. Tabelle 2.1 auf der nächsten Seite zeigt, das der Einfluss deutlich geringer ist, wenn ungerade Faktoren für die Überabtastung verwendet werden. Dies liegt daran, dass in einem solchen Fall die Maxima und Minima der verschobenen Si-Funktionen nicht aufeinanderfallen.

Für die Verwendung eines Rechteck-Signals ist es also notwendig, eine ausreichend große Abtastrate zu wählen, um den Einfluss von Aliasing auf das Signal klein zu halten. Theoretisch ist es möglich, zusätzliche Frequenzanteile für die Auswertung EIS heranzuziehen. Dabei ist zum einen zu beachten, dass dann der Überabtastungsfaktor auf die größte auszuwertende Frequenz bezogen werden muss, andererseits ist zu bedenken, dass der Signal-Rausch Abstand der größer werdenden Frequenzanteile dadurch sinkt, dass diese immer kleiner Amplituden aufweisen.

Das Erzeugen eines Rechtecksignals erfordert zwar weniger Aufwand, als das Erzeugen eines gesteuerten Sinus-Signals, jedoch ist hierfür dennoch eine Regelung notwendig. Für

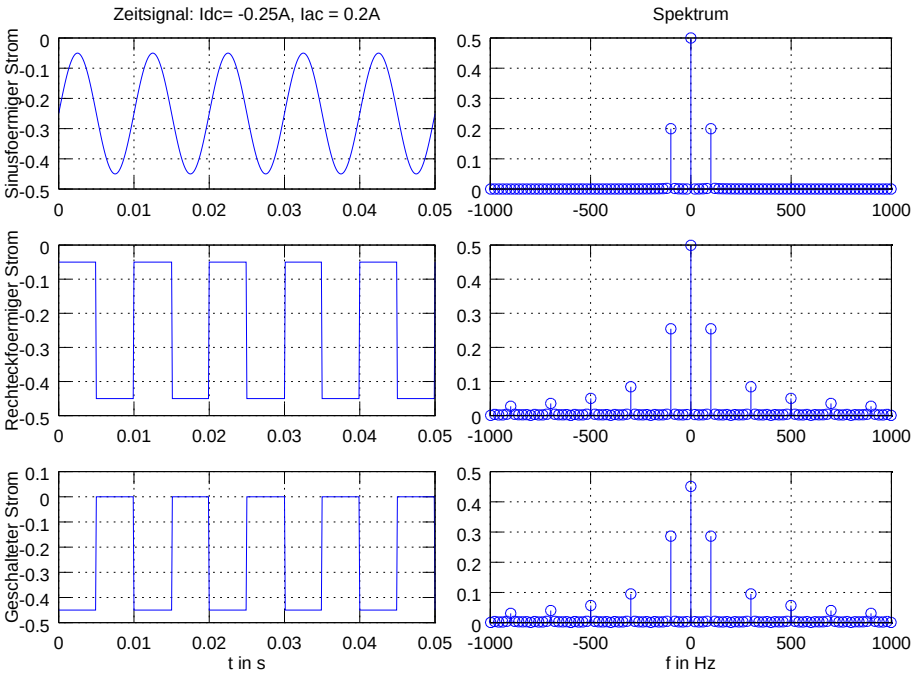


Abbildung 2.24: Vergleich verschiedener Anregungen und ihrer Spektren. Oben die monofrequente Anregung mit einem Sinus. Das Spektrum weist lediglich den Gleichanteil und die Messfrequenz auf. In der Mitte sieht ist ein rechteckförmigen Signal mit Stromoffset abgebildet. Der Gleichanteil im Signal ist identisch mit der Sinusanregung, zusätzlich finden sich im Spektrum die ungeraden Harmonischen der Grundfrequenz. Das untere Bild erzeugt einen Strom wie er beim gepulsten Einschalten einer ohmschen Last entsteht. Der Gleichanteil ist verschoben, da er automatisch in der Mitte zwischen Maximalstrom und Null liegt.

eine sehr kostengünstige Erzeugung der Anregung kann eine **geschaltete ohmsche Last** verwendet werden. Das entsprechende Stromsignal ist in Abbildung 2.24 in der untersten Reihe abgebildet. Der Strom ist Null, solange die Last ausgeschaltet ist. Wird die Last eingeschaltet, so stellt sich ein Strom ein, der abhängig von der aktuellen Zellspannung, dem Innenwiderstand der Zelle und dem Last-Widerstand ist. In diesem Fall kann kein Arbeitspunkt eingestellt werden, da der Gleichanteil des Stromsignals immer der halben Amplitude entspricht. Das dargestellte Spektrum zeigt jedoch, dass auch in einem solchen Signal die gesuchte Grundfrequenz vorhanden ist und somit zur Auswertung zur Verfügung steht.

Überabtastung K	7	8	15	16	31	32	63	64
Anteil / %	0.84	2.61	0.183	0.65	0.04	0.16	0.01	0.04

Tabelle 2.1: Aliasig-Anteil der Grundwelle in Abhängigkeit der Überabtastung aus [7]

Messung der Systemantwort

Die Spannungsantwort des Prüflings auf einen Anregestrom muss gemessen werden. Wird bei einer galvanostatischen Messung der Strom

$i(t) = I_{DC} + \hat{i} \cdot sin(2\pi f \cdot t)$

in den Prüfling eingeprägt, so hat die Spannungsantwort die Form $u(t) = U_{OCV} + Z_{DC} \cdot I_{DC} + \hat{u} \cdot sin(2\pi f \cdot t + \phi)$, da lineare Systeme lediglich eine Skalierung der Amplitude und eine Phasendrehung verursachen können.

Dabei ist I_{DC} der überlagerte Gleichstrom der Anregung, U_{OCV} die Ruhespannung des Prüflings für den gegebenen SoC und Z_{DC} der für Gleichstrom wirksame Wert des Innenwiderstandes des Prüflings.

Analog zum Widerstand, der bei Gleichströmen dem ohmschen Gesetz nach durch $R = {}^{U}/_{I}$ bestimmt werden kann, erhält man ein Aussage über die frequenzabhängige Impedanz durch Rechnung aus:

$$\begin{aligned} \underline{Z}(j\omega) &= \frac{\underline{U}(j\omega)}{\underline{I}(j\omega)} \\ &= \frac{\mathcal{F}\{u(t)\}}{\mathcal{F}\{i(t)\}} \\ &= \frac{\mathcal{F}\{\hat{u} \cdot sin(2\pi f_{EIS} \cdot t + \phi)\}}{\mathcal{F}\{\hat{i} \cdot sin(2\pi f_{EIS} \cdot t)\}} \end{aligned}$$

Da die Aufzeichnung von $u(t)$ und $i(t)$ durch Abtasten erfolgt, sind diese Signale zeitlich diskretisiert, sodass statt der Fourier-Transformation die Diskrete-Fourier-Transformation (DFT) durchgeführt werden muss. Diese liefert keine kontinuierlichen Spektren sondern nur Spektrallinien bei bestimmten Frequenzen. Für diese gilt bei einer Abtastrate von F_a und einer Anzahl an transformierten Werten N_{DFT}, dass folgende Frequenzen abgebildet werden können:

$$f_{DFT} = \left\{ 0, \frac{1 \cdot F_a}{N_{DFT}}, \frac{2 \cdot F_a}{N_{DFT}}, ..., \frac{(1 - N_{DFT}) \cdot F_a}{N_{DFT}} \right\} \tag{2.6}$$

Da bei der EIS-Messung, im Unterschied zu vielen Anderen Messaufgaben, die gesuchte Frequenz bekannt ist, ist es sinnvoll, das Verhältnis aus Abtastrate und Anzahl der Abtastwerte so anzupassen, dass die Anregungsfrequenz F_{EIS} genau von einer Frequenz der Menge

f_{DFT} getroffen wird. Den diskreten Spektren in Abbildung 2.24 auf Seite 53 kann entnommen werden, dass bei sinusförmiger Anregung immer nur ein Frequenzwert von Interesse ist. Die DFT oder auch die auf Rechenaufwand optimierte Variante Fast-Fourier-Transformation (FFT) berechnen aber immer alle Frequenzpunkte die in f_{DFT} enthalten sind. Auch wenn man bei rechteck-förmiger Anregung mehrere Frequenzen auswertet, so bleibt immer der Großteil der berechneten Werte ungenutzt. Daher ist es sinnvoll nach einer Möglichkeit zu suchen, gezielt nur bestimmte Werte der DFT zu berechnen. Ein Algorithmus, der diese Aufgabe übernimmt, ist der im Folgenden vorgestellte Goertzel-Algorithmus.

Goertzel-Algorithmus

Die hier dargestellte Herleitung des Goertzel-Algorithmus aus der DFT entstammt [31] und wurde zum besseren Verständnis mit mehr Zwischenschritten nachvollzogen.

Für die DFT $X[k]$ einer Folge $x[n]$ der Länge N gilt:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad k = 0, 1, \ldots, N-1, \tag{2.7}$$

Wobei für die sogenannten Twiddle-Faktoren $W_N^{kn} = e^{-j(2\pi kn/N)}$ gilt. Die Berechnung aller N Werte von $X[k]$ benötigt laut Literatur $4N^2$ reele Multiplikationen und $N(4N-2)$ reele Additionen.

Das Ergebnis der e-Funktion hat immer den Betrag 1 und beschreibt für $\phi = 0 \rightarrow 2\pi$ einen Vollkreis in der komplexen Ebene. Daraus resultiert, dass die Funktion periodisch mit ganzahligen Vielfachen des Exponenten von 2π ist.

Daher gilt, dass

$$W_N^{-kN} = e^{j\frac{2\pi}{N}Nk} = e^{j2\pi k} = 1 \qquad | \quad k \in \mathbb{N}$$

Das bedeutet, dass dieser Term an jeden anderen multipliziert werden kann, ohne das Ergebnis zu verändern. Dann gilt für Gleichung (2.7):

$$\begin{aligned} X[k] &= W_N^{-kN} \sum_{r=0}^{N-1} x[r] W_N^{kr}, && k = 0, 1, \ldots, N-1, \\ &= \sum_{r=0}^{N-1} x[r] W_N^{-k(N-r)}, && k = 0, 1, \ldots, N-1, \end{aligned} \tag{2.8}$$

.

Unter Zuhilfenahme der diskrete Darstellung der Sprungfunktion:

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} \tag{2.9}$$

Kann die folgende Gleichung aufgestellt werden:

$$y_k[n] = \sum_{r=-\infty}^{\infty} x[r] W_N^{-k(n-r)} u[n-r] \tag{2.10}$$

Da Eingangsfolge und Ausgangsfolge auf N Werte beschränkt sind, gilt:

$x[r] = 0$ für $r < 0$ und $r \geq N-1$
$y_k[n] = 0$ für $n < 0$ und $n \geq N-1$

und Gleichung (2.10) kann beschränkt werden auf:

$$y_k[n] = \sum_{r=0}^{N-1} x[r] W_N^{-k(n-r)} \tag{2.11}$$

Daraus folgt beim Vergleich mit Gleichung (2.8) auf der vorherigen Seite

$$X[k] = y_k[n]|_{n=N} \tag{2.12}$$

Der in Abbildung 2.25 auf der nächsten Seite gezeigte Signalflussgraph realisiert die Gleichung (2.11), wenn er vor $n = 0$ in Ruhe ist.

Da es sich sowohl bei $y_k[n]$ als auch bei W_N^{-k} um komplexe Werte handelt, benötigt die Berechnung eines neuen $y_k[n]$ vier reelle Multiplikationen und vier reelle Additionen.

Um einen bestimmten Wert $X[k] = y_k[N]$ zu berechnen, müssen alle Iterationsschritte $y_k[1], y_k[2], \ldots, y_k[N-1]$ berechnet werden. Daraus resultiert, dass die Berechnung eines Wertes $X[k] = y_k[N]$ $4N$ reelle Additionen und $4N$ reelle Multiplikationen benötigt. Das ist minimal schlechter als die DFT für die Berechnung aller $X[k]$ benötigt. Dennoch hat diese Implementierung schon in dieser Form den Vorteil, das die Twiddle-Faktoren W_N^{-nk} nicht berechnet und gespeichert werden müssen. Stattdessen muss einmal W_N^{-k} berechnet werden und die weiteren Werte entstehen durch die Rekursion.

Um diesen Vorteil der einfachen Twiddle-Faktor Generierung bei gleichzeitiger Reduktion der Multiplikationen zu Nutzen, wird zunächst die Übertragungsfunktion des Signalflussgraphen in Abbildung 2.25 auf der nächsten Seite aufgestellt:

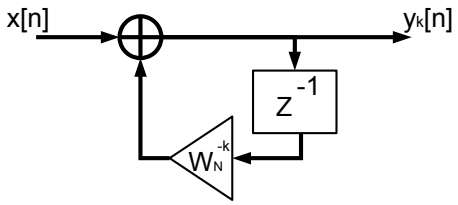


Abbildung 2.25: Signalflußgraph rekursive Berechnung von X[k]. Durch die Rekursion muss nur einmal der Wert W_N^{-k} berechnet werden und für die Twiddle-Faktoren muss kein Speicher reserviert werden.

$$H_k[z] = \frac{1}{1 - W_N^{-k} z^{-1}} \tag{2.13}$$

Erweitert man mit dem Term $1 - W_N^k z^{-1}$ so erhält man:

$$\begin{aligned}
H_k[z] &= \frac{1 - W_N^k z^{-1}}{(1 - W_N^{-k} z^{-1})(1 - W_N^k z^{-1})} \\
&= \frac{1 - W_N^{-k} z^{-1}}{(1 - e^{-j\frac{2\pi k}{N}} z^{-1})(1 - e^{j\frac{2\pi k}{N}} z^{-1})} \\
&= \frac{1 - W_N^{-k} z^{-1}}{(1 - e^{-j\frac{2\pi k}{N}} z^{-1} - e^{j\frac{2\pi k}{N}} z^{-1} + e^{j\frac{2\pi k}{N}} e^{-j\frac{2\pi k}{N}} z^{-2})} \\
&= \frac{1 - W_N^{-k} z^{-1}}{(1 - (e^{-j\frac{2\pi k}{N}} + e^{j\frac{2\pi k}{N}}) z^{-1}) + z^{-2})} \\
&= \frac{1 - W_N^{-k} z^{-1}}{1 - \left(\left(\cos\left(-\frac{2\pi k}{N}\right) + j \sin\left(-\frac{2\pi k}{N}\right)\right) + \left(\cos\left(\frac{2\pi k}{N}\right) + j \sin\left(\frac{2\pi k}{N}\right)\right)\right) z^{-1} + z^{-2}} \\
&= \frac{1 - W_N^{-k} z^{-1}}{1 - \left(\left(\cos\left(\frac{2\pi k}{N}\right) - j \sin\left(-\frac{2\pi k}{N}\right)\right) + \cos\left(\frac{2\pi k}{N}\right) + j \sin\left(\frac{2\pi k}{N}\right)\right) z^{-1} + z^{-2}} \\
&= \frac{1 - W_N^{-k} z^{-1}}{1 - 2\cos\left(\frac{2\pi k}{N}\right) + z^{-2}}
\end{aligned} \tag{2.14}$$

Diese Übertragungsfunktion wird durch den in Abbildung 2.26 auf der nächsten Seite gezeigten Signalflussgraphen realisiert. Hierbei ist zu beachten, dass zur Berechnung des Wertes $y_k[N]$ lediglich der rekursive linke Teil N-fach ausgeführt werden muss. Die Multiplikation mit dem Twiddle-Faktor muss nur einmal bei der Berechnung des Ausgangswertes durchgeführt werden. Wenn man von einer komplexen Eingangsgröße $x[n]$ ausgeht, liegt der Aufwand der Berechnung bei zwei reellen Multiplikationen und vier reellen Additionen für jede Rekur-

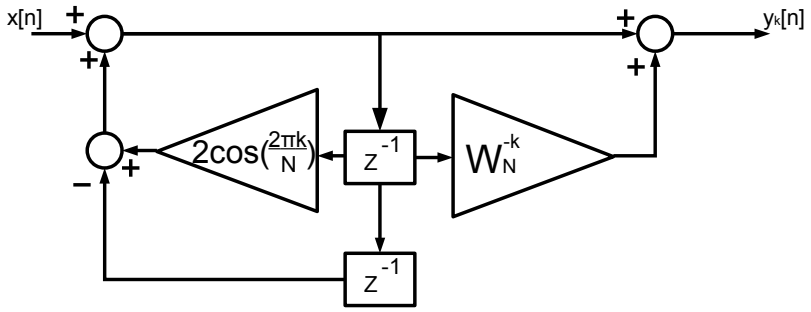


Abbildung 2.26: Signalflussdiagramm zweiter Ordnung zur rekursiven Berechnung.(Goertzel-Algorithmus).

sion und vier reelle Multiplikationen und vier reellen Additionen für den Ausgangswert. Damit ergibt sich der Aufwand zur Berechnung eines Wertes zu 2(N+2) reellen Multiplikationen und 4(N+1) reellen Additionen. Die Berechnung kommt also mit etwa der Hälfte der Multiplikationen der direkten Methode aus und bietet zusätzlich den Vorteil, dass die Faktoren $2\cos\left(\frac{2\pi k}{N}\right)$ und W_N^k nur einmalig berechnet und gespeichert werden müssen.

Die Werte W_N^{nk} entstehen auch hier durch die Rekursion. Wenn festgelegt werden kann, dass die Eingangsfolge $x[n]$ reell ist, so bleiben alle Werte im rekursiven Pfad reell. Damit reduziert sich der Aufwand für eine Rekursion auf eine reelle Multiplikation und zwei reelle Additionen.

Ein weiterer Vorteil dieses Algorithmus ist, dass neben dem Speicherplatz für die Eingangsfolge nur Speicher für die zwei Faktoren und die zwei Verzögerer reserviert werden muss. Dabei ist der Faktor $2 \cdot \cos\left(\frac{2\pi k}{N}\right)$ immer reell. Da wir von einer reellen Eingangsfolge Ausgehen, muss auch für die zwei Verzögerer der Speicher nicht für komplexe Zahlen ausgelegt werden, sodass insgesamt nur fünf Speicherstellen notwendig sind. Durch den geringen Speicherbedarf, die Möglichkeit gezielt einzelne Frequenzen, anstatt des gesamten Spektrums, zu berechnen und weil das Berechnen der W_N^{nk} durch die Rekursion so einfach wird, eignet sich der Goertzel-Algorithmus hervorragend für die Implementation in Mikrocontrollern und DSPs mit beschränkter Rechenleistung und Speicherangebot.

2.2.3 Impedanzspektroskopie an Akkumulatoren

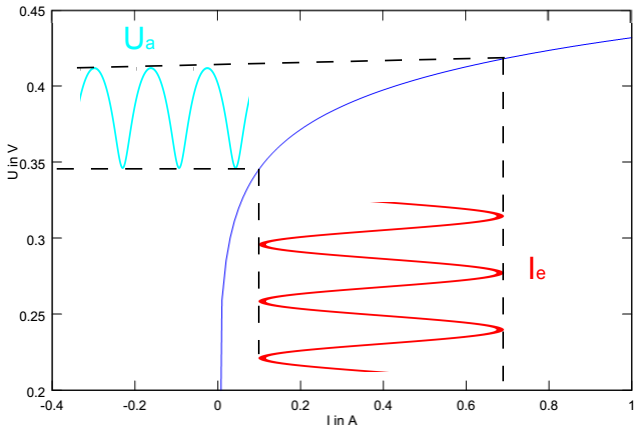
Die Eigenschaften von Akkumulatoren stellen einige besondere Ansprüche an die Elektrochemische Impedanz-Spektroskopie, die im Folgenden kurz vorgestellt werden sollen.

Linearität des Systems

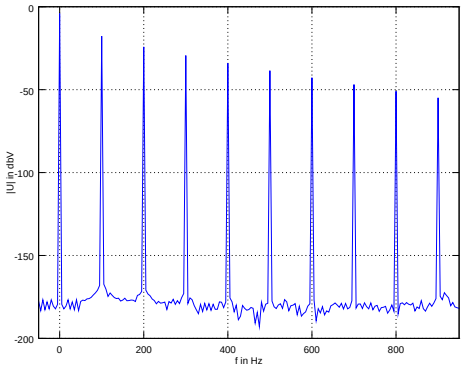
Chemische Prozesse wie sie in einem Akkumulator ablaufen, weisen im Allgemeinen ein nicht-lineares Strom/Spannungsverhalten auf [32]. Bei ausreichend kleiner Auslenkung der Spannungsantwort auf eine Anregung kann das Strom/Spannungsverhalten als quasi-linear über die Höhe der Auslenkung angenommen werden.

Abbildung 2.27 auf der nächsten Seite zeigt die durch Nichtlinearität eines Systems entstehende Verzerrung beispielhaft an einer Diodenkennlinie mit $I_s = 1\,\mathrm{uA}$ und $U_T = 26\,\mathrm{mV}$. Man kann erkennen, dass für große Eingangsströme I_e die Ausgangsspannung U_a verzerrt wird, was sich im Spektrum durch nicht ausreichend bedämpfte Harmonische der Grundschwingung zeigt. Das zweite Bild zeigt, dass für kleine Amplituden der Kurvenverlauf annähernd linear ist, sodass die Verzerrung mit dem Auge nicht mehr zu sehen ist. Da es sich allerdings nur um eine annähernde Linearität handelt, findet man im Spektrum des zweiten Ausgangssignals immer noch die Harmonischen der Grundschwingung, jedoch deutlich stärker bedämpft.

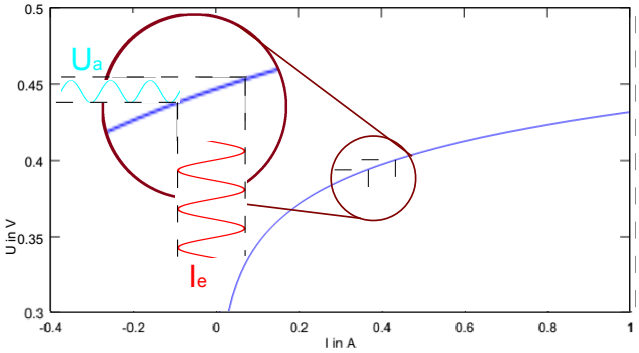
Der Wechselanteil des Anregestroms ist also ausreichend klein zu wählen, um eine Verzerrung der Spannungsantwort durch Nichtlinearität zu vermeiden. Andererseits sinkt mit der Amplitude der Antwort auch der Signal-Rausch-Abstand. Verschiedene Literaturquellen nennen dazu die maximale Wechselspannungsamplituden zwischen 1 mV und 10 mV pro Zelle als ausreichend klein, um von einem linearen Verhalten der Antwort auszugehen [6], [7], [18]. In [7] wird eine Methode festgelegt, wie durch Messung des Klirrfaktors, also des Verhältnisses aus Grundfrequenzleistung und Leistung der Harmonischen, eine Eingangsgröße für einen Regler gewonnen werden kann, der die Verzerrung auf ein vorgegebenes Maß beschränkt und dabei den Signal-Rausch-Abstand minimiert.



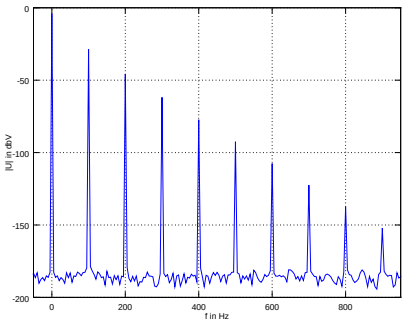
(a) Verzerrung bei zu großem Eingangsstrom.



(b) Spektrum der verzerrten Spannungsantwort



(c) Quasi verzerungsfreie Spannungsantwort bei ausreichend kleinem Eingangsstrom.



(d) Spektrum der quasiverzerruhngsfreien Spannungsantwort aus c)

Abbildung 2.27: Einfluss des Eingangsstrom-Amplitude auf die Verzerrung der Ausgangsspannung. Die Verzerrung durch den zu großen Eingangsstrom ist in (a) mit bloßem Auge zu erkennen, in (c) sieht man, dass die Harmonischen der Eingangsfrequenz nur langsam abnehmen. In (b) wird mit deutlich kleinerer Amplitude angeregt, sodass (d) zeigt, dass hier die Harmonischen deutlich stärker bedämpft sind.

Erzeugung der Anregung

Der Betrag des Innenwiderstands eines Akkumulators bewegt sich in der Größenordnung 1 $m\Omega$ bis 100 $m\Omega$. Um die genannten Ausgangsspannungen an diesen Innenwiderständen zu erzeugen, sind Anregungsamplituden im Bereich 10 mA bis 10 A notwendig. In Abschnitt 2.2.2 auf Seite 51 wurde festgestellt, dass die Impedanzspektroskopie eines Akkumulators in verschiedenen Arbeitspunkten erfolgen sollte, sodass der Gleichstrom für die Arbeitspunkteinstellung hinzukommt, der in der Größenordnung der Anregungsamplitude und darüber anzusetzen ist.

Daraus folgt, dass die Anregeschaltung ausreichend groß dimensioniert werden muss, um die positiven Ströme dieser Größenordnung im Ladebetrieb bereitstellen zu können und die negativen Ströme im Entladebetrieb aufzunehmen und in Wärme zu wandeln. Die Integration einer Anregeschaltung, die neben hohen Strömen auch eine gezielte Signalform bereitstellen können, kann in der Herstellung große Kosten verursachen. Dies kann gerade bei kostengünstigen EIS-Sensoren im Feldeinsatz zum Problem werden. In [7] wird festgestellt, dass ein Zellensensor, der eine Batterie in ihrer realen Betriebsumgebung (in Situ) überwacht, Herstellungskosten aufweisen muss, die in einem günstigen Verhältnis zu den Kosten des Akkumulators stehen. Die dieser Arbeit zugrunde liegende Betrachtung von Flugzeugbatterien lässt hier mehr Spielraum, da zum einen der Preis eines Akkumulators für den Flugbetrieb den eines Kraftfahrzeugs bei Weitem übersteigt. Zum anderen liegt das Hauptaugenmerk hier auf der Erhöhung der Betriebssicherheit und damit dem Verhindern von Ausfallzeiten des ganzen Flugzeugs ode,r wie im Falle der Boeing 777, der gesamten Flotte dieses Typs. Ein Sicherheitssystem, das die Wahrscheinlichkeit solcher Ausfälle minimiert, kann durchaus deutlich Produktionskosten aufweisen und dennoch wirtschaftlich sein.

Stabilität des Ladezustands

Bei Strömen im Amperebereich muss darauf geachtet werden, dass der sich ändernde Ladungszustand während einer Messung keinen Einfluss auf das Ergebnis hat. Da bei Lithium-Eisenphosphat-Akkumulatoren der Verlauf der Ruhespannung über dem State of Charge relativ flach im Vergleich zu anderen Batterietechnologien ist, entstehen hier weniger Probleme durch den Einfluss der sich ändernden Ruhespannung. Allerdings weisen auch die Elemente des Ersatzschaltbildes eine Abhängigkeit vom SoC auf, sodass eine geringe Änderung über die Messung anzustreben ist, um für die anschließende Parametrierung von einem einzigen SoC für die Berechnung auszugehen. In der Literatur finden sich verschiedene Angaben darüber, wie viel Änderung des SoC über einen Messzyklus einer EIS zulässig ist. In [33] werden maximal 10%-Punkte als zulässig angegeben, [34] nennt 5%-Punkte als guten Kompromiss zwischen Systemstabilität und zur Verfügung stehender Messdauer.

3 Analyse und Kozeptionierung

In diesem Kapitel wird zunächst das bestehende System aus Zellen-Sensor und Batteriesteuergerät analysiert, um einen Überblick über die vor dieser Arbeit bereits vorhandene Hardware zu geben. Im Anschluss wird auf Grundlage der Erkenntnisse aus Abschnitt 2.2 ein Anforderungsprofil eines Zellen-Sensors für die EIS ermittelt und im abschließenden Abschnitt dieses Kapitels ein Konzept für einen Zellensensor mit erweitertem Leistungsumfang entwickelt.

3.1 Analyse des bestehenden Systems

Für diese Arbeit wurde bei der Entwicklung eines Zellensensors für die EIS auf verschiedene Vorarbeiten zurückgegriffen. Von Phillip Durdaut [8] und Nico Sassano [9] existieren Arbeiten zum Thema Batterie-Zellen-Sensoren der Klasse 3. Ebenfalls von Sassano entstand, als Vorarbeit für seine eigene Masterthesis zeitlich parallel zu dieser Arbeit, eine neue Basisstation, die für alle Tests und Messungen in dieser Arbeit verwendet werden konnte [35].

Eigenschaft	Klasse 1	Klasse 2	Klasse 3
Möglichkeit zur Übertragung	Nur Uplink	Uplink und reduzierter Downlink (WakeUp)	Uplink und vollwertiger Downlink
Empfänger im Sensor	Kein Empfänger	passiver Empfänger (RFID)	aktiver Empfänger
Messbetrieb und Kommunikation	zeit- oder ereignisgesteuert	Aufwecken von extern, dann zeit- oder ereignisgesteuert	Aufwecken und Steuern von extern
Hardwareaufwand	niedrig	aufwendig	sehr aufwendig
Realisierungen	[36] [37] [38]	[39]	[8] [9]

Tabelle 3.1: Einteilung der Zellensensoren des BATSEN-Projektes abgewandelt nach [40]

3.1.1 Zellsensoren mit funksynchroner Messung

Das Forschungsprojekt ‚Drahtlose Zellensensoren für Fahrzeugbatterien' (BATSEN) der Hochschule für Angewandte Wissenschaften (HAW) Hamburg forscht an verschiedenen Sensoren, mit denen neben dem State of Health (SoH) auch der State of Charge (SoC) bestimmt werden soll, der Aufschluss über die im Akkumulator vorhandene Energie gibt. Dazu werden drahtlose Sensoren auf den einzelnen Zellen eingesetzt, welche die jeweilige Spannung, sowie die Temperatur bestimmen können. Zusätzlich ist pro Akkumulator ein sogenanntes Batteriesteuergerät notwendig, das zum einen die Kommunikation mit den einzelnen Zellensensoren führt, zum anderen den Strom durch alle Zellen misst. Dafür wurden drei Klassen von Sensoren definiert, die sich hauptsächlich in ihren Möglichkeiten zur Kommunikation mit dem Batteriesteuergerät unterscheiden. Dabei wird die Kommunikation vom Sensor zum Steuergerät als Uplink bezeichnet, die Kommunikation vom Steuergerät zu den Sensoren als Downlink. Die Sensorklassen sind in Tabelle 3.1 übersichtlich zusammengefasst und werden im Folgenden kurz vorgestellt.

Klasse 1
Bei der Klasse 1 handelt es sich um Sensoren ohne Downlink. Das bedeutet, sie verfügen lediglich über einen Transmitter im 434 MHz ISM-Band und übertragen ihre Messergebnisse nach einem Protokoll, das Kollision mit anderen Sendern dadurch vermeidet, dass große zeitliche Lücken zwischen den Paketen bestehen. In diesen Lücken haben andere Sensoren die Möglichkeit Daten zu übermitteln. Diese Sensoren müssen selbstständig die für das Batteriesteuergerät interessanten Ereignisse erkennen, um diese mit passender zeitlicher Auflösung aufnehmen zu können. Dadurch, dass es keine Synchronisation zwischen Batteriesteuergerät und Sensoren gibt, kann es jedoch zu Datenverlust kommen, da verlorene Pakete nicht wieder angefordert werden können. Diese Sensoren sind mit geringem Hardwareaufwand zu fertigen und weisen einen geringen Energiebedarf auf. Nach Vorarbeit

von Plaschke [41] sind solche Sensoren und die zugehörigen Mess- und Sendeprotokolle Thema der Abschlussarbeiten von Püttjer [36], Kube [37] und Meinzer [38] gewesen.

Klasse 2
Die Klasse 2 beschreibt Sensoren mit WakeUp über einen passiven Empfänger und eingeschränkter Kommunikation über diesen Kanal. Dazu wird zu einem Uplink, der mit der Klasse 1 vergleichbar ist ein Downlink über eine RFID Verbindung realisiert. Da diese in einem anderen Frequenzband arbeiten als der Transmitter, ist eine weitere Antenne samt Receiver notwendig. Dies bedeutet einen zusätzlichen Hardwareaufwand gegenüber der Klasse 1. Im Gegenzug ist es somit möglich, die Sensoren in einen tieferen Energiesparzustand zu versetzen, aus dem sie nur durch einen externen Interrupt aufgeweckt werden können. Dadurch ist eine zusätzliche Möglichkeit der Energieeinsparung, insbesondere bei langen Zeiten ohne interessante Ereignisse (z. B. Lagerung), geschaffen. Außerdem können über diesen Kanal eingeschränkt Befehle an die Sensoren ausgegeben werden. Nach Vorarbeit zum Thema RFID durch Krannich [42] und Eger [43] wurde ein solcher Sensor von Jeggenhorst [39] realisiert.

Klasse 3
Zur Klasse 3 werden solche Sensoren gezählt, die mit einem vollwertigem Downlink ausgestattet sind. Dazu besitzen sie statt eines Transmitters im ISM-Band einen Transceiver. Dadurch ist es möglich die Sensoren zu konfigurieren und die Daten gezielt von einzelnen Sensoren zu empfangen. In der Arbeit von Durdaut [8] ist ein solcher Sensor entstanden. Als Besonderheit besitzt dieser einen Signalverarbeitungspfad, der es ermöglicht, über ein im ISM-Band gesendetes Signal eine WakeUp-Schaltung zu realisieren. Dadurch wird der große Vorteil der RFID-Lösung des Klasse 2 Sensors auch auf diesem Sensor nutzbar, ohne das eine zweite Antenne notwendig wird. In einer weiteren Arbeit hat Sassano [9] den Downlink zum Sensor genutzt um synchronisierte Messungen auf allen Sensoren eines Batteriesteuergerätes zu ermöglichen.

Da für eine EIS eine synchronisierte Messung zwischen Batteriesteuergerät und allen Zellen-Sensoren notwendig ist, erscheint die Klasse 3 als am geeignetsten, um sie im Rahmen dieser Arbeit zu verwenden. Aus diesem Grund wird der aktuelle Stand der Klasse 3 Sensoren nach Sassano im Folgenden vorgestellt. In Abbildung 3.1 ist der Sensor gezeigt und die wichtigsten Komponenten hervorgehoben.

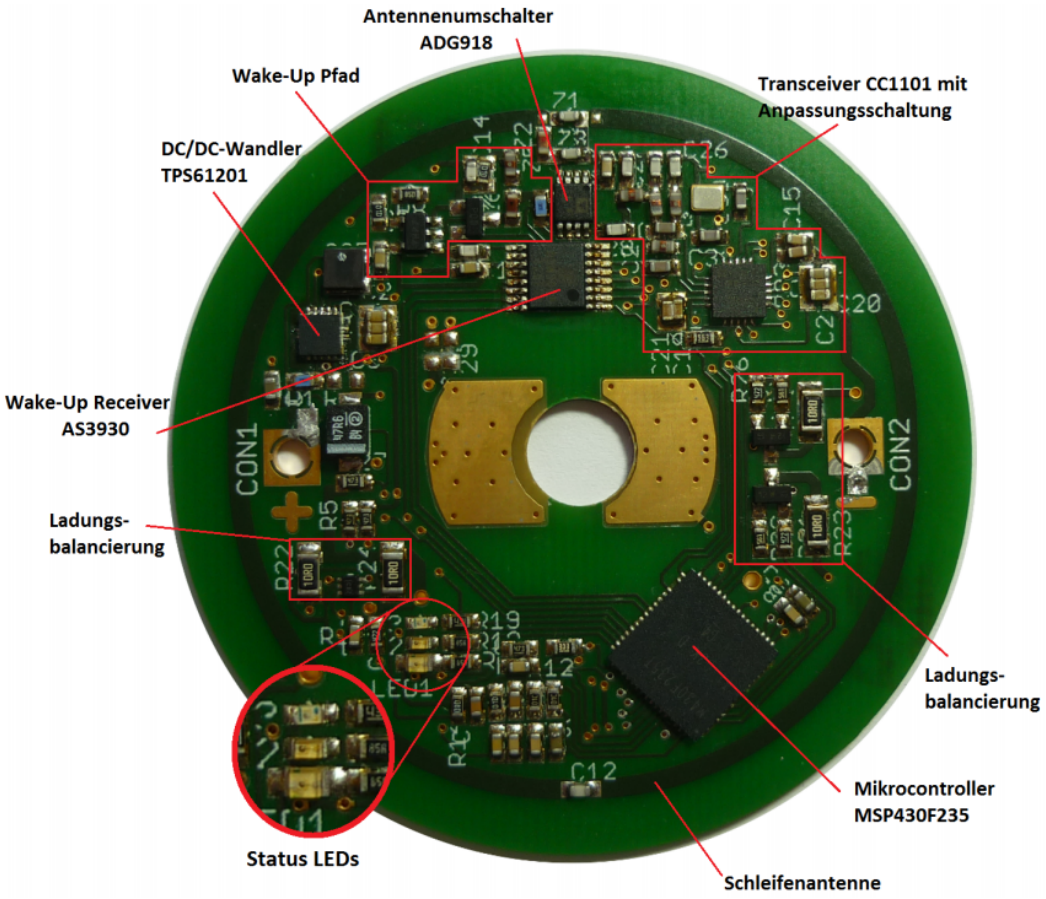


Abbildung 3.1: Zellsensor der Klasse 3 V0.3 nach Sassano entnommen aus [9]

Hardwareanalyse des Zellsensors

Abbildung 3.2 zeigt ein Blockschaltbild der wichtigsten Hardware-Komponenten des Zellensensors Version v0.3, wie sie von Sassano verwendet wurde. Tabelle 3.2 listet die verwendete Hardware auf.

Bauteil	Bezeichnung	Hersteller	Package
Mikrocontroller	MSP430F235	Texas Instruments	QFN-64
UHF Transceiver	CC1101	Texas Instruments	QFN-20
Spannungswandler	TPS61201	Texas Instruments	DFN-10
LF WakeUp Receiver	AS3930	AMS	TSSOP-16
Antennenumschalter	ADG918	Analog Devices	SOP-8
Temperatursensor	TMP102	Texas Instruments	SOT563

Tabelle 3.2: Übersicht Hardwarekomponenten des Zellsensors v0.3

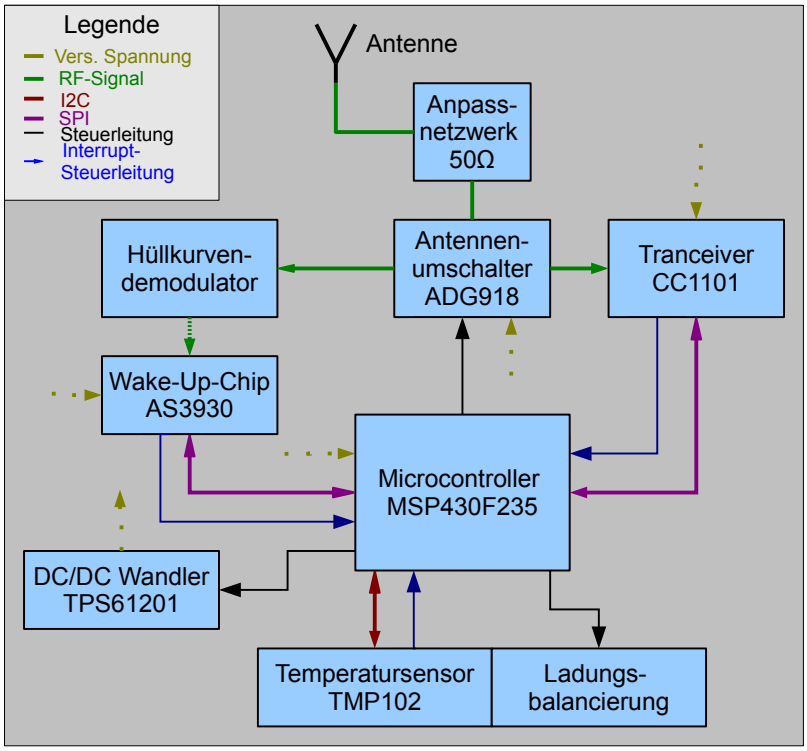


Abbildung 3.2: Blockschaltbild des bestehenden Zellen-Sensors nach Sassano [9]

Mikrocontroller

Es kommt der 16 Bit Mikrocontroller MSP430F235 der Firma Texas Instruments zum Einsatz [44]. Der Controller verfügt über 16 kByte Flash-Speicher, der hauptsächlich als Programmspeicher genutzt wird. Dieser steht jedoch auf Grund der von Neumann-Architektur auch zur Laufzeit als Speicher offen. Wesentlich kürzere Zugriffszeiten erhält man jedoch auf dem 2 kByte großen RAM.

Die Familie der MSP430 besteht aus Mikrokontrollern mit optimierter Leistungsaufnahme und ist daher gut für den Einsatz bei Versorgung aus einer Batterie geeignet. Dazu stehen dem Anwender verschiedenen Low-Power Modi zur Verfügung. Im niedrigstem Modus (LPM4) sinkt die Stromaufnahme bis auf 1 μA. Im Betrieb lässt sich die Leistungsaufnahme reduzieren, indem die Taktrate angepasst wird. Der Controller kommt ohne externen Quarz aus und der interne Digital Controlled Oscillator (DCO) lässt sich zwischen 1 MHz und 16 MHz einstellen.

Für die Kommunikation mit den übrigen ICs auf der Sensorplatine werden sowohl die SPI als auch die I2C Schnittstelle genutzt. Über SPI kommuniziert der Controller mit dem Transceiver, sowohl für die Konfiguration, als auch beim Senden und Empfangen von Paketen. Zusätzlich sind hier noch zwei dedizierte Steuerleitungen am interruptfähigen Eingängen des MSP angeschlossen.

Ebenfalls über SPI ist der WakeUp Chip angeschlossen, der so konfiguriert werden kann. Da das eigentliche WakeUp Signal einen Interrupt im Controller erzeugen muss, wird hierfür ebenfalls eine dedizierte Steuerleitung mit einem Interrupt fähigen Pin verbunden.

Der Temperatursensor ist über die I2C Schnittstelle angebunden und für die Temperaturalarm-Funktionalität ebenfalls zusätzlich mit einem Interrupt-Eingang verbunden.

Der Controller verfügt über zwei 16 Bit-Timer, von denen der erste fünf und der zweite drei Vergleichsregister besitzt. Der erste Timer wird für die Ladungsbalancierung verwendet. Der zweite Sensor wird genutzt, um das Zeitfenster für die Burstmessung zu erzeugen. Der Vorgang ist in Abschnitt 3.1.1 auf Seite 73 näher erläutert.

Für die Spannungsmessung wird auf der aktuellen Zellsensor-Platine der interne 12 Bit ADC des MSP430F235 verwendet. Dieser verfügt über eine interne Referenzspannung von 2.5 V. Um damit die Zellspannung zwischen 2 V und 3.6 V messen zu können, ist ein ½ Spannungsteiler mit zwei 100 kΩ Widerständen realisiert. Damit ergibt sich der Wert eines LSB bei diesem Aufbau zu:

$$\frac{2.5\,\mathrm{V}}{2^{12}} \cdot 2 = \frac{2.5\,\mathrm{V}}{2^{11}} \approx 1.22\,\mathrm{mV} \tag{3.1}$$

Man kann sagen, dass durch den Spannungsteiler und das dadurch erreichte Verdoppeln des Eingangsbereiches die absolute Auflösung des ADC um ein Bit sinkt.

MSP430F235 Übersicht

- 16 Bit-Mikrocontroller
- 16 kByte Flash-Speicher / 2 kByte RAM
- verschiedene Energiesparzustände
- umschaltbarer DCO (1MHz - 16MHz)
- I2C und SPI Schnittstelle
- 2 16 Bit Timer
- 12 Bit A/D Umsetzer
- kein externer Quarz notwendig

UHF-Transceiver

Als Transceiver wurde ein CC1101 von Texas Instruments gewählt [45]. Dieser Transceiver kann durch externe Beschaltung auf einen Frequenzbereich zwischen 315 und 900 MHz abgestimmt werden. Der Transceiver beherrscht verschiedene Modulationsarten, von der in dieser Arbeit lediglich ein binäres ASK-Verfahren (Amplitude-Shift-Keying dt.: Amplitudenumtastung) verwendet wird. Dabei werden die 1 und 0 als verschiedenen Amplituden des Trägers moduliert. Ist bei einer der beiden Zustände der Träger ausgeschaltet, so spricht man auch von On-Off-Keying (OOK). Der Transceiver kann sowohl in einem synchronen als auch einem asynchronen Modus betrieben werden, beim asynchronen Modus wird das empfangene bzw. gesendete Signal binär auf einer der GPIO-Leitungen ausgegeben bzw. von dieser gelesen. Dies wird bei der Burstmessung genutzt, um auf die Flanken des Burstsignals mit einem Port-Interrupt zu reagieren.

Der synchrone Modus wird für den Paketversand genutzt. Dabei erledigt der CC1101 die Verarbeitung der Pakete nach einmaliger Konfiguration selber und liefert lediglich die eigentlichen Daten. Ein Paket besteht aus:

- 2-24 Preamble Bytes, die aus einer alternierenden ’1’ und ’0’ Folge bestehen
- 2 Sync-word-Bytes, die benutzerdefiniert sind,
- optional einem Feld mit der Länge der Nachricht ohne Preamble und Sync-Bytes
- optional einem Feld mit der Adresse des Empfängers
- den Daten aus einer FIFO
- sowie 2 Byte CRC-Checksumme

Die beiden optionalen Bytes werden von uns nicht verwendet, da für die Zwecke der Zellsensoren ein eigener Header innerhalb der Paketdaten die relevanten Informationen enthält. Dieser wird in Abschnitt 3.1.1 auf Seite 73 erläutert. Die Länge der FIFO beträgt 64 Byte für RX und TX zusammen. Das bedeutet, es muss konfiguriert werden, wie viele Bytes jeweils für das Senden und Empfangen zur Verfügung stehen. Allerdings ist es möglich, jeweils vor dem Senden die vollen 64-Byte für die TX-FIFO zu reservieren und anschließend an die RX-FIFO zurückzugeben. Dann kann in dieser Zeit allerdings nicht empfangen werden und für das neue Konfigurieren muss Zeit aufgewendet werden.

Der CC1101 benötigt einen externen 26 MHz Quarz-Oscilator als Referenz für die Erzeugung des Trägersignals.

Sowohl Durdaut als auch Sassano verwenden schematisch das Referenzdesign aus [45] aus konzentrierten Bauteilen. Allerdings sind bei Durdaut Tiefpassfilter und Symetrieglied wie im Referenzdesign in einer Kette angeordnet und stehen über einer Massefläche. Bei Sassano befinden sich alle Bauteile von Symmetrie-Glied und Tiefpassfilter auf einem Haufen, ohne

dass darunter eine Masse verlaufen würde. Dies kann einer der Gründe sein, warum bei Sassano eine deutlich schwächere Eingangsleistung beim Senden und Empfangen erreicht wird als bei Durdaut.

CC1101 Übersicht

- Sende- Empfangsfrequenz 434 MHz
- Übertragungsraten bis 250 kbps
- 64 Byte FIFO-Ringspeicher
- externer 26 MHz Quarz

Spannungswandler

Als Spannungswandler kommt ein TPS61201 von Texas Instruments zum Einsatz [46]. Dieser hat eine feste Ausgangsspannung von 3.3 V und liefert je nach Zellspannung zwischen 700 mA @ 2 V und 1300 mA @ 3.6 V, wobei auch der kleinere Wert mehr als ausreichend ist. Der Wandler bietet außerdem eine einstellbare Abschaltschwelle (Minimum 0.3 V) und einen sog. Power-Save-Mode. Dabei wird der Spannungswandler abgeschaltet, wenn der Ausgangsstrom unter 300 mA sinkt und die Ausgangsspannung über ihrem Nominalwert liegt. Da für unsere Anwendung der aufgenommene Strom fast immer unter 300 mA liegt, kann so zusätzlich Leistung eingespart werden. Allerdings bedingt dies Schwankungen in der 3.3 V Ausgangsspannung.

TPS61201 Übersicht

- Ausgangsspannung 3.3 V
- Eingangsspannungen von 0.3 V bis 5.5 V
- Ausgangsstrom mindestens 700 mA
- Einstellbare Abschaltspannung

LF-WakeUp Receiver

Um aus dem tiefsten Energiesparmodus aufgeweckt zu werden benötigt der MSP430 ein Interrupt-Signal. Dies kann z. B. von einem internen Timer kommen. Dann müsste der Controller überprüfen, ob der Transceiver ein WakeUp Signal empfangen hat und dieses abarbeiten und sich ansonsten wieder in den Energiesparmodus begeben. Dafür muss allerdings der Transceiver im energie-intensiven Empfangsmodus verbleiben und es entsteht eine Verzögerung in Abhängigkeit des Aufwachintervalles.

Daher sind die Sensoren der Klasse 3 mit dem Low-Frequency WakeUp Receiver AS3930 der Firma AMS ausgestattet [47]. Dieser löst einen Interrupt aus, sobald ein Trägersignal zwischen 110 kHz und 150 kHz für mindestens 550 μs erkannt wird. Da die für den Empfang im 434 MHz ISM-Band ausgelegte Antenne bei diesen kleinen Frequenzen kaum noch Leistung absorbiert, wird stattdessen ein Signal mit 434 MHz Trägerfrequenz ausgesendet. Das mit einer Frequenz von 125 kHz ein und ausgeschaltet wird. Dies ist möglich, da der Transceiver CC1101 auf der Basisstation bis zu 250 kBaud Datenrate übertragen kann. Dieses Signal wird über eine Hüllkurvendetektion in ein Signal mit 125 kHz gewandelt. Da die sonst bei der Hüllkurven-Demodulation üblichen Dioden eine Sperrspannung haben, die deutlich über dem zu erwartenden Eingangspegel liegt, kommt dabei eine inkohärente Demodulation mit der Schotky-Diode HSMS-285C zum Einsatz. Diese Diode ist speziell für die Demodulation von kleinsten Eingangssignalen in der RF-Technik ausgelegt und daher geeignet. Das demodulierte Signal wird im Anschluss mit einer aktiven Bandpassschaltung gefiltert und auf den AS3930 gegeben. Für den Bandpass kommt der Operationsverstärker MCP6071 zum Einsatz, der eine sehr geringe Stromaufnahme von nur 110 μA hat.

AS3930 Übersicht

- Arbeitet im Langwellenfrequenzbereich von 110 kHz bis 150 kHz
- WakeUp Frequenz wurde auf125 kHz festgelegt
- gibt ein digitales Ausgangssignal aus
- Konfiguration über SPI

Antennenumschalter
Um sowohl den Transceiver als auch den Hüllkurvendemodulator mit der Schleifenantenne verbinden zu können, ohne dass sich die Eingangsleistung auf beide Pfade aufteilt, wurde ein Antennenumschalter vom Typ ADG918 der Firma Analog Devices eingesetzt [48]. Dieser besitzt eine sehr geringe Stromaufnahme von 1 μA und verbindet, gesteuert durch ein Eingangssignal, die Schleifenantenne mit einem von zwei Ausgängen. Dabei haben Ein- und Ausgänge des ADG918 eine Impedanz von 50 Ω, sodass noch vor dem Umschalter ein Impedanzanpassungs-Netzwerk nötig ist, da die Schleifenantenne aufgrund ihrer Abmessung eine deutlich kleinere Impedanz hat.

ADG918 Übersicht

- Eingangsimpedanz 50 Ω
- Ausgangspfad lässt sich umschalten
- geringer Strombedarf

Temperatursensor
Als Temperatursensor kommt ein TMP102 von Texas Instruments zum Einsatz [49]. Dieser hat zum einen eine sehr kleine Bauform in einem für Transistoren gedachten SOT-Package und hat außerdem eine sehr geringe Stromaufnahme von 15 μA. Zusätzlich bietet er die Möglichkeit einen Temperaturschwellwert zu programmieren, bei dessen Erreichen ein Interrupt-Signal an den Controller gesendet wird. Darüber wird bei der Balancierung sichergestellt, dass die Balancierungswiderstände nicht überhitzen. Intern arbeitet der TMP102 mit einem 12 Bit Δ - Σ-ADC der eine Temperaturauflösung von 0.0625 °C bei 0.5 °C Genauigkeit erreicht.

TMP102 Übersicht

- I2C Anbindung
- 12 Bit Delta-Sigma-A/D Umsetzer zur Erfassung der Temperatur
- 0.0625 °C Temperaturauflösung
- 0.5 °C Genauigkeit
- -25 °C bis +85 °C Temperaturbereich
- 15 μA Stromaufnahme

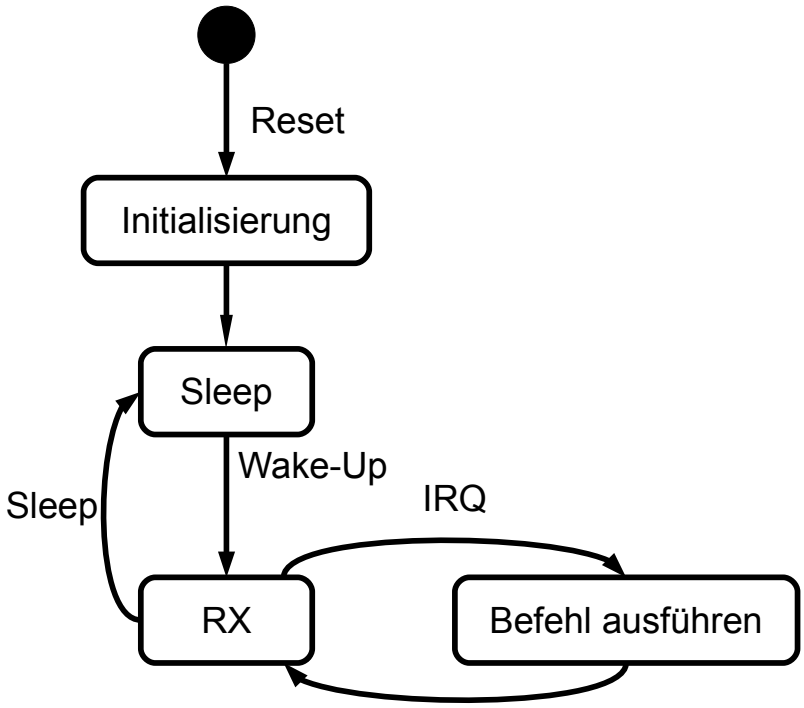


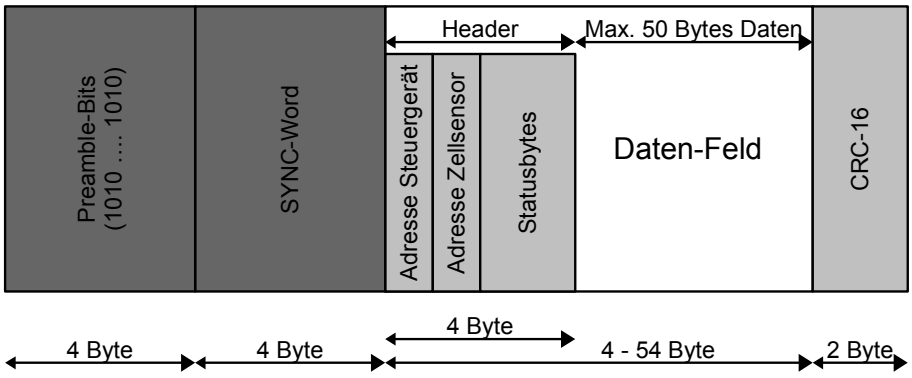
Abbildung 3.3: Zustandsdiagram des Zellsensors nach dem Reset nach [8]

Software des bestehenden Zellsensors

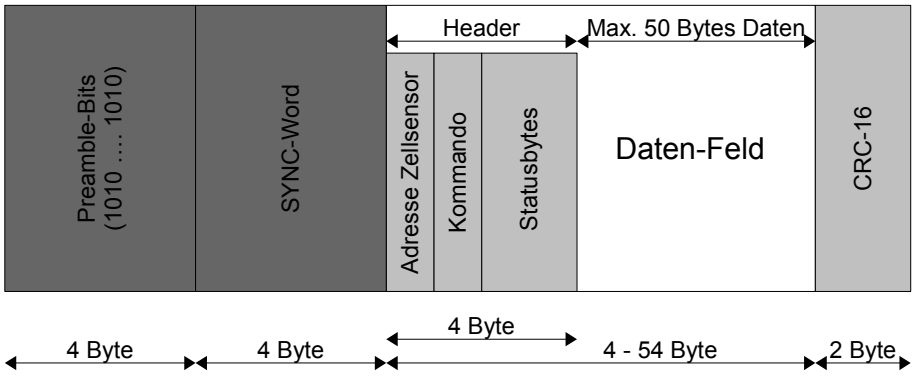
Der Programmablauf nach dem Anlegen der Spannungsversorgung ist in Abbildung 3.3 dargestellt. Zunächst werden alle einmaligen Konfigurationen an der Peripherie des Zellsensors vorgenommen. Hervorzuheben ist, dass der Antennenumschalter auf die Hüllkurvendetektion gestellt werden muss. Im Anschluss geht der Sensor in den sog. 'sleep'-Modus, in dem die Stromaufnahme des Mikrocontrollers unter 1 μA sinkt. Aus diesem 'Low-Power-Mode 4 (LMP4)' genannten Modus wacht der Controller erst auf, wenn der WakeUp-Receiver ein WakeUp-Signal empfängt und über eine Steuerleitung einen Port-Interrupt des Controllers auslöst. Nach dem Aufwachen schaltet der Controller den Antennenumschalter um, sodass der CC1101 mit der Schleifenantenne verbunden ist. Dieser wird dann für den Empfang von Befehlen konfiguriert. Nach der Abarbeitung eines Befehls wird wieder in den Empfangsmodus gewechselt und auf den nächsten Befehl gewartet, solange es sich nicht um den Sleep-Befehl gehandelt hat. Nach einem solchen Befehl wird der Controller in den ‚Low-Power-Mode' versetzt und mit dem Antennenumschalter wird wieder der WakeUp-Pfad mit der Schleifenantenne verbunden.

Paketverarbeitung

Wie im Abschnitt 3.1.1 beschrieben, übernimmt der Transceiver einen großen Teil der Paketverarbeitung bereits automatisch. In der bestehenden Implementierung wird auf die automatische Adressüberprüfung verzichtet. Ebenfalls wird die Länge einer Nachricht nicht mit übertragen. Stattdessen wird ein eigener Header von 4 Byte Länge innerhalb des Datenpaketes versendet. Dieser unterscheidet sich je nachdem, ob es sich um eine Uplink-Nachricht, also eine Sendung vom Sensor zum Batteriesteuergerät oder eine Downlink-Nachricht handelt. Bei einer Uplink-Nachricht enthält das erste Headerbyte die Adresse des Steuergerätes (fest bei 0xFF), das zweite Byte die Adresse des Sensors (bis zu 254 Möglichkeiten: 0x01 - 0xFE). Die übrigen beiden Headerbytes sind aktuell nicht verwendete Statusbytes. Bei Downlink-Nachrichten enthält das erste Byte die Adresse des Zellsensors und die zweite Adresse den auszuführenden Befehl. Auch hier werden die übrigen beiden Headerbytes derzeit nicht verwendet.



(a) Paketformat einer Uplink-Nachricht



(b) Paketformat einer Downlink-Nachricht

Abbildung 3.4: Paketformat des bestehenden Systems

CODE	Bezeichnung	Beschreibung der Reaktion des Sensors
0x00	WAKEUP	Sensor soll wach bleiben.
0x01	WAKEUP_DONE	Sleep wenn kein WAKEUP empfangen wurde.
0x02	IS_AWAKE	Mit IS_AWAKE antworten.
0x03	SAMPLE_VOLTAGE	Zellspannung aufnehmen
0x04	SEND_VOLTAGE	Zellspannung versenden
0x05	SLEEP	In Energiesparmodus wechseln.
0x08	BALANCING_ON	Balancierung bis angegebenen Spannungswert
0x09	BALANCING_ON FF	Balancierung ausschalten
0x0A	SEND_VOLTAGE _TEMPERATURE	TMP102-Temperatur und Zellspannung aufnehmen
0x0B	SAMPLED_VOLTAGE _TEMPERATURE	TMP102-Temperatur und Zellspannung aufnehmen
0x0C	BURST_MODE	Burstmessung starten (.vgl Abschnitt 3.1.2)
0x0D	BURST_DATA_RQ	Burstdatenversand vorbereiten (.vgl Abs. 3.1.2)
0x0E	BURST_DATA_RX	Frame der Burstdaten übertragen (.vgl Abs. 3.1.2)
0x10	CONFIG_SET	Empfang für TMP102 Config-Daten vorbereiten
0x11	CONFIG	TMP102 mit empfangenen Daten einstellen
0x12	SAMPLE_TEMPERATURE_TMP102	TMP102-Temperatur aufnehmen
0x13	SEND_TEMPERATUREe_TMP102	TMP102-Temperatur versenden
0x14	SAMPLE_TEMPERATURE _TMP102_CALI	unkalibrierte TMP102-Temperatur zur Berechnung eines neuen Kalibrierwertes aufnehmen
0x15	SEND_TEMPERATURE _TMP102_CALI	unkalibrierte TMP102-Temperatur zur Berechnung eines neuen Kalibrierwertes versenden
0x16	SAMPLE_TEMPERATURE_MSP430	MSP430-Temperatur aufnehmen
0x17	SEND_TEMPERATURE_MSP430	MSP430-Temperatur versenden
0x18	SAMPLE_TEMPERATURE _MSP430_CALI	unkalibrierte MSP430-Temperatur zur Berechnung eines neuen Kalibrierwertes aufnehmen
0x19	SEND_TEMPERATURE _ MSP430_CALI	unkalibrierte MSP430-Temperatur zur Berechnung eines neuen Kalibrierwertes versenden
0x1A	CALIBRATION_TMP102	Temperaturkalibrierung für TMP102 setzen
0x1B	CALIBRATION_MSP430	Temperaturkalibrierung für MSP430 setzen
0x1C	CALIBRATION_ADC	Spannungskalibrierung für MSP430 setzen
0x1D	SEND_VOLTAGE _TEMPERATURE_ALL	TMP102-Temperatur, MSP430-Temperatur und Zellspannung aufnehmen
0x1E	SAMPLE_VOLTAGE _TEMPERATURE_ALL	TMP102-Temperatur, MSP430-Temperatur und Zellspannung versenden

Tabelle 3.3: Liste der Kommandos zur Steuerung der Zellsensoren

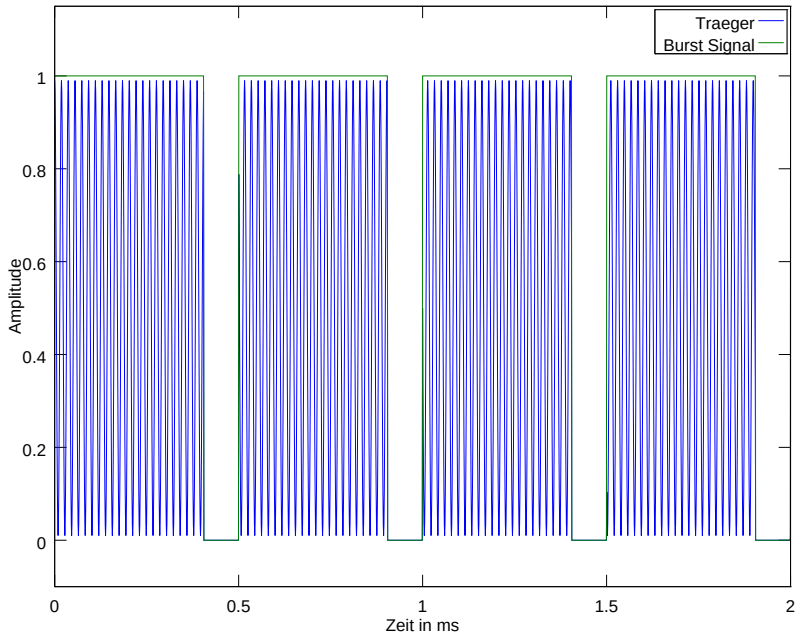


Abbildung 3.5: Burst-Signal für die Messwertsynchronisation. Dargestellt ist eine Burst Messung mit 2 kHz. Das Frequenz des Trägersignals ist um den Faktor 4000 verkleinert um den Träger sichtbar zu machen.

Burstmessung

Wie Tabelle 3.3 auf der vorherigen Seite zeigt, gibt es eine Reihe von Befehlen zur Steuerung der Burstmessung. Da diese Messart die für eine Impedanzspektroskopie notwendige synchronisierte Messung ermöglicht, wird der Vorgang der Burstmessung hier vorgestellt.

Um die Messung zu synchronisieren sendet die Basisstation ein Trägersignal, das periodisch für 95 μs unterbrochen wird. Ein beispielhaftes Signal ist in Abbildung 3.5 dargestellt. Es wird deshalb mit kleiner negativer Puls-breite gearbeitet, da bei kurzen Trägerimpulsen und damit langen Zeiten ohne Trägersignal die Automatische Verstärker Steuerung (Automatic Gain Controll AGC), die Eingangsschwelle soweit verschiebt, dass Rauschen als Signal erkannt wird.

Die Sensoren sollen dabei auf die fallende Flanke dieses 'Burst' genannten Signals reagieren, indem sie eine Messung starten. Da es auch bei kleiner negativer Pulsbreite noch zu Störungen im Burstsignal zwischen zwei Messzeitpunkten kommen kann, wird der Interrupt, der die fallende Flanke erkennt, zeitabhängig freigeschaltet. Dabei wird, ausgehend vom zuletzt erkannten Messzeitpunkt, eine Timer-Steuerung realisiert, die ein 400 μs breites Zeitfenster öffnet. Dieses Zeitfenster liegt mittig um den ausgehend von der bekannten Messfrequenz nächsten erwarteten Messzeitpunkt. Die Breite der Interrupt-Freischaltung ergibt sich

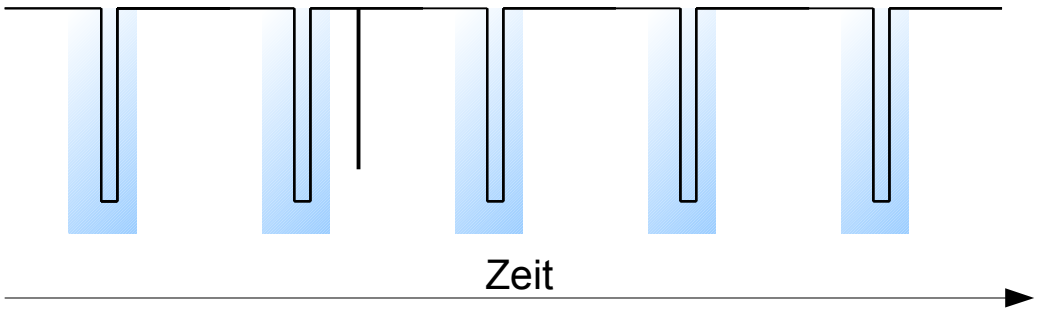


Abbildung 3.6: Die Abbildung zeigt die Interrupt-Freigabe des Burst Signals. In den Blau markierten Bereichen wird eine Fallende Flanke des Burstsignals als Startzeitpunkt für die Messung erkannt. Der dargestellte Fehler im Signal wird dagegen ausgeschlossen.

dabei aus der angegebenen Ungenauigkeit des digital gesteuerten Oszillators innerhalb des Mikrocontrollers, der bis zu 6 % betragen kann. Aus der Breite der Freischaltung ergibt sich ebenfalls das für Messfrequenzen über $1/400\,\mathrm{us} = 2.5\,\mathrm{kHz}$ ein Aussperren fehlerhafter Signale nicht mehr möglich ist. Hier wird demnach immer empfangen. Abbildung 3.6 zeigt das Freischalten der Interrupts und das Ausschließen von Störungen im Burst-Signal.

Wird ein Signal erkannt, so wird zunächst der Interrupt für die Flankenerkennung gesperrt und anschließend die Messung durchgeführt. Danach wird ein Timer eingestellt, der dafür sorgt, dass das Zeitfenster wieder geöffnet wird. Dabei muss von der erwarteten Zeit zwischen zwei Messzeitpunkten $1/f_{Mess}$ zunächst die halbe Zeit der Fensteröffnung $t_{on}/2$ abgezogen werden. Außerdem muss noch die Zeit subtrahiert werden, die bereits für das Abarbeiten der Interrupt-Service Routine (ISR) für die Messung verbraucht wurde. Abbildung 3.7 auf der nächsten Seite zeigt den Zusammenhang zwischen diesen Zeiten. Beim aktuellen Zellensensor sind diese Zeiten konstant, 400 μs für die Fensteröffnung und 92 μs für die ISR. Für verschiedene Messfrequenzen sind unterschiedliche Vorteiler für das Taktsignal des Timers nötig, um mit den 16 Bit breiten Vergleichsregistern die benötigten Zeiten bei maximal möglicher Genauigkeit zu bestimmen. Daher ist auch für die oben genannten festen Zeiten ein frequenzabhängiger Wert einzustellen, der sich jedoch im Vorfeld berechnen lässt.

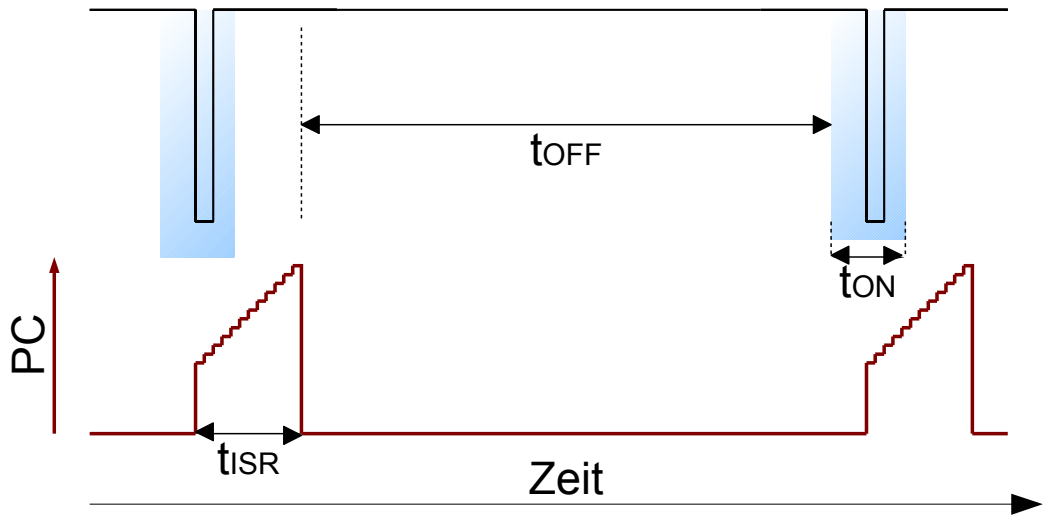


Abbildung 3.7: Berechnung der Timereinstellung für das Zeitfenster. Die Verzögerung durch die ISR ist in Rot in Form des fortlaufenden Programm Counters (PC) dargestellt. Da die ISR eine feste Laufzeit hat, lassen sich alle Zeiten im Vorfeld ermitteln.

3.1.2 Basisstation

Die Basisstation ist Teil der Arbeit [35] und basiert auf einem sog. Launchpad für den Mikrocontroller 'Tiva C Series TM4C129' der Firma Texas Instruments (TI). Als Launchpad bezeichnet TI traditionell Entwicklungsplattformen für die eigenen Mikrocontroller. Diese bieten den Vorteil, dass sie bereits mit notwendiger Peripherie, wie z. B. Quarz, Steckverbinder, Programmierinterfaces und einfachen Bedien- und Anzeigeelementen ausgestattet sind. Dadurch kann zum Beginn einer Entwicklung die Funktion des Mikrocontrollers evaluiert werden, ohne bereits kostenintensive Prototypen produzieren zu müssen.

Die Launchpads der TIVA Serie bietet eine definierte Schnittstelle aus zwei doppelreihigen Pfostenleisten mit definierter Geometrie und Pinbelegung, die für sog. BoosterPacks vorgesehen sind. Das verwendete Launchpad bietet zwei dieser Anschlüsse mit je 40-Pins. Für einen dieser Booster-Pack Anschlüsse hat Sassano eine Platine entworfen und gefertigt, die mit einem CC1101 RF-Transceiver und der notwendigen Peripherie bestückt ist. Über eine USB-zu-RS232 Umsetzung erfolgt die Kommunikation zwischen einer beliebigen Terminal-Software auf einem Personal-Computer und dem Launchpad. Über SPI kann dann das Launchpad über den Transceiver im 434MHz Band mit den Sensoren kommunizieren. Dazu besteht die Möglichkeit die in Tabelle 3.3 auf Seite 75 genannten Befehle per direkter Eingabe der Kommando-Bytes zu senden. Außerdem sind Funktionen für die Koordination der Burstmessung vorhanden, die das dort notwendige Timing realisieren. Dazu dienen die folgenden beiden Funktionen:

burst(frequency,values) Diese Funktion erwartet als Parameter die Frequenz, mit der die Burstmessung erfolgen soll, sowie die Anzahl der aufzunehmenden Messwerte. Dabei müssen beide Werte aus einer Reihe vorgegebenen Menge ausgewählt werden, da diese bereits auf den Sensoren hinterlegt sind (vgl. Tabelle 3.4 auf der nächsten Seite). Die nicht fortlaufende Codierung der Frequenzen basiert auf dem späteren Hinzufügen höherer Messfrequenzen, nachdem die Grenzen des Sensors getestet wurden. Die Funktion sendet zunächst einen Broadcast-Befehl an alle Sensoren, in denen es die angegebene Frequenz und die Zahl der aufzunehmenden Messwerte mitteilt. Dies ist notwendig, damit alle Sensoren die Timer für die Interrupt-Freischaltung initialisieren, sowie den Speicherplatz für die Messwerte aloziieren können.

Nach einer Wartezeit, welche die Sensoren für die Konfiguration benötigen, sendet daraufhin die Basisstation das in Abbildung 3.5 auf Seite 76 gezeigte Signal für die Synchronisation der Messung.

get_burst(sensorAdress) Die Funktion erwartet die Adresse des Sensors, dessen Burst Daten empfangen werden sollen. An diesen Sensor sendet das Kommando BURST_DATA_RQ, woraufhin der Sensor mitteilt, ob Daten zum Versand gespeichert sind. Außerdem legt der Sensor die Zahl der zu sendenden Frames und die Anzahl der Messwerte Pro Frame fest und teilt dies der Basistation in seiner Antwort mit. Diese Information nutzt die Basisstation um im Anschluss mit dem Kommando BURST_DATA_RX die Frames nacheinander abzurufen und über die RS232 Schnittstelle an den PC zu senden.

Die get_burst()-Funktion nacheinander für jeden Sensor aufgerufen werden, dessen Daten ausgelesen werden sollen. Durch das Auslesen werden die Daten auf den Sensoren nicht gelöscht. Sie werden erst überschrieben, wenn ein neuer burst()-Aufruf empfangen wird. Sie werden auch dann überschrieben, wenn sie noch nicht ausgelesen wurden.

Frequenz [Hz]	HEX-Wert
50	0x14
100	0x13
150	0x12
200	0x11
250	0x10
300	0x0F
350	0x0E
400	0x0D
450	0x0C
500	0x0B
550	0x0A
600	0x09
650	0x08
700	0x07
750	0x06
800	0x05
850	0x04
900	0x03
950	0x02
1000	0x01
2000	0x16
4000	0x15
6000	0x17
8000	0x18
10000	0x19

Messwerte	Hex-Wert
50	0x01
100	0x02
150	0x03
200	0x04
250	0x05
300	0x06
350	0x07
400	0x08
450	0x09
500	0x0A
550	0x0B
600	0x0C
650	0x0D
700	0x0E
750	0x0F
800	0x10
850	0x11
900	0x12
1000	0x13
1500	0x14
1900	0x15

Tabelle 3.4: Mögliche Burst-Frequnezen und Messwertanzahl sowie zugehörige Hex-Werte

3.2 Anforderungen der Impedanzspektroskopie an den Zellsensor

In diesem Abschnitt soll geklärt werden, welche Anforderungen eine aufwandsreduzierte Elektrochemische Impedanzspektroskopie an einen Zellensensor nach BATSEN-Bauart stellt. Dazu hat Angold im Rahmen von [6] bereits Untersuchungen angestellt. Dabei ging es vor allem um die Festlegung von Mindestanforderungen und die Implementierung einer Anregeschaltung mit günstigen Bauteilen. Auf Basis dieser Erkenntnisse soll im Folgenden festgelegt werden, welche Anforderungen an den weiter zu entwickelnden Zellensensor für die Elektrochemische Impedanzspektroskopie gestellt werden.

3.2.1 Phasenauflösung

Die angestrebte Phasenauflösung der EIS-Messergebnisse wurde durch das Projekt mit 10 ° festgelegt [50]. Daraus resultiert, dass mindestens 36 Abtastwerte für jede Periode des Anregungssignals der EIS gemessen werden müssen, um diese Auflösung zu ermöglichen.

3.2.2 Synchronisierte Messung

Da die derzeit maximal mögliche Messrate der Burstmessung bei 10 kHz liegt, kann, mit der im letzten Abscchnitt festgelegten Phasenauflösung, die maximale zeitliche Asynchronität $\pm\Delta t$ festgelegt werden:

$$\Delta t < \frac{10\,^\circ}{10\,\mathrm{kHz} \cdot 360^\circ} = 2.8\,\mu\mathrm{s} \tag{3.2}$$

Die Synchronität wird bei der Burstmessung durch das Öffnen des Zeitfensters realisiert. Die Fensteröffnung muss mindestens $\pm 3\%$ der Periodendauer der Burstfrequenz betragen, da der interne Oszillator eine maximale Abweichung in dieser Größenordnung besitzt. Aktuell ist das Fenster bei allen Burstmessungen 400 μs breit, was der maximalen Abweichung bei einer Messfrequenz von 300 Hz entspricht. Um eine Synchronität auch bei höheren Messfrequenzen zu erreichen, muss die Fensterbreite hier angepasst werden.

3.2.3 Amplitudenauflösung

In [6] und [7] wird dargestellt, dass es notwendig ist, die Amplituden der bei der EIS verwendeten Spannungsschwankungen gegenüber der Zellengleichspannung klein zu halten, um die näherungsweise Linearität im Kleinsignalverhalten auszunutzen. Wählt man die Amplituden dieser Signale zu groß, so verfälschen die Nichtlinearitäten des Systems das Ergebnis.

Angold gibt an, das Amplituden im Bereich 1 mV bis 5 mV pro Zelle geeignet sind. Daraus ergibt sich die Anforderung der Amplitudenauflösung. Da schon jetzt davon ausgegangen werden kann, dass der Zellsensor weiterhin eine 3.3 V Spannungsversorgung behalten wird, scheint sichergestellt, dass auch das neue Messverfahren einen 2:1 Spannungsteiler vor dem ADC behalten wird. Damit sinkt die Kleinsignalamplitude am Eingang des ADCs auf den Bereich 500 μV bis 2.5 mV.

Die kleinste aufzulösende Spannung des neuen Messsystems sollte daher um mindestens eine Größenordnung kleiner sein. Als Arbeitsgrundlage werden zwei anzustrebende Spannungsauflösungen V_{LSB} definiert:

- V_{LSB1} = 10 μV
- V_{LSB2} = 100 μV

Im ersten Fall würden die Kleinsignale mit 50 bis 250 Stufen, also ca. 5.5 Bit bis 8 Bit Auflösung des Kleinsignals aufgelöst werden. Im zweiten Fall verbleiben zwischen 5 und 25 Stufen respektive 2.33 Bit bis 4.6 Bit.

Da die Genauigkeit der späteren Impedanzauflösung der EIS abhängig ist von der Auflösung der Spannungs- und Strommessung, ist der erste Fall hier deutlich vorzuziehen. Dennoch wird im Folgenden auch betrachtet, welche Voraussetzungen erforderlich sind, um den zweiten Fall zu erfüllen, mit dem unter Umständen ein brauchbares Ergebnis der EIS durch Mittlung mehrerer Ergebnisse erzielt werden kann.

Im Folgenden wird bestimmt, welche Auflösung ein ADC haben muss, um die definierten Auflösungen des Kleinsignals zu erreichen, wenn das Gesamtsignal aus Zellgleichspannung und EIS-Signal gemessen werden soll. Dabei wird davon ausgegangen, dass der neue ADC eine Referenzspannung von 2.5 V besitzt wie der ADC der aktuellen Sensorgeneration. Dann ergibt sich die benötigte Auflösung zu:

$$\begin{aligned} N_1 &= log_2\left(\frac{V_{REF}}{V_{LSB1}}\right) \\ &= log_2\left(\frac{2.5\,\mathrm{V}}{10\,\mathrm{uV}}\right) \\ &= log_2\,(250000) \\ &\approx 17.9\,\mathrm{Bit} \end{aligned} \tag{3.3}$$

$$\begin{aligned} N_2 &= log_2\left(\frac{V_{REF}}{V_{LSB2}}\right) \\ &= log_2\left(\frac{2.5\,\mathrm{V}}{100\,\mathrm{uV}}\right) \\ &= log_2\,(25000) \\ &\approx 14.6\,\mathrm{Bit} \end{aligned} \tag{3.4}$$

3.3 Messverfahren für kleine Wechselspannungen mit großem DC-Offset

3.3.1 Direkte hochauflösende Messung

In Abschnitt 3.2.3 auf Seite 82 wurde festgestellt, dass zum Auflösen der Zellspannung auf 10 μV knapp 18 Bit erforderlich sind. Diese Auflösung lässt sich nicht mehr mit den für Mikrocontroller üblichen Analog Digital Umsetzern mit sukzessiver Approximation (SAR) erreichen. Bei diesen wird mittels eines Digital-Analog-Umsetzers (DAC) zunächst die halbe Referenzspannung auf einen Komparator gegeben, an dessen anderem Eingang eine Sample-and-Hold Schaltung liegt, die den Messwert über den Konvertierungsvorgang auf einem Niveau hält. Zeigt der Komparator an, dass die Eingangspannung höher ist als die Ausgangsspannung des DAC, so wird die DAC-Spannung um ein Viertel der Referenzspannung erhöht, andernfalls um ein Viertel der Referenzspannung abgesenkt (dies geschieht durch Setzen des nächst niederwertigeren Bits und Beibehalten oder Löschen des aktuellen Bits im SAR). Im nächsten Schritt geschieht dasselbe mit einem Achtel der Referenzspannung usw. . Daraus wird ersichtlich, dass diese Art des ADC ‚N' Schritte zum Konvertieren benötigt, wobei ‚N' die Bitbreite des Ausgangssignals ist. Mit steigender Wort-Breite nimmt die Wandlungszeit zu, sodass üblicherweise ADCs bis 12 Bit in dieser Technik realisiert sind.

Deutlich höhere Auflösungen erreicht man mit einem sogenannten Delta-Sigma-Umsetzer (Δ-Σ-ADC). Abbildung 3.8 auf der nächsten Seite zeigt das Schema eines Δ-Σ-Umsetzers erster Ordnung. Der Kern dieses ADCs ist ein Quantisierer, der als 1 Bit ADC verstanden werden kann. Seine 1 Bit breiter Ausgang weist dem Eingangswert entweder die minimale oder die maximale Ausgangsspannung zu. Somit entsteht ein großer Quantisierungsfehler, der bis zum halben Eingangsspannungsbereich groß sein kann (vgl.: [51]). Dadurch ist das entstehende Quantisierungsrauschen relativ groß.

Der Quantisierungsfehler wird ermittelt, indem der quantisierte Wert durch einen 1 Bit DAC wieder in eine Spannung gewandelt wird. Diese Spannung, die sozusagen den Schätzwert der letzten Messung enthält, wird vom nächsten Eingangswert subtrahiert. Der Subtrahierer findet als das Delta (Δ) Einzug im Namen des Umsetzers. Über diesen Schätzfehler wird integriert (entspricht im zeitdiskreten Fall der Summe (Σ)). Durch die Integration wird die Leistung des Quantisierungsrauschens zu höheren Frequenzen hin verschoben, es geschieht eine sog. Rauschformung. Dies ist wichtig, da das Quantisierungsrauschen durch die geringe Anzahl der Stufen ansonsten erheblichen Einfluss auf das Ausgangssignal hätte.

Im Falle eines Δ-Σ-ADCs sind 1 Bit ADC und DAC die einzigen getakteten Bauteile. Der 1 Bit ADC kann als Komparator verstanden werden, der mit der halben Eingangsspannung vergleicht, der DAC als Umschalter zwischen der maximalen und minimalen Ausgangsspannung. Durch ihren einfachen Aufbau lassen sich diese Bauteile sehr schnell takten. Als

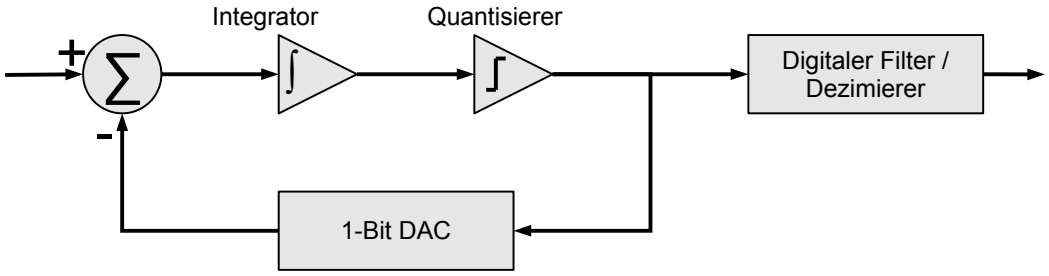


Abbildung 3.8: Schema eines Δ-Σ-Analog-Digital-Umsetzers.

Ausganssignal der Δ-Σ-Stufe kann der Bitstrom verstanden werden, den der Quantisierer ausgibt. Dieses Signal ist mit der Abtastfrequenz f_s von Quantisierer und 1 Bit DAC getastet und enthält noch das volle Quantisierungsrauschen, allerdings verschoben zu hohen Frequenzen. Der Takt der Δ-Σ-Stufe ist meistens deutlich schneller als das aufzuzeichnende Signal, dessen Frequenz als f_D anzunehmen ist. Daher ist es möglich, das quantisierte 1 Bit Signal über einen Tiefpass zu filtern und die Taktrate zu dezimieren. Dadurch werden die großen Anteile des Quantisierungsrauschens bei hohen Frequenzen ignoriert und der Signal-Rauschabstand steigt. Dabei ist der Anteil des Rauschens, das im Ausgangssignal verbleibt, abhängig davon, über wie viele 1 Bit Werte gemittelt wird. Diese Anzahl wird als Dezimierungsrate bezeichnet.

Abbildung 3.10 auf der nächsten Seite zeigt eine Schematische Darstellung der Rauschanteile im Ausgangssignal für zwei verschiedene Dezimierungsraten. Dadurch wird deutlich, dass mit steigender Dezimierungsrate die Genauigkeit des Ausgangssignals wächst. Als Kennzahl wichtiger, als die absolute Höhe des Quantisierungsrauschens, ist die Anzahl der rauschfreien Bits ((ENOB) Effektive Number of Bits). Es ist üblich, dass Δ-Σ-ADCs mit N-Bit Auflösung diese n-Bit Auflösung nicht oder nur bei der größten Dezimierungsrate erreichen. Daher muss für die jeweils verwendete Dezimierungsrate die tatsächliche Anzahl der rauschfreien Bits aus dem Datenblatt entnommen und mit den Anforderungen verglichen werden.

Das Tiefpassfilter vor der Dezimierung kann im einfachsten Fall eine Mittelung über alle zwischen zwei dezimierten Ausgangswerten liegenden Eingangswerten sein. Allerdings kann mit besseren Filtern eine zusätzliche Rauschunterdrückung der hohen Frequenzen erreicht werden. Prinzipiell kann jedes digitale Filter zum Einsatz kommen, jedoch ist das am häufigsten anzutreffende ein FIR-Filter als sog. Sync-Filter [52]. Bei diesem Filter wird die Impulsantwort eines idealen rechteckigen Tiefpassfilters abgetastet. Die Impulsantwort des Rechtecks ist die Si-Funktion, im englischen Sinc-Funktion genannt. Durch das Abtasten verliert das Filter seine idealen Eigenschaften, die jedoch umso besser nachgebildet werden, je mehr Stützstellen das Filter aufweist. Dadurch wird deutlich, dass für eine gute Rauschunterdrückung lange Verzögerungen des FIR-Filters in Kauf genommen werden müssen.

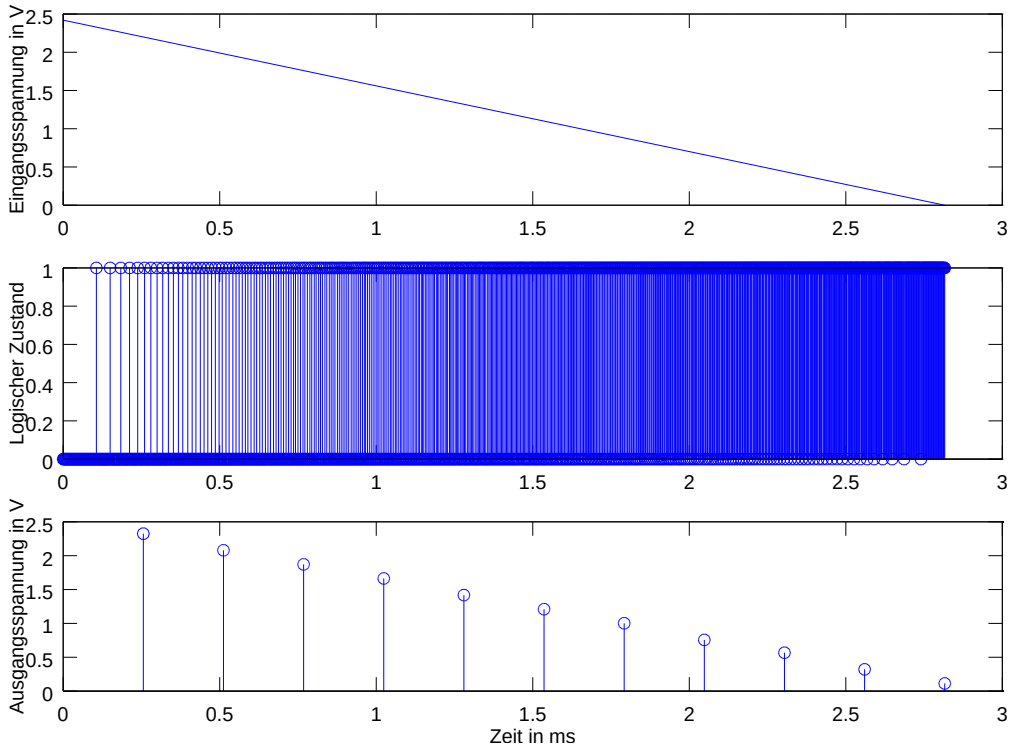


Abbildung 3.9: Zustand eines Δ-Σ-Umsetzers in verschiedenen Stufen. Die oberste Grafik zeigt ein Eingangssignal, das mit 500 [kHz] abgetastet wird. Im zweiten Bild sieht man, dass mit sinkender Eingangsspannung die Zahl der Schätzwerte, die der minimalen Ausgangsspannung entsprechen, zunehmen. Im letzten Bild ist, für eine Oversamplingrate von 128 und Mittelung als Tiefpassfilter, das Ausgangssignal dargestellt.

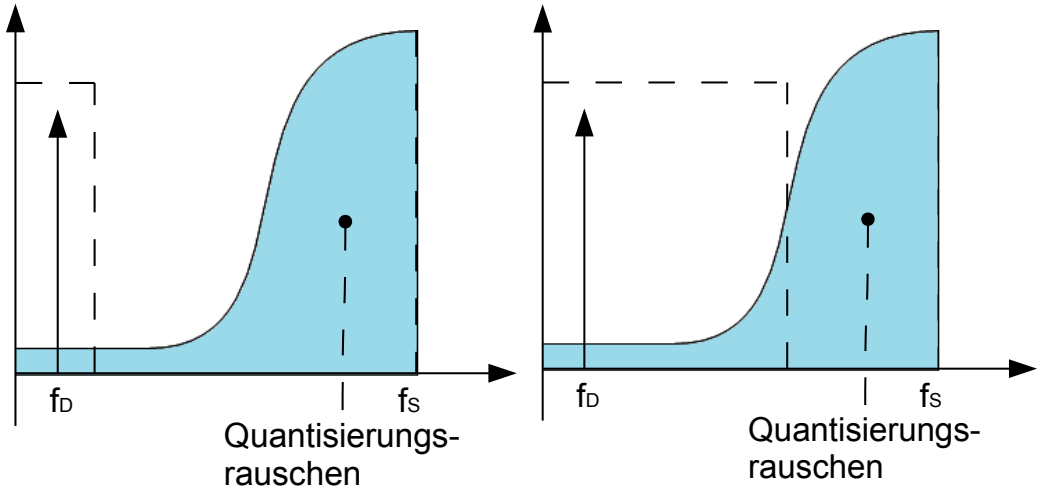


Abbildung 3.10: Einfluss der Dezimierungsrate auf das Rauschen (vgl. [52]).

3.3.2 Offsetkorrektur

Im Folgenden wird nun erörtert, wie Eigenschaften von $LiFePO_4$-Akkumulatoren und EIS genutzt werden können, um hier mit weniger hochauflösenden ADCs arbeiten zu können.

Da die Entladeschlussspannung einer $LiFePO_4$-Zelle bei etwa 2.6 V, die Ladeschlussspannung bei 4 V liegt, sind die außerhalb dieses Bereiches liegenden Spannungen für den Zellsensor vollkommen uninteressant. Daher besteht die Möglichkeit den Bereich der Zellspannung durch eine Offsetkorrektur um 2.6 V auf den Bereich 0 V bis 1.4 V abzubilden. Hier wäre eine Verwendung des internen 12 Bit ADC des MSP430, ohne Spannungsteiler möglich, wodurch sich die Spannungsauflösung von jetzt ca. 1.2 mV auf 600 μV verbessern würde. Dies unterscheidet sich noch deutlich von denn in den Gleichungen 3.3 und 3.4 geforderten 10-100 μV. Würde man die verbleibenden 1.4 V Spannungshub so verstärken, dass sie den vollen 2.5 V Eingangsspannungsbereich eines handelsüblichen ADCs ausnutzen, so läge die notwendige Spannungsauflösung noch bei:

$$10\,\mu\text{V} \cdot \frac{2.5\,\text{V}}{1.4\,\text{V}} \approx 17.9\,\mu\text{V}$$

Daraus resultiert eine notwendige Bitzahl von:

$$\begin{aligned} N_{Bit} &= log_2\left(\frac{U_{REF}}{U_{LSB}}\right) \\ &= log_2\left(\frac{2.5\,\text{V}}{17.9\,\text{uV}}\right) \\ &\approx 17\,\text{Bit} \end{aligned}$$

Es wäre nur 1 Bit einzusparen im Vergleich zur Messung ohne Offset-Korrektur.

Aus den Erkenntnissen zur EIS kann der Umstand ausgenutzt werden, dass für die EIS nicht die Gesamtspannung der Batterie von Interesse ist, sondern lediglich der durch den eingeprägten Strom entstehende Wechselanteil. Dieser liegt im Bereich 1 mV bis 5 mV pro Zelle. Wenn also der gesamte Gleichanteil der Batterie ausgekoppelt werden könnte, so würden für die Abtastung des Wechselsspannungssignals ein ADC mit folgender Auflösung notwendig sein:

$$\begin{aligned} N_{Bit} &= log_2\left(\frac{U_{Max}}{U_{LSB}}\right) \\ &= log_2\left(\frac{5\,\mathrm{mV}}{10\,\mathrm{uV}}\right) \\ &= log_2\,(500) \\ &= 8.97\,\mathrm{Bit} \end{aligned}$$

Hier würden also die bereits auf dem aktuellen Zellsensor vorhandenen 12 Bit ADCs mehr als ausreichen. Für eine Genauigkeit von $10\,\mu\mathrm{V}$ kann ein 12 Bit-ADC
$10\,\mu\mathrm{V} \cdot 2^{12} = 40.96\,\mathrm{mV}$
überdecken. Dies ließe genug Spielraum für eine gewisse Änderung der Zell-Ruhespannung, ohne die Offsetkorrektur anpassen zu müssen. Da Referenzspannungen im mV-Bereich unüblich sind, müsste eine Verstärkerschaltung eingesetzt werden, welche die Amplitude des Wechselspannungs-Signals auf die mögliche Eingangsspannung des ADCs abbildet. Solche Schaltungen bringen zusätzliche Nichtlinearitäten in das Messsignal und sind immer in gewissem Maße temperaturabhängig.

Das Auskoppeln des Gleichspannungsanteil kann theoretisch auf zwei Arten geschehen.

- Durch **analoge Filterung** kann der gesamte Gleichanteil ausgekoppelt werden. Für die EIS scheint diese Methode wenig geeignet zu sein, da hier extrem niedrige Frequenzen bis in den mHz Bereich betrachtet werden müssen. Daraus resultiert, dass die Grenzfrequenz des analogen Hochpassfilters, das den Gleichanteil auskoppelt deutlich unter 1 mHz liegen muss. Dies würde Bauteile erfordern, die zum einen für eine Integration in einen kleinen Zellsensor zu groß sind. Des Weiteren würden durch die vollständige Auskopplung des Gleichanteils auch negative Spannungen im Eingangssignal des ADCs entstehen, die von den internen ADCs des MSP430 nicht gemessen werden können.
- Durch **adaptive Offsetkorrektur** ließe sich das Problem negativer Spannungen umgehen, indem nicht die gesamte Ruhespannung korrigiert wird, sondern der Korrekturwert um die maximal erwartete Amplitude des Wechselspannungsanteils verringert wird. Dann lägen alle Spannungen im positiven Bereich. Zu diesem Zweck muss die Ruhespannung der Zelle gemessen werden, was zusätzlichen Hardware- und Messaufwand bedeutet.

Das analoge Auskoppeln des Gleichanteils kommt aufgrund der niedrigen Messfrequenzen nicht in Frage. Eine mögliche Beschaltung für das Auskoppeln eines vorzugebenden Spannungsoffset ist die sogenannte Spannungslupe, die in Abbildung 3.11 auf der nächsten Seite dargestellt ist.

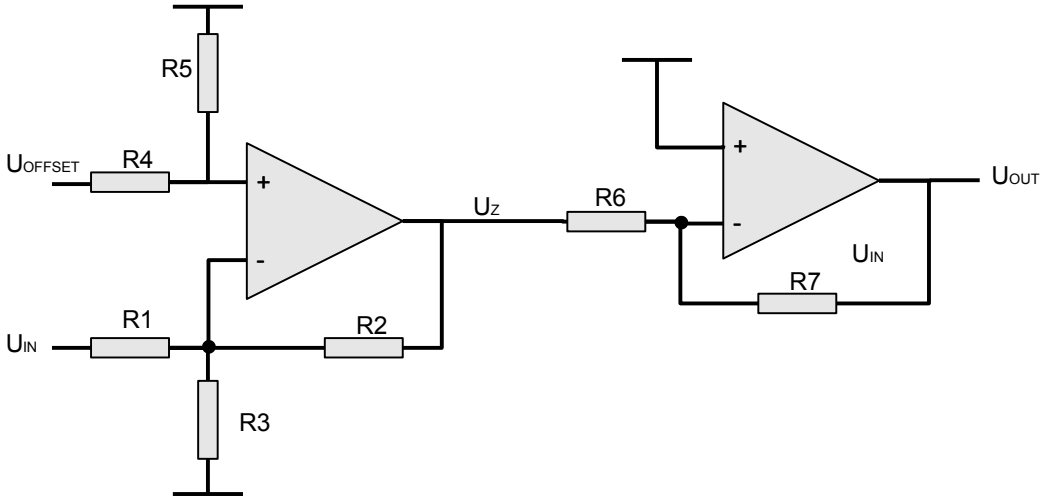


Abbildung 3.11: Schaltplan einer Spannungslupe. Die erste Operationsverstärkerschaltung hat einen Verstärkungsfaktor von -1 und Kompensiert die Eingangsspannung um einen über U_{Offset} einstellbaren Spannungswert. Die zweite Operationsverstärkerschaltung verstärkt das übrig bleibende Signal so, dass es den gesamten zur Verfügung stehenden Eingangsbereich von 0 V bis 2.5 V abdeckt.

In Anhang F ist dargestellt, wie mit einer Operationsverstärkerschaltung die Auskopplung eines Gleichanteil bei gleichzeitiger Verstärkung des Wechselanteils so gestaltet werden kann, dass mit einem 12 Bit ADC die Aufnahme eines 40 mV breiten Spannungsbereiches mit der geforderten Auflösung von 10 μV möglich wird. Ein Nachteil dieser Lösung ist, dass durch die Verwendung reeler Bauteile Nichtlinearitäten entstehen können. Insbesondere ist die Temperaturstabilität zu beachten. Außerdem sind beide Operationsverstärkerschaltungen invertierend, weshalb negative Versorgungsspannungen benötigt werden.

Tabelle 3.5 auf der nächsten Seite fasst den notwendigen Aufwand für verschiedene Optionen der Messung des Wechselspannungssignals, das durch die EIS erzeugt wird, zusammen. Das Hochpassfilter ist aufgrund der niedrigen Frequenzen, die gemessen werden sollen nicht geeignet. Das Abziehen eines Offsets in Höhe der Entladeschlussspannung bietet nur geringen Vorteil für die benötigte Auflösung, sodass der zusätzliche Schaltungsaufwand nicht gerechtfertigt ist. Es verbleiben als Möglichkeit die hoch aufgelöste Messung des gesamten Signals mit mindestens 18 Bit Auflösung oder die aufwändige einstellbare Offsettkorrektur mit anschließender Verstärkung, um eine ausreichende Spannungsauflösung mit 12 Bit ADC zu gewährleisten. Im zweiten Fall ist der Schaltungsaufwand nicht unerheblich, außerdem kann eine solche Schaltung frequenz- und temperaturabhängige Nichtlinearitäten in den Messpfad einbringen. Zusätzlich muss die Zellenspannung noch einmal unkorrigiert mit kleinerer Auflösung gemessen werden, um die abzuziehende Offsetspannung zu ermitteln.

	Direkte Messung	DC-Filter	Spannungslupe fest	Spannungslupe variabel
Schaltungs-aufwand passiv	Spannungs-teiler	Filter, Span-nungsteiler	-/-	(Tiefpass s.u.)
Schaltungs-aufwand aktiv	-/-	Messverstärker	Differenz-verstärker, Offsetspan-nungsquelle	Differenz-verstärker
Anforderungen μC	-/-	-/-	-/-	Ruhespannungs-messung, DAC (PWM mit Tief-pass)
Auflösung ADC	18 Bit	12 Bit	17 Bit	12 Bit

Tabelle 3.5: Vergleich von Methoden zur Messung kleiner Wechselgrößen bei großem Offset

Da der derzeitig verwendete Mikrocontroller nicht über einen echten Digital-Analog Wandler verfügt und dieser auch bei den zukünftig vorgesehenen Controllern nicht vorhanden ist, müsste die Ausgabe der Offsetkorrektur mittels tiefpassgefilterter PWM erfolgen. Da jedes Tiefpassfilter eine endliche Dämpfung bei der PWM-Frequenz aufweist, könnten so Störungen in das Messsignal gelangen.

Letztlich fällt die Entscheidung für eine hochaufgelöste Messung des gesamten Signals, neben den Nachteilen der anderen Methoden auch, weil so die gesamte Information des Signals gewonnen werden kann. Teile dieser Informationen sind eventuell für die EIS verzichtbar, können aber z.B. bei der Vorverarbeitung des Signals helfen, wenn zum Beispiel die Spannungsdrift korrigiert werden soll. Des Weiteren ist der Sensor mit einer hoch auflösenden Messung des Gesamtsignals auch in der Lage eine genauere Messung für bereits implementierte und zukünftige Verfahren bereitzustellen. Tabelle 3.5 zeigt, dass für dieses Vorhaben ein ADC verwendet werden muss, der mindestens 18 Bit Auflösung besitzt. Solche Auflösungen lassen sich in kostengünstiger Ausführung nur mit Δ-Σ-ADC realisieren. Im folgenden Abschnitt wird daher ein geeigneter ADC gesucht und ein Konzept erstellt, mit diesem eine EIS-Messung durchzuführen.

3.4 Konzeption der Impedanzspektroskopie mit drahtlosen Zellsensoren

3.4.1 Auswahl eines geeigneten Δ-Σ-ADCs

Für die Auswahl eines passenden ADCs lassen sich aus den bisherigen Erkenntnissen und Vorarbeiten eine Reihe von Anforderungen erstellen:

- Die **Sample-Rate** ergibt sich aus zwei Faktoren. Zum einen den Anforderungen der EIS. In kommerziellen Laborgeräten für die Elektrochemische Impedanzspektroskopie werden Frequenzen von wenigen mHz bis einigen kHz gemessen. Aus [9] wissen wir, dass die Zellsensoren der Klasse 3 im synchronisierten Messbetrieb bis zu 10 kHz Messdatenrate erreichen können. Da an der Funktionsweise des Messbetriebes keine Anpassung vorgenommen werden sollen, bleibt dies als Obergrenze der Messrate bestehen. Für die EIS ergeben sich dadurch Einschränkungen des darstellbaren Impedanzspektrums.
- Für eine synchronisierte Messung ist es unerlässlich, dass die Sensoren auf allen Zellen ihre Spannungsmessung gleichzeitig starten. Dieser Startzeitpunkt muss mit dem Zeitpunkt der Strommessung im Batteriesteuergerät synchronisiert werden können. Daher ist es wichtig jede Messung manuell anzustoßen, um den Startzeitpunkt festzulegen. Da Δ-Σ-ADC für die Filterung und Dezimierung des Bitstroms verschiedene digitale Filter verwenden, kommt es zu Unterschieden im Einschwingverhalten, wenn erst beim manuellen Anstoßen der Messung die Konvertierung in einen Messwert beginnt. Bei ADCs, bei denen kontinuierlich konvertiert wird und bei denen das Anfordern eines Messwertes lediglich die Ausgabe ansteuert, sind Sample-Rate und Datenrate identisch. Wenn jedoch zunächst die digitalen Filter einschwingen müssen, wird die mögliche **Daten-Rate**, die ein ADC erzeugen kann, begrenzt.
- Die **Auflösung** ist nach Gleichung (3.3) auf Seite 83 zu 18 Bit bzw. 14 Bit festgelegt. Diese Auflösung muss bei der maximalen Daten-Rate auch noch erreicht werden. Wie oben gezeigt, sinkt die Auflösung eines Δ-Σ-ADC mit sinkendem Dezimierungsfaktor, also mit steigender Sample-Rate. Daher muss betrachtet werden, welche Auflösung bei der gewünschten Rate noch erreicht wird.
- Als **Interface** muss eine Kommunikationsmöglichkeit existieren, die eine einfache Anbindung an den Mikrocontroller ermöglicht, die mit wenigen Datenleitungen auskommt. Hier kommen SPI und I^2C in Frage.
- Da der ADC vom Zellsensor Klasse 3 versorgt werden soll, muss der Energie-Umsatz gering gehalten werden. Dazu gehört, dass die **Versorgungsspannung** vom Regler

des Zellsensors übernommen werden soll, um Verluste in einem zusätzlichen Spannungsregler zu vermeiden. Daher soll der Sensor bei 3.3 V Versorgungsspannung arbeiten können.

- Um Platz und Kosten einsparen zu können und das Design simpel zu halten, sollte der ADC bereits über eine **interne Referenzdpannung** und eine **interne Takterzeugung** für die Messung verfügen.

Tabelle 3.6 auf der nächsten Seite zeigt eine Auswahl von 24 Bit Δ-Σ-ADCs, die zumindest Teile der gestellten Anforderungen erfüllen. Allen gemein ist, dass sie über SPI kommunizieren, sodass diese Eigenschaft keinen Eingang in die Tabelle gefunden hat. Es ist ersichtlich, dass kein ADC alle gestellten Anforderungen erfüllen kann. Sehr problematisch ist, dass nur ein einziger Sensor bei den geforderten 3.3V Betriebsspannung betrieben werden kann. Allerdings ergeben sich auch bei den anderen Anforderungen Probleme. So gibt es nur wenige Umsetzer, die mit interner Referenzspannung und Takterzeugung ausgestattet sind.

Als die zwei besten Optionen erscheinen:

- Der **AD7176-2** von Analog Devices, der sehr gute Eigenschaften hat, aber 5 V Spannungsversorgung benötigt [53].
- Oder der **ADS1291** von Texas Instruments, der als einziger gefundener Wandler mit 3.3 V versorgt werden kann. Dieser kann lediglich bis zu 1.7 kSPS abtasten und liefert die gewünschten 18-bit bei maximal 500 SPS und die geforderten 14-bit bei maximal 2 kSPS [54].

Trotz der Tatsache, dass der ADS1291 die gestellten Forderungen nicht vollständig erfüllen kann, fällt die Entscheidung für diesen Baustein und gegen den AD7176-2. Das Einführen eines zweiten DC/DC-Wandlers würde zum einen dem Konzept widersprechen, die Zellsensoren effizient zu designen, da sie die Energie der Batteriezelle nicht unnötig verbrauchen soll. Außerdem entstehen durch den aktuell verwendeten Boost-Konverter TPS61201 bereits beträchtliche Störungen in der Zellspannung. Dies wird in der aktuellen Form umgangen, indem der TPS61201 für den Zeitpunkt der Messung abgeschaltet wird. Da Δ-Σ-ADCs ohne Sample-and-Hold Glied auskommen, müsste der Spannungswandler für den gesamten Zeitraum abgeschaltet werden. Dies ist jedoch nicht möglich, da Δ-Σ-ADCs mit einer Überabtastung über längere Zeiträume die Messung durchführen. Daher muss jeder notwendige Spannungswandler eingeschaltet bleiben und die Störungen durch einen zweiten sind nicht zu vernachlässigen.

Da das Ziel dieser Arbeit der Nachweis einer möglichen Elektrochemischen Impedanzspektroskopie mit drahtlosen Zellsensoren ist und nicht die Entwicklung eines marktreifen Produktes wird an dieser Stelle akzeptiert, dass mit dem gewählten Sensor keine vollständige EIS erfolgen kann. Der eingeschränkte Frequenzbereich sollte dem Nachweis der Funktionsfähigkeit nicht im Wege stehen.

Bezeichnung	Hersteller	nom. Auflösung	nom. Sample-Rate [kSPS]	Auflösung @ 10 kHz	Data Rate [kHz]	V_{supply} [V]	int. V_{REF} [V]	int. MCLK [MHz]
LTC2440	LT[a]	24 Bit	4	17 Bit @4 kHz	4	4.5 - 5.5	n/a	9
AD7765	AD[b]	24 Bit	156	n/a	156	4.8 - 5.5	n/a	n/a
AD7766	AD	24 Bit	128	n/a	2	2.4 - 2.6	n/a	n/a
AD7176-2	AD	24 Bit	250	19 Bit	50	5	2.5	16
AD7762	AD	24 Bit	625	n/a	250	5	n/a	n/a
AD7732	AD	24 Bit	15.4	15 Bit	12	5	n/a	n/a
AD1155	AD	24 Bit	16	n/a	16	4.8 - 5.2	n/a	n/a
ADS1210	TI[c]	24 Bit	16	10 Bit @1 kHz	16	4.8 - 5.2	2.5	n/a
ADS1251	TI	24 Bit	20	19 Bit	20	4.8 - 5.2	n/a	n/a
ADS1252	TI	24 Bit	40	19 Bit	20	4.8 - 5.2	n/a	n/a
ADS1259	TI	24 Bit	14.4	>19.6 Bit	1.78	4.8 - 5.2	2.5	7.3
ADS1271	TI	24 Bit	105	n/a	1.45	4.8 - 5.2	n/a	n/a
ADS1291	TI	24 Bit	8	17 Bit @1 kHz	1.69	2.7 - 3.6	2.5	0.512

Tabelle 3.6: Gegenüberstellung verschiedener Δ-Σ-ADCs - Relevante Auswahlkriterien sind entsprechend iher Anforderungserfüllung farbig markiert.

[a] Linear Technologie
[b] Analog Devices
[c] Texas Instruments

3.4.2 Integration der hochauflösenden Messung

Der ausgewählte Sensor kommt aufgrund der internen Referenzspannung und Takterzeugung mit relativ wenigen passiven externen Bauteilen aus. Dennoch werden insgesamt zehn Kondensatoren für das Abblocken der Spannungsversorgung, das Stabilisieren der Referenzspannung und zum Stabilisieren nicht genutzter Eingänge benötigt. Außerdem ein 1:2 Spannungsteiler, der es ermöglicht, mit der internen Referenzspannung von 2.42 V bis zu 4.84 V. Spannung zu messen. Der ADC selber kommt in einem TQFP32 Package. Betrachtet man Abbildung 3.1 auf Seite 65 so wird klar, dass es nicht ohne Weiteres möglich ist, diese zusätzlichen Bauelemente auf einem neuen Design des Zellsensors unterzubringen, das alle Eingenschaften des alten beibehält.

Daher wurde entschieden, für den Δ-Σ-ADC eine eigene Platine zu entwerfen und diese über eine Steckverbindung auf einem funktional unveränderten Zellensensor der Klasse 3 zu positionieren. Eine solche Aufsteckplatine wird im Folgenden Erweiterungsmodul genannt. In [55] wurden bereits Erfahrungen mit dem nachträglichen Einbringen einer Platine mit zusätzlicher Funktionalität auf ein bestehendes Sensorsystem gesammelt. In dem Fall wurde ein Zellensensor der Klasse 1 mit einem Erweiterungsmodul ausgestattet, das mit optischen Messmethoden den Säuregehalt des Elektrolyten in einer Bleibatterie bestimmt. Sensor und Erweiterungsmodul sind in Abbildung 3.12 dargestellt.

Abbildung 3.12: Beispiel eines Erweiterungsmoduls auf einem Zellensensor der Klasse 1 nach [55]

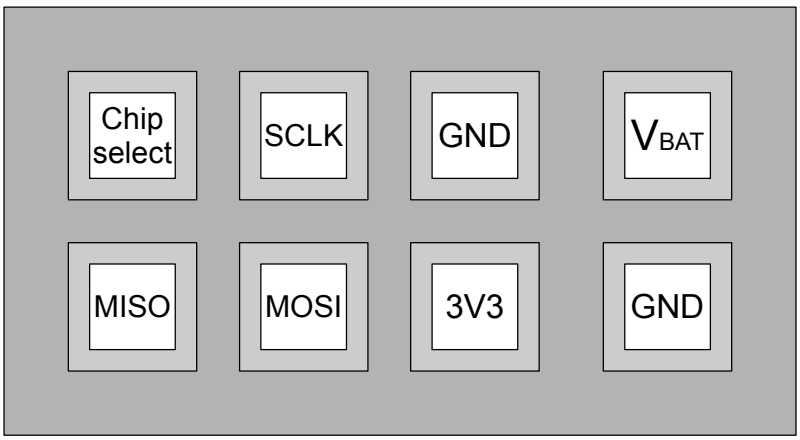


Abbildung 3.13: Pinbelegung des Pfostensteckers für das Erweiterungsmodul in der Draufsicht.

Die Kommunikation zwischen ADS1291 und dem Zellensensor erfolgt über SPI. Dazu werden 4 Signale benötigt. Neben den drei SPI-Signalen MISO MOSI und SCLK wird noch ein Chip-Select Signal benötigt, da auch der Transceiver CC1101 und der WakeUp-Chip AS3930 über SPI kommunizieren. Außerdem wird für die Versorgung des Erweiterungsmoduls die 3.3 V Versorgungsspannung und ein Rückleiter nach Ground benötigt. Abschließend muss noch eine Verbindung zur zu messenden Zellenspannung hergestellt werden. Diese Verbindung sollte eine möglichst geringe Impedanz aufweisen, um Störung und Verfälschung des Signals zu minimieren. Dazu sollte zum einen die Verbindung zwischen positivem Batteriepol auf dem Zellensensor und dem Erweiterungsmodul-Stecker kurz und breit sein. Zum anderen ist es wünschenswert, hochfrequente Signalleitungen von dieser Messleitung fern zu halten. Um dies zu gewährleisten, wird zu den sieben genannten Verbindungen eine zweite Masseverbindung hinzugefügt, sodass eine Pfostenleiste mit 2x4-Pins verwendet werden kann, die entsprechend Abbildung 3.13 angeordnet sind.

Durch das Anordnen der Batteriespannung zwischen den zwei Masseleitungen und der relativ konstanten Spannungsversorgung soll verhindert werden, dass durch die SPI Schnittstelle Störungen in die zu messende Spannung einkoppeln.

Ein weiterer Vorteil, der durch die Entwicklung eines Zellensensors der Klasse 3 mit Anschlussmöglichkeit für ein Erweiterungsmodul entsteht, ist die Wiederverwendbarkeit dieser Schnittstelle. Sollte sich zukünftig ein ADC finden, der besser geeignet ist, so wäre lediglich das relativ einfache Erweiterungsmodul zu ersetzen, auf der zusätzlich ausreichend Platz ist.

Auf Grund der Tatsache, dass alle MSP430 die SPI Schnittstelle mit einer sog. Universal Synchronous/Asynchronous Receiver Transmitter (USART) Schnittstelle realisieren, ergibt sich ein noch breiteres Feld von Einsatzmöglichkeiten, da über dieselben physikalischen Leitungen auch eine I^2C oder UART Verbindung aufgebaut werden kann.

In beiden Fällen lassen sich die dann nicht mehr benötigten Kommunikationsleitungen als I/O-Pins oder für Sonderaufgaben verwenden, da die MSPs mit einer zur Laufzeit programmierbaren Port-Map ausgestattet sind, die es ermöglicht, viele logischen Funktionen mit fast jedem physikalischen Pin zu verbinden. In Anhang H wird etwas ausführlicher auf die Möglichkeiten der Erweiterungsmodul-Schnittstelle eingegangen.

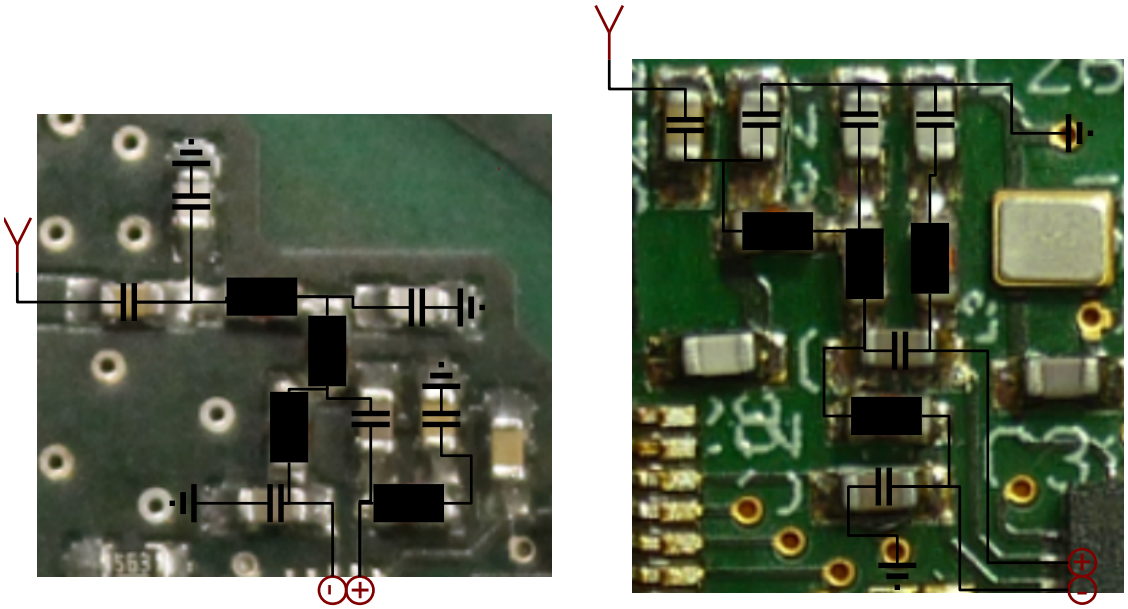
3.4.3 Anpassungen am Zellsensor Klasse 3

Um das geplante Erweiterungsmodul realisieren zu können, müssen einige Anpassungen am bestehenden Sensor der Klasse 3 vorgenommen werden. Abbildung 3.1 auf Seite 65 zeigt, dass auf der Version v0.3 nur sehr wenig Platzreserven vorhanden sind. Daher muss nach Wegen gesucht werden, Platz einzusparen, um den notwendigen Pfostenstecker mit 2x4-Pins auf der Platine unterbringen zu können. Außerdem zeigte die Version v0.3 des Klasse 3 Zellensensors ein deutlich schlechteres Sende- und Empfangsverhalten, als die Version v0.1 aus [8]. Daher werden Überlegungen angestellt, worin die Ursache für dieses Problem liegen kann.

RF-Beschaltung der Antenne

Durch das generell vollkommen unterschiedliche Layout ist es verständlich, dass das Anpassnetzwerk der Schleifenantenne von Sensorversion v0.1 nicht mehr für den Sensor v0.3 gilt. Daher wurde wie schon beim ersten Sensor eine Messplatine angefertigt, mit der die S-Parameter der Antenne bestimmt werden können, um, mit Hilfe des Programms CST-Studio, die Werte für ein Anpassnetzwerk für einen Wellenwiderstand von 50 Ω berechnen zu lassen.

Diese Anpassung findet vor dem Antennenumschalter statt. Der Transceiver benötigt zusätzlich zu einem Antennensignal mit 50 Ω Wellenwiderstand noch einen Tiefpass oberhalb der Trägerfrequenz und einen sogenannten Balun. Dieses Bauteil erhält seinen Namen aus dem Kofferwort für Balanced/Unbalanced und beschreibt ein Symmetrie-Glied, das aus dem massebezogenen Antennensignal ein differentielles Signal erzeugt, das auf den differentiellen RF-Eingang des Transceivers gegeben werden kann. Da Reichweite und Fehlerhäufigkeit der WakeUp-Schaltung bei beiden Sensorversionen identisch sind, ist davon auszugehen, dass die verminderte Leistungsfähigkeit des neueren Sensors in der Realisierung von Tiefpass und Balun liegt. Abbildung 3.14 auf der nächsten Seite zeigt jeweils einen Ausschnitt des Layouts der beiden Baugruppen dieser Sensoren. Man erkennt, dass v0.1 eine Massefläche sowohl im Top-Layer als auch im Bottom-Layer verwendet und so die Impedanzen niederohmig an Masse anschließen kann. Außerdem erkennt man, dass hier der Vorgabe des Referenzdesign nicht nur schematisch entsprochen wird, sondern dass das Layout sich



(a) Zellsensor Klasse 3 Version v0.1

(b) Zellsensor Klasse 3 Version v0.3

Abbildung 3.14: Layout des RF-Pfades auf den bisherigen Zellensensoren. Man erkennt rechts, dass kurze Wege für die Anbindung des Transceivers gewählt wurden. Ausßerdem hat dieser Sensor Masseflächen in beiden Layern und somit eine kurze Anbindung an GND. Links erkennt man, dass der Anschluss des differentiellen Signals hinter dem Balun über zwei Leitungen realisiert ist, die deutlich in der Länge voneinander Abweichen.

eng an die vorgegebene Ausrichtung der Bauteile hält. Zusätzlich sind alle Verbindungen möglichst kurz ausgeführt. Das differentielle Signal am Ausgang des Baluns wird über eine parallele Doppelleitung geführt, wobei beide Leitungen die gleiche Länge haben.

In Version v0.3 wurde vollständig auf eine Massefläche im Bottom-Layer verzichtet, um den Einfluss auf die Antenne durch die Minimierung von Kupfer auf der Platine gering zu halten. Daher erfolgt die Masseanbindung der Bauelemente von Tiefpass und Balun hier über dedizierte Masseleitungen, die nicht unterhalb der entsprechenden Elemente verlaufen und auch nicht im Top-Layer und Bottom-Layer übereinstimmen. In Bezug auf die differentielle Signalführung hinter dem Balun stellt man fest, dass hier zwei Leitungen deutlich unterschiedlicher Länge verwendet wurden.

Da der Einfluss des veränderten Layouts der RF-Komponennten nicht quantifiziert werden kann, wird im Folgenden auf Grundlage einfacher Überlegungen festgelegt, wie das Layout der RF-Komponenten im zu entwerfenden Sensor aussehen soll.

- Es wird weiterhin davon ausgegangen, dass ein Großteil der Abweichung der Antennenparameter vom in [8] theoretisch bestimmten Wert, durch die Kupferflächen innerhalb der Platine entsteht. Daher wird auch weiterhin keine großflächige Masse im Top-Layer oder Bottom-Layer vorgesehen. Allerdings reagiert das RF-Signal aufgrund seiner hohen Frequenz von 434 MHz empfindlicher auf Änderungen des Wellenwiderstandes der Leitung. Daher soll gezielt für den Transmissionsweg dieses Signals eine **Massefläche** vorgesehen werden.
- Die Anbindung der Bauteile an die Masse soll niederohmig und für jedes Bauelement gezielt über Vias zur Masse im Bottom-Layer erfolgen, anstatt über Masse-Leitungen im Top-Layer gebündelt abgeführt zu werden.
- Die Platzierung der Bauelemente soll möglichst dicht und entsprechend der Vorgabe im Referenz-Schaltkreis und **Referenzlayout** erfolgen.
- Die differentielle Signalleitung hinter dem Balun soll kurz und mit identischer Länge erfolgen.

Weiterhin zeigt ein Vergleich der Zellensensorversionen v0.1 und v0.3, dass bei der späteren Version aus Platzgründen Bauteile und Leitungen deutlich dichter an der Schleifenantenne platziert sind. Dies kann ebenfalls zum Verstimmen der Schleifenantenne führen. Aus diesem Grund soll in dem neu zu entwerfenden Zellensensor auf mehr Abstand zwischen der Schleifenantennen und den Bauteilen und Leitungen geachtet werden.

3.4.4 EMV Anfälligkeit langer Leitungen

Vergleicht man die beiden Sensor-Layouts im Allgemeinen, so stellt man fest, dass bei v0.1 generell alle Leitungen relativ kurz gehalten werden konnten. Durch die große Öffnung in der Mitte zum Einbau auf den LiFePO-Zellen müssen bei Version v0.3 alle Bauelemente um einen Kreis herum angeordnet werden, was generell die Leitungsführung deutlich erschwert. Das Layout zeigt mehrere Leitungen, beispielsweise die Masseleitung oder die Leitung zwischen den Balancierungswiderständen und die Versorgungsspannung, die mehr als die Hälfte des Sensors umspannen. Dabei erreichen diese Leitungen geschätzte Längen zwischen 10 cm und 17 cm. Die Wellenlänge des RF-Signals liegt im Vakkum bei:

$$\lambda = \frac{c_0}{f_{RF}} \approx \frac{300 \cdot 10^6 \,\mathrm{m/s}}{434 \cdot 10^6 s^{-1}} \approx 70\,\mathrm{cm} \tag{3.5}$$

Damit liegen die langen Leitungen auf der Sensorplatine v0.3 im Bereich $\lambda/8$ - $\lambda/4$ womit sie parasitäre Antennen darstellen können. Diese können Teile der Signalleistung des gesendeten oder empfangenen RF-Signals aufnehmen, wodurch Empfangs- und Sendeleistung verschlechtert werden.

Da auch beim Redesign die Möglichkeit erhalten bleiben soll den Sensor auf eine Zelle aufzuschrauben, wird es nicht immer möglich sein, auf eine kurze Leitungsführung zu achten. Daher sollen bei allen Leitungen, welche die Gesamtlänge von $\lambda/8$ überschreiten, Drosseln in Form konzentrierter Bauelemente in den Leitungspfad eingefügt werden. Diese sollen für das RF-Signal bei 434 MHz eine hochohmige Barriere aufbauen und so die Leitung in Abschnitte unterteilen, die kleiner als $\lambda/8$ sind. Dadurch soll erreicht werden, dass diese verkürzten Leitungsstücke keine guten Antenneneigenschaften für das RF-Signal mehr haben. Sollte diese Regelung Signalleitungen mit hochfrequenten Anteilen wie etwa eine SPI Leitung betreffen, so muss im Einzelfall betrachtet werden, wie der Einfluss einer Drossel auf die Signaleigenschaft wirkt. Wichtig für diese Drosseln ist es, Induktivitäten mit geringem Serienwiderstand zu verwenden, wenn über die Leitung Leistung transportiert werden soll.

3.4.5 Zusammenfassen von Mikrocontroller und Transceiver

Die bisher beschriebenen Änderungen sind eher dazu geeignet, zusätzlichen Platz auf der Platine zu verbrauchen, als welchen einzusparen. Um dennoch den nötigen Raumgewinn für die Platzierung der Pfosten-Leiste für das Erweiterungsmodul zu erzielen, wird in diesem Abschnitt die Möglichkeit geprüft, den Mikrocontroller MSP430F235 und den Transceiver CC1101 zusammenzufassen. Texas Instruments bietet mit der CC430-Familie eine Reihe von Mikrocontrollern auf Basis eines MSP-430 an, die über einen integrierten sogenannten Radio-Core verfügt, der die Eigenschaften eines CC1101 nachbildet [56].

Die CC430-Mikrocontroller gibt es in zwei Baugrößen mit 64 oder 48 Pins. Tabelle 3.7 auf der nächsten Seite vergleicht die Eigenschaften von dediziertem Mikrocontroller und Transceiver mit den jeweils leistungsstärksten Varianten der beiden CC430 Baugrößen. Zunächst ist festzustellen, das beide Mikrocontroller bei den vorhandenen 3.3 V betrieben werden können. Allerdings zeichnen sich beide CC430 durch einen erhöhte Leistungsaufnahme im tiefsten Ruhemodus aus. Dies führt zu einer höheren Belastung der Zelle durch den Sensor. Da der Ruhestrom immer noch in der Größenordnung der Selbstentladung der Zelle liegt, kann dies in Kauf genommen werden. Die Peripherie der CC430 leistet das Gleiche wie die des MSP430. Einzige Ausnahme bildet der zweite Timer, der beim MSP430 sieben Capture/Compare Register besitzt und bei allen CC430 Versionen nur fünf. Da in der derzeitigen Software allerdings nur zwei davon verwendet werden, scheint auch dies einen Wechsel nicht zu behindern. Die Taktfrequenz des MSP430 lässt sich um bis zu 25% gegegenüber dem MSP430 steigern, was eventuell zu einer weiteren Erhöhung der Burstmessrate genutzt werden kann. Ein deutlicher Vorteil ergibt sich aus dem verdoppelten RAM-Speicher, der es ermöglicht, die Anzahl der aufgenommenen Burst-Werte zu verdoppeln, ohne auf Komprimierungsmaßnahmen zurückgreifen zu müssen.

Auch der größere CC430F6137 kommt bereits im gleichen Package wie der MSP430F235, sodass durch das Einsparen des CC1101 Bausteins bereits ein Raumgewinn erzielt werden kann. Allerdings zeigt ein Blick auf den Schaltplan des Zellsensors v0.3, dass 19 Pins unbelegt sind.

Da die CC430 die Masseanbindung über das Thermal-Pad an der Unterseite des Bausteins realisieren, werden hier eine ganze Reihe Pins eingespart. Von den 48 Pins des CC430F5137 sind zehn fest für die Funktionalität des Radio-Cores verwendet, sechs dienen der Spannungsversorgung und Stabilisierung und weitere sechs sind für das Programmier- und Debug-Interface notwendig. Damit verbleiben 26 Pins, die durch eine interne PortMap-Matrix mit allen logischen Ports und Sonderfunktionen wie ADC, UART oder Timer verbunden werden können. Der Zellsensor v0.3 verwendet derzeit 23 Pins für die verschiedenen Funktionen. Da die Verbindung zum CC1101 eingespart werden kann, werden davon vier Pins frei. Ein zusätzlicher Pin muss für das Chip-Select Signal des Erweiterungsmoduls vorgesehen werden. Daher ist damit zu rechnen, dass 20 I/O-Pins verwendet werden müssen. Damit verbleibt eine Reserve von 5 Pins für eventuell während des Designs auftretende Änderungen. Durch das kleinere Package und weniger unverbundene Pins wird das Wegführen der Signalleitungen vom Mikrokontroller komplizierter, da mehr Leitungen auf engem Raum zusammen liegen, dennoch ist der Platzgewinn, der durch das kleinere Package erreicht werden kann, für die zusätzliche Verbindung zum Erweiterungsmoduls voraussichtlich notwendig.

Daher wird beschlossen, das Redesign des Zellsensors mit einem CC430F5137 als Mikrocontroller durchzuführen.

	MSP430F235	CC1101	CC430F6137	CC430F5137
Allgemein				
U_S	1.8 - 3.6 V	1.8 - 3.6 V	2.0 - 3.6 V	2.0 - 3.6 V
Active Mode Current				
@16MHz	6.5 mA		3.45 mA	3.45 mA
@ 1MHz	320 μA		320 μA	320 μA
LPM Current				
LPM0	75 μA		90 μA	90 μA
LPM2	23 μA		7.5 μA	7.5 μA
LPM3	1 μA		2.1 μA	2.1 μA
LPM4	0.1 μA		1.2 μA	1.2 μA
Rx/Tx Current				
Sendebetrieb		29 mA	29 mA	29 mA
Empfangsbetrieb		17 mA	17 mA	17 mA
RF sleep mode		0.5 μA	Bereits in LPM Current enthalten	
Temperaturbereich	-40 - 105°C	-40 - 85°C	-40 - 85°C	-40 - 85°C
Microcontroller				
Architektur	16 Bit RISC		16 Bit RISC	16 Bit RISC
Taktfrequenz	16 MHz		20 MHz	20 MHz
SRAM (kByte)	2		4	4
Flash (kByte)	16		32	32
Bauform				
Package	QFN	QFN	QFN	QFN
Pin Anzahl	64	20	64	48
Maße	9 x 9 mm	4 x 4 mm	9 x 9 mm	7 x 7 mm
Peripherie				
GPIO -Pins	48		44	26
USCI-Kanäle	A & B		A & B	A & B
Timer	2x 16 Bit		2x 16 Bit	2x 16 Bit
Multiplizierer	HW 16x16		HW 32x32	HW 32x32
AD-Umsetzer				
Typ	12 Bit SAR		12 Bit SAR	12 Bit SAR
externe Kanäle	8		12	8
Temperatursensor	Ja		Ja	Ja
RF-Eigenschaften				
433 MHz-Band		Ja	Ja	Ja
Eingangs-		-116 dBm @	-117 dBm	-117 dBm @
Empfindlichkeit		0.6 kBaud	0.6 kBaud	0.6 kBaud
Ausgangsleistung		12 dBm	12 dBm	12 dBm
Datenrate		0.6-600 kbps	0.6-600 kbps	0.6-600 kbps
Schnittstelle		SPI	Memory-Mapped	Memory-Mapped

Tabelle 3.7: Vergleich zwischen MSP430F25 und CC1101 gegenüber CC430FX137

4 Realisierung und Redesign

4.1 Zellensensor Klasse 3 Version v0.4

4.1.1 Auswahl der Bausteine

Die Entscheidung für grundlegende Design-Änderungen wurden im vorangegangenen Kapitel getroffen. Statt des Mikrocontrollers MSP430F235 und des Transceivers CC1101 wird ein CC430F5137 eingesetzt, der die Eigenschaften beider Bausteine kombiniert. Außerdem wird, entsprechend der Empfehlung aus [9], der AS3930 in einem kleineren Gehäuse verwendet. Tabelle 4.1 zeigt die für den Zellensensor Klasse 3 v0.4 verwendeten ICs.

Da der RF-Oszilator CTS-405C11A26M00000, der im alten Zellsensor verwendet wurde, nicht zeitnah lieferbar war, wurde auf den Q24FA20H00005 gewechselt, der ebenfalls die geforderten 26 MHz bereitstellt und dieselbe Genauigkeit von 10 ppm aufweist.

Als Pfostenstecker, der die Verbindung zum Erweiterungsmodul herstellen soll, wurde einer aus der gleichen Familie verwendet, der schon für das Programmier-Interface auf der Unterseite des Sensors zum Einsatz kommt.

Abbildung 4.1 auf der nächsten Seite zeigt ein Blockschaltbild des veränderten Zellensensor. Statt des Transceiver ist nun das Erweiterungsmodul über eine SPI Verbindung angebunden.

Bauteil	Zellsensor Klasse 3 v0.3		Zellsensor Klasse 3 v0.4	
	Bezeichnung	Package	Bezeichnung	Package
Mikrocontroller	MSP430F235	QFN-64	CC430F5137	QFN-48
UHF Transceiver	CC1101	QFN-20	-entfällt-	-entfällt-
Spannungswandler	TPS61201	DFN-10	TPS61201	DFN-10
LF Wake-Up Receiver	AS3930	TSSOP-16	AS3930	QFN-16
Antennenumschalter	ADG918	SOP-8	ADG918	SOP-8
Temperatursensor	TMP102	SOT563	TMP102	SOT563

Tabelle 4.1: Übersicht über die Hardwarekomponenten des Zellsensors v0.4 im Vergleich zum Zellensensor v0.3

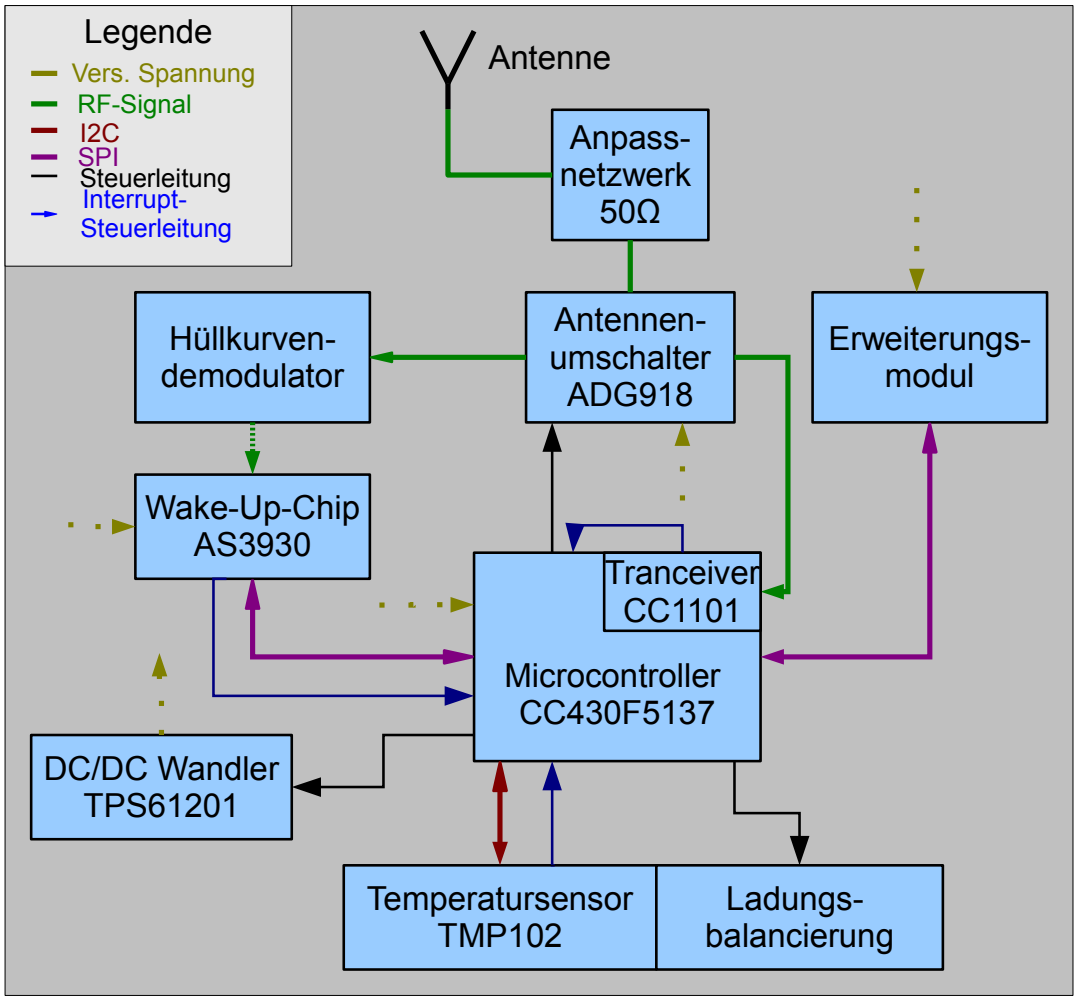


Abbildung 4.1: Blockschaltbild des redesignten Zellensensors.

4.1.2 Schaltplan

Die erstellten Schaltpläne sind in Anhang C.1 abgebildet und außerdem auf dem beigefügten Datenträger zu finden.

Obwohl der CC430F5137 über eine Port-Map-Matrix verfügt, die es ermöglicht, jede beliebige Sonderfunktion und jeden logischen Port-Pin auf einen der 26 frei verfügbaren physikalischen Pins zu legen, wurden alle Sonderfunktionen an den physikalischen Pins belassen, mit denen sie nach einem Reset verbunden sind.

Alle weiteren Funktionen sind so auf die noch freien Pins verteilt worden, wie es dem Layout am dienlichsten war. Dadurch ergeben sich sehr deutliche Unterschiede zur Pinbelegung des alten Zellensensors. Dabei wurde darauf geachtet, dass nur die Ports 1 und 2 Interrupt fähige Eingänge besitzen, weshalb Signale die einen Interrupt auslösen sollen mit einem dieser Ports verbunden wurden.

Mit den in Abschnitt 3.4.3 auf Seite 96 besprochenen Drosseln zum vermindern des Einflusses langer Leitungen auf die Sende-/Empfangsleistung wurde die Versorgungsspannungsleitung +3V3 in fünf Abschnitte unterteilt. Ebenfalls wurde eine Drossel kurz vor den Enable-Eingang des DC/DC-Konverters gesetzt, da diese Leitung ebenfalls den halben Sensor umspannt. Durch diese Drossel sollen Fehlschaltungen des Konverters verhindert werden.

Es soll noch erwähnt werden, dass durch die Änderung des Packages am AS3930 die Signalleitung AS3930_CL_DAT, die in der bestehenden Software keine Verwendung findet, am neuen Package nicht mehr auf einen PIN geführt wird und daher entfällt.

4.1.3 Platinenlayout

Für eine Reihe von Komponenten wurde festgelegt, dass ihre Position auf dem neuen Zellensensor mit der auf dem alten identisch sein soll. Dazu zählen:

- Die **Schleifenantenne** mit einem Radius von 2.5 cm sollte unverändert bleiben, um die Sende- und Empfangseigenschaften möglichst wenig zu verändern.
- Der **Pol-Anschluss** im Zentrum des Senders, der einen Radius von 10 mm hat und eine Zentralbohrung mit dem Durchmesser 8.1 mm für eine M8-Schraube besitzt.
- Die **Anschlusskontakte** für den nicht mit dem zentralen Kontakt verbundenen Pol, um weiterhin frei wählen zu können, an welchen Zell-Pol der Sensor mechanisch angeschlossen wird.
- Die **Programmierschnittstelle** für den Nadel-Adapter, um den Nadel-Adapter nicht für den neuen Zellensensor anpassen zu müssen.

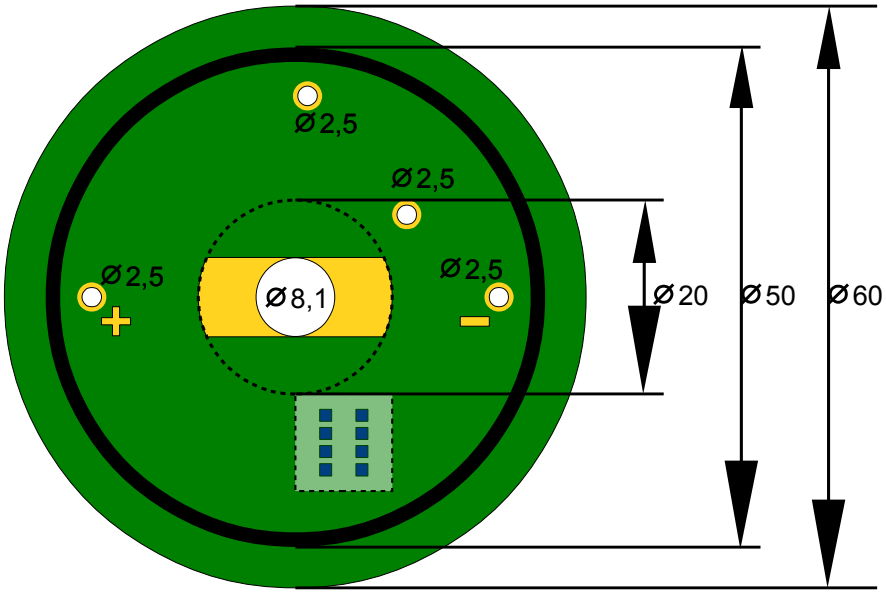


Abbildung 4.2: Maße der Platine für den Zellensensor Klasse 3 Version v0.4. Alle Angegeben Maße in mm. Die in blau dargestellten Pads für den JTAG-Nadeladapter befinden sich auf der Unterseite der Platine.

Zusätzlich werden noch zwei weitere Bohrungen für eine starre Verbindung zum Erweiterungsmodul vorgesehen. Diese sollen zusammen mit dem Pfostenstecker eine stabile Drei-Punkt-Verbindung zwischen Zellensensor und Erweiterungsmodul schaffen. Um den Platzbedarf gering zu halten, werden wie für die Anschlusskontakte 2.5 mm Bohrungen verwendet. Die Bohrungen werden an der dem Plus-Pol gegenüberliegenden Seite angeordnet, da der Stecker möglichst dicht an diesem platziert werden soll, um eine störungsarme Anbindung des ADC auf dem Erweiterungsmodul zu gewährleisten.

Abbildung 4.2 zeigt die genannten Komponenten zusammen mit den wichtigen Maßen.

Das fertige Layout ist im Anhang D in Abbildung D.1 dargestellt. Dabei wurde die Position des Mikrocontrollers im unteren rechten Bereich beibehalten, da hier der meiste Platz für die zahlreichen Leitungen vorhanden ist.

Es ist zu erkennen, dass die Anschlüsse der Antenne um 45° gegenüber des bisherigen Sensors verschoben wurde. Dies war deshalb notwendig, da durch den Wegfall des CC1101 der Weg zwischen Antennenumschalter und Mikrocontroller verkürzt werden musste. In diesem Bereich befindet sich nun die entzerrte RF-Schaltung aus Tiefpass und Balun, die in ihrem Aufbau der Schaltung aus dem Referenzdesign entspricht. Unterhalb dieser Signalpfades liegt die einzige Massefläche des Sensors, die an ihren Enden auf kurzem Weg mit der Antennenmasse und der Masse des Mikrocontroller/Transceiver verbunden ist.

Die Pfostenleiste für den Anschluss des Erweiterungsmoduls befindet sich direkt neben dem positiven Anschlusspol, um eine Messung der Batteriespannung auf kurzem Wege zu ermöglichen.

4.2 Erweiterungsmodul v0.1 mit 24 Bit Δ-Σ-ADC

Beim Design des Erweiterungsmoduls mit dem ADS1291 wurde, wie schon beim Zellensensor, auf einen kurzen Weg der zu messenden Spannung zum Sensor gesetzt. Ansonsten wurde im wesentlichen das Referenzdesign des ADS1291 als Vorlage verwendet.

Anders als auf der Platine des Zellensensors ist auf dem Erweiterungsmodul viel freier Raum vorhanden, sodass alle relevanten Signale mit einem Testpunkt versehen werden konnten. Dabei ist vor allem der Testpunkt am Data-Ready Signal relevant. Dieses Signal zeigt an, wann der ADC fertige Daten besitzt. Das Signal wird nicht zum Mikrocontroller geführt, um den Stecker möglichst klein zu halten. Die Konversionszeit einer Messung ist deterministisch bestimmbar. Da die Konversion vom Mikrocontroller angestoßen werden soll, um die Synchronität mit dem Burst-Signal zu gewährleisten, kann daher auch ein Timer eingesetzt werden, um den Zeitpunkt der Datenabfrage zu bestimmen. Für den Test während der Inbetriebnahme kann mit Hilfe dieses Testpunktes die exakte Berechnung der Konversionsdauer geprüft werden.

4.2.1 Schaltplan

Abbildung 4.3 auf der nächsten Seite zeigt die Beschaltung des ADS1291. Die nicht genutzte Eingänge wurden beidseitig auf das Versorgungsspannungsniveau gelegt. Da der interne Taktgeber verwendet werden soll, wurde ebenfalls das Clk-Select Signal auf das HI Potential gelegt und der CLK-Pin offen gelassen. An diesem Pin könnte der interne Takt des ADS1291 abgerufen werden, es wurde darauf verzichtet diesen mit einem Testpunkt zu verbinden, um die Oszillatorschaltung nicht zu verstimmen.

Abbildung 4.4 auf Seite 109 zeigt neben den Signalen, die über einen Testpunkt verfügen, noch die elektrische und mechanische Anbindung des Zellensensors. Die elektrische Anbindung geschieht über eine Buchse für Pfosten-Stecker im 1.27 mm Raster. Sie stellt das Gegenstück zur Pfostenleiste auf dem Zellsensor dar und ist entsprechend verdrahtet. Als mechanische Verbindung wurden vier Bohrungen vorgesehen. Zwei davon liegen beim späteren Einbau exakt über den extra für diesen Zweck vorgesehenen Bohrungen auf dem Zellsensor. Die anderen beiden liegen über den Bohrungen in den Anschlusskontakten des Zellensensors. Sollte der Zellensensor mit dem zentralen Anschluss-Pol auf einer Zelle verschraubt sein, kann der nicht genutzte Anschlusskontakt ebenfalls für die mechanische Verbindung genutzt werden.

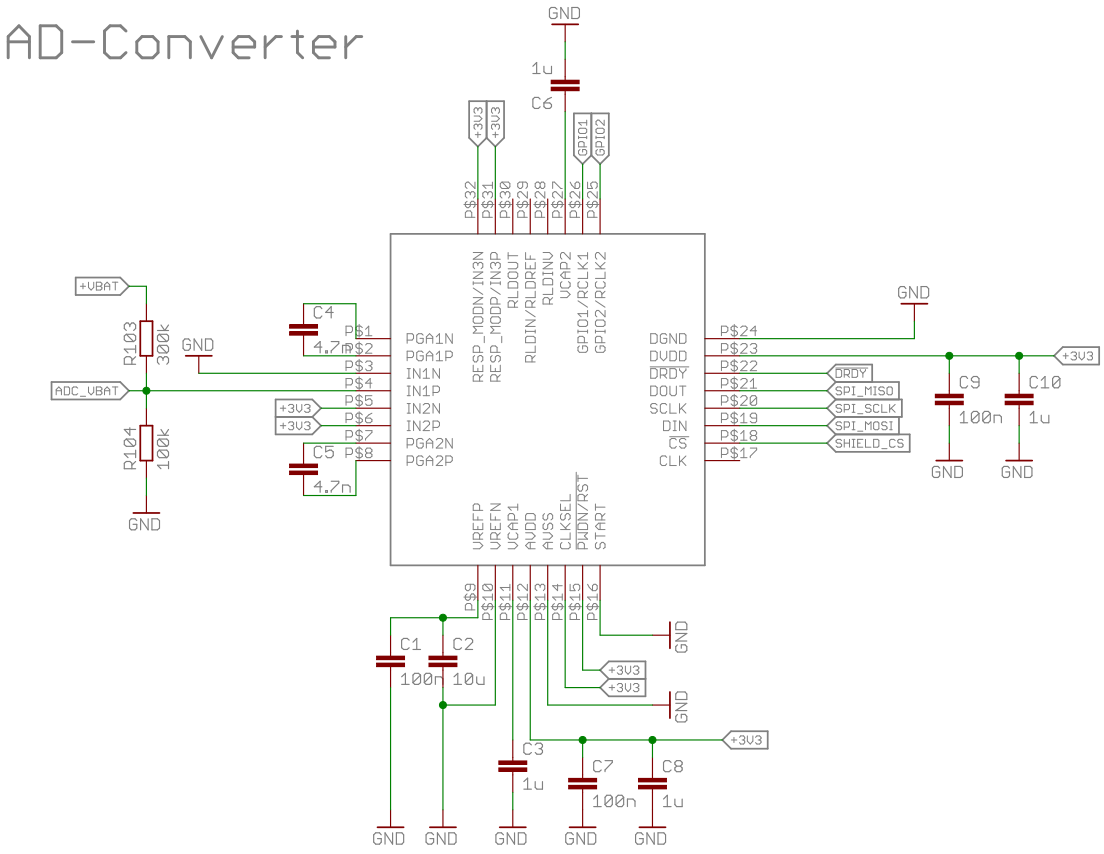
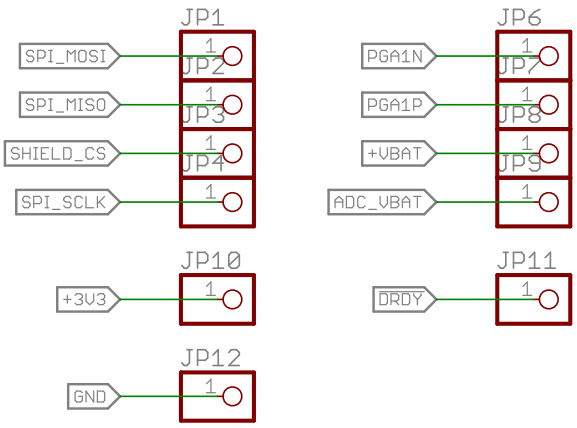
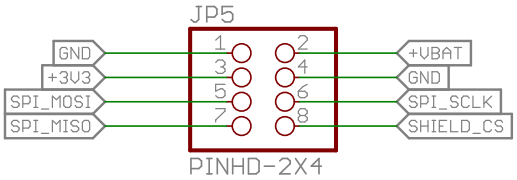


Abbildung 4.3: Beschaltung des ADS1291.



(a) Testpunkte für die SPI-Schnittstelle, die Spannungsversorgung, die Messspannung und das Ausgangssignal der internen Verstärkerspannung



(b) Pfostenstecker

(c) Mechanische Verbindung

Abbildung 4.4: Peripherie auf des Erweiterungsmoduls.

4.2.2 Platinenlayout

Die Geometrie, der in Abbildung D.3 auf Seite 175 gezeigten Platine, wird maßgeblich von den vier aus dem Zellsensorlayout übernommenen mechanischen Anschlusspunkten bestimmt. Die ungewöhnliche fünfeckige Form wurde gewählt, um zehn Platinen des Erweiterungsmoduls auf das Format einer Eurokarte zu bekommen. Dadurch können bei der Fertigung Kosten eingespart werden.

Die Platzierung des ADS1291 auf dem Erweiterungsmodul ist so gewählt, dass erneut ein kurzer Weg der zu messenden Zellenspannung realisiert werden kann. Die übrige Leitungsführung erfolgt auf kurzem Weg und ist aufgrund des ausreichend vorhandenen Platzes unkritisch. Die gesamte Schaltung wurde mit einer Massfläche unterlegt, um den Einfluss von Störungen auf den ADC zu vermindern. Dieser ist deshalb anfälliger als der interne ADC des Mikrocontrollers, weil er deutlich kleiner Spannungsänderungen erfasst. Da sich die Massefläche deutlich innerhalb der Schleifenantennen-Geometrie auf dem Zellensensor befindet und außerdem noch oberhalb befindet, ist davon auszugehen, das die Beeinflussung der Schleifenantenne vertretbar ist.

4.3 Softwareentwicklung

Da die verwendete Peripherie in weiten Teilen mit den bisherigen Sensoren übereinstimmt, können große Teile der Software wiederverwendet werden. Dabei ist es von Vorteil, dass die bestehende Software eine Strukturierung aufweist, die jedem Baustein der Peripherie, sowie verschiedenen logischen Einheiten des Mikrocontrollers, einen eigene Header-Datei und eine eigene C-Datei zuordnet. Innerhalb dieser Dateien sind die Funktionen definiert, mit denen der Mikrocontroller auf den Baustein Zugriff hat.

4.3.1 CC430

Die Tabellen E.1 und E.2 im Anhang E zeigen einen sogenannten Change-Log der Software. Darin sind alle Quellcode-Dateien des Zellsensor-Projektes und ihre jeweiligen Autoren aufgeführt. Bei Dateien, die nicht im Rahmen dieser Arbeit entstanden sind, wird erläutert, welche Änderungen für den neuen Zellsensor vorgenommen wurden. Innerhalb der Quellcode-Dateien sind alle Änderungen mit Datum und dem Kürzel des Autors dokumentiert.

Weil sich an der Peripherie nur wenige Änderungen ergeben haben, können diese Dateien weiter verwendet werden. Da die Pinbelegung beim CC430F5137 anders ist als beim MSP430F235, muss diese in der Software angepasst werden. Die bestehende Software ist in dieser Hinsicht generisch geschrieben, daher musste in den meisten Fällen nur die Header-Datei angepasst werden.

Neu angelegt wurde die Software zur Steuerung des Radio-Core innerhalb des CC430. Die Software, welche für den Transceiver CC430 erstellt wurde, basiert teilweise auf der Software des CC1101 von Sassano [9] und Durdaut [8]. Insbesondere wurde die funktionale Aufteilung identisch vorgenommen, d. h., dass z. B. die Funktionen inklusive ihrer Parameter der alten Software gleichen. Dadurch wurde sichergestellt, dass ein einfaches Einpflegen in den übrigen bestehenden Quellcode möglich ist. Innerhalb der Funktionen kommt es zu Abweichungen, die auf zwei Ursachen zurückzuführen sind. Zum einen ist es nicht mehr nötig, den Transceiver über das Serial Peripheral Interface (SPI) anzusprechen, da alle Register der RF-Hardware über die Memory-Map des Controllers ansprechbar sind. Dadurch sinkt der Quellcodeumfang und der Zeitbedarf der zur Laufzeit beim Konfigurieren und Auslesen nötig ist. Des Weiteren war es in der bisherigen Software der Fall, dass je nach Übertragungsart, d.h. Paketbetrieb oder Burst, die drei Daten-Ausgänge des CC1101 (GDO0-GDO2) jeweils unterschiedlich konfiguriert wurden, um auf verschiedene Ereignisse zu reagieren. Diese Ereignisse haben in der Hardware des CC430 eigene Interrupts, sodass es nicht mehr notwendig ist, hier neu zu konfigurieren.

Dadurch verbessert sich die Fehlersicherheit, da ein Signal nur noch eine einzige Bedeutung haben kann und nicht mehr unterschiedliche Ereignisse beschreibt. Außerdem steigt die Übersichtlichkeit des Quellcodes. Für die Realisierung der Kommunikation wurden Teile des Beispielcodes verwendet, den Texas Instruments für die CC430 Familie bereitstellt.

Für die Softwareentwicklung wurde die Entwicklungsumgebung Code-Composer-Studio von Texas Instruments verwendet. Da die Deklaration von ISR von der zuvor verwendeten Entwicklungsumgebung anders gehandhabt wird, mussten alle Interrupt-Service Routinen umgeschrieben werden.

4.3.2 ADS1291

Für den 24 Bit Δ-Σ-ADC ADS1291 musste umfangreiche Software erstellt werden. Diese ist im Anhang E in den Abschnitten E.3 und E.4 dargestellt. Die maximal mögliche Messfrequenz der Burst-Messung liegt derzeit bei 10 kSPS, oder entsprechend 100 μs Zeit, um einen Messwert zu erzeugen.

Der ADS1291 kann zwar mit bis zu 8 kSPS Werte erzeugen, allerdings werden diese im sogenannten ‚Continuous-Mode', bei dem der ADC die Konversion kontinuierlich durchführt und das Vorhandensein eines Wertes über sein DataReady anzeigt, erzeugt.

Mit dieser Methode ist es nicht möglich, den genauen Startzeitpunkt der Messung festzulegen. Um den Startzeitpunkt festlegen zu können, muss der Sensor aus dem kontinuierlichen Messmodus herausgeholt und jede Messung explizit mit dem Kommando 'Start' getriggert werden. Nach einem solchen Startkommando benötigt der ADC eine gewisse Anzahl Taktzyklen, bis die Filter eingeschwungen sind und die tatsächliche Konversion beginnen kann.

Timereinstellungen

In [9] wird für das Aufnehmen eines Spannungswertes mit dem internen 12 Bit AD-Wandler eine feste Aufnahmezeit von $92\mu s$ veranschlagt. Durch das Einschwingen der Filter beim ADS1291 sind die Aufnahmezeiten zum einen deutlich länger als die 92 us des internen 12 Bit ADC, zum anderen ergeben sich je nach eingestellter Samplerate unterschiedliche Aufnahmedauern.

Damit ist die in Abbildung 3.7 auf Seite 78 gezeigte Ausführungszeit der ISR nicht mehr konstant, sondern abhängig von der eingestellten Burstrate. Daher muss für jede Burstrate die Timereinstellung für das geschlossene Zeitfenster t_{OFF} separat bestimmt und eingestellt werden. Daraus ergibt sich das in Abbildung 4.5 auf der nächsten Seite dargestellte Bild.

Tabelle 4.2 auf Seite 115 zeigt die Anzahl der für das Einschwingen der Filter notwendigen Taktzyklen. Ein solcher Zyklus benötigt nach Datenblatt maximal 868 μs [54]. Daraus lässt sich die Zeit berechnen, die der ADC benötigt, um einen Wert zu erzeugen. Zu dieser Zeit addieren sich noch 74 μs für die ISR bevor das 'Start'-Kommando gesendet wird.

Diese Zeit muss für die jeweilige Burst-Frequenz von dem Wert abgezogen werden, der sich für das geschlossene Zeitfenster ergibt. so wird die in Abbildung 4.5 auf der nächsten Seite dargestellte Zeit t_{OFF} erhalten. Diese muss mittels eines Timers abgewartet werden, bis sich das Fenster für das nächste Burst-Signal wieder öffnen soll.

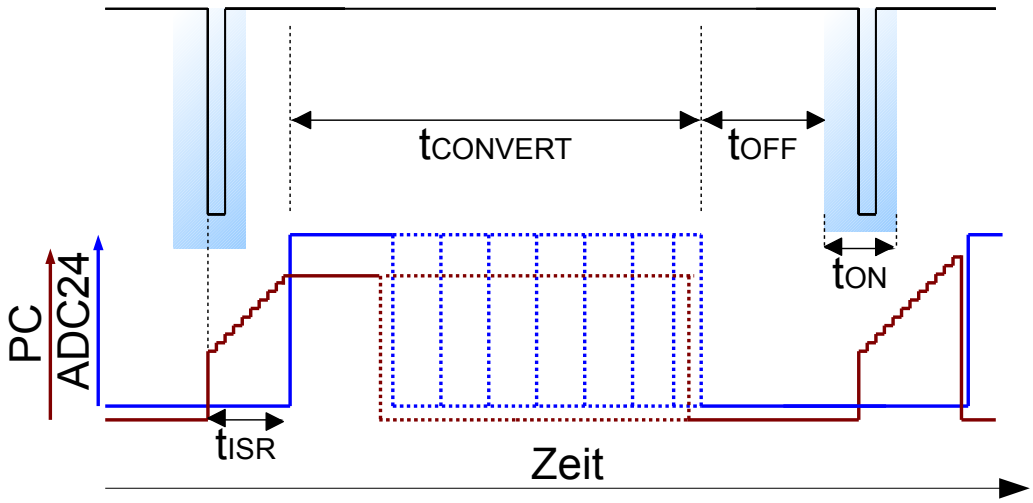


Abbildung 4.5: Berechnung der Timereinstellung für das Zeitfenster. Die Verzögerung durch die ISR ist in rot in Form des fortlaufenden Programm Counters (PC) dargestellt. Die Laufzeit der ISR ist abhängig von der eingestellten Samplerate, daher muss sie für jede Mögliche Einstellung im Vorfeld berechnet werden.

Für die maximal einstellbare Frequenz kommt zu dieser Zeit auch noch die halbe Breite des Zeitfensters hinzu, da das Fenster erst geöffnet werden darf, wenn die letzte Kalkulation abgeschlossen ist. Die halbe Fensterzeit beträgt momentan 200 μs. In der genannten Tabelle wird daher auch diese Zeit und die daraus resultierende maximale Messrate gezeigt. Daraus resultiert, dass die synchronisierte Messung mit dem ADS1291 mit maximal 1150 Hz erfolgen kann.

Für höhere Frequenzen wurde daher die kontinuierliche Messmethode implementiert, die zwar die Werte mit der geforderten Geschwindigkeit liefert, aber bei der die Synchronität verloren geht.

Der Tabelle können wir außerdem entnehmen, dass die Abtastrate begrenzt werden muss, um die in der Konzeptionierung gestellten Anforderungen an die Spannungsauflösung zu erfüllen. Um das 10 μV Kriterium einzuhalten, war eine Auflösung von 17.9 Bit gefordert. Soll diese erreicht werden, so muss als Einstellung für den ADC 1 kSPS gewählt werden, was im synchronisierten Betrieb maximal eine Burstmessung mit 210 Hz ermöglicht. Eine synchronisierte Messung bei eingestellter Sample-rate von 2 kSPS ist mit bis zu 395 Hz möglich, wobei hier die gestellte Anforderung an die Auflösung nur leicht verletzt wird. Das schwächere Auflösungskriterium 100 μV kann für Burst-Messungen mit bis zu 704 Hz erfüllt werden.

DR [2:0]	Sample-rate	Settling Cycles	Settling Time	einschließlich ISR-Time	Datenrate	ENOB
000	125 SPS	4100	35,59 ms	35,67 ms	27 Hz	20.1 Bit
001	250 SPS	2052	17,82 ms	17,89 ms	55 Hz	19.6 Bit
010	500 SPS	1028	8,93 ms	9,01 ms	108 Hz	19.1 Bit
011	1 kSPS	516	4,48 ms	4,56 ms	210 Hz	18.5 Bit
100	2 kSPS	260	2,26 ms	2,34 ms	395 Hz	17.4 Bit
101	4 kSPS	132	1146 us	1220 us	704 Hz	14.9 Bit
110	8 kSPS	68	591 us	665 us	1157 Hz	12.5 Bit
111		-	-	-	-	-

Tabelle 4.2: Maximale Konversionsrate und zugehörige rauschfreie Bits des ADS1291

Minimierung des Speicherbedarfs

In den bisherigen Sensoren wurde zum Speichern der 12 Bit Abtastwerte des internen ADC ein 2 Byte breiter Speicherbereich reserviert. Da der ADC auf dem Erweiterungsmodul 24 Bit liefert, kann nicht ohne weiteres die bisherige Speicheraufteilung verwendet werden. Da die Größe der Messwerte im Speicher bestimmt, wie viele Messwerte aufgenommen werden können, ist es Sinnvoll zu überprüfen, ob der benötigte Speicherbedarf begrenzt werden kann.

Aus Gleichung (3.3) auf Seite 83 wissen wir, das ca 18 Bit Auflösung für das Einhalten des 10 muV Kriteriums notwendig sind. Tabelle 4.2 zeigt uns, dass nur bei langsamen Messungen mehr als 20 Bit reele Auflösung erreicht werden. Daher wird beschlossen, dass es ausreichend ist, die aufgenommenen Messwerte mit einer Auflösung von 18 Bit zu speichern.

Da die Zellensensoren genutzt werden sollen, um Lithium-Eisenphosphat-Akkumulatorzellen zu vermessen, sind Spannungen unter 2.8 V am Eingang des ADC nicht zu erwarten, da dies die Entladeschluss-Spannung dieses Akkumulatortyps ist. Wir korrigieren deshalb den aufgenommenen Messwert um diese Spannung. Der ADS1291 hat eine interne Referenzspannung von 2.21 V. Durch den 2:1 Spannungsteiler vor der Messung wird aus den zu korrigierenden 2.8 V die halbe Spannung von 1.4 Volt und für den zu kompensierenden Offset gilt:

$$N_{OFFSET} = \frac{^1/_2 U_{OFFSET}}{U_{REF}} \cdot 2^{18} = 166064 = 0x288B0 \tag{4.1}$$

Dieser Wert wird vom Messwert abgezogen. Da 18 Bit noch 2 Bit zu viel sind, um sie direkt in den vorhandenen 2 Byte Speicher pro Messwert unterzubringen, soll betrachtet werden, was passiert, wenn diese Bits ignoriert werden. Dann stehen 2^{16} Werte zur Verfügung um den Spannungsbereich oberhalb der 2.8 V abzudecken. Für den damit erfassbaren Spannungshub ergibt sich:

$$\begin{aligned} 2^{16} &= \frac{{}^{1}/_{2}U_{Hub}}{U_{REF}} \cdot 2^{18} \\ U_{Hub} &= \frac{2^{16}}{2^{18}} \cdot 2.42\,\mathrm{V} \cdot \\ &= 1.105\,\mathrm{V} \end{aligned}$$

Werden also die ersten 2 Bit des auf 18 Bit reduzierten Messwertes ignoriert, kann der Spannungsbereich von 2.8 V bis 3.9 V gespeichert werden. Dies deckt den gesamten für die EIS interessanten Spannungsbereich ab, da eine Durchführung der Impedanzspektroskopie im Lade- oder Entladeschluss wenig Sinn ergibt. Für die Auswertung der aufgenommenen Spannungen muss der Offset wieder eingerechnet werden. Sollten dennoch Spannungen außerhalb des angegeben Bereiches auftreten, kommt es zu einem Über- oder Unterlauf.

5 Erprobung des Testsystems

In diesem Kapitel wird der neue Zellensensor vorgestellt und die synchronisierte Messung von Strom und Spannung mit dem System aus dem neuen Zellensensor und der Basistation aus [35] getestet. Anschließend wird eine beispielhafte Impedanzspektroskopie an einer Testimpedanz mit bekannten Eigenschaften vorgenommen. Abschließend erfolgt die Vermessung der Impedanz einer realen $LiFePO_4$-Zelle mit den Möglichkeiten des neuen Zellensensors.

5.1 Zellensensor Klasse 3 Version v0.4

Abbildung 5.1 auf der nächsten Seite zeigt den bestückten Zellensensor v0.4. Links in der Nähe des positiven Anschlusskontaktes befindet sich die Pfostenleiste zum Aufstecken des Erweiterungsmoduls. Für die mechanische Absicherung des Erweiterungsmoduls befinden sich Bohrungen am oberen Ende des Sensors und unterhalb des Antennenumschalters. Außerdem relevant ist der neue Mikrocontroller CC430F5137 zusammen mit dem RF-Signalpfad, der die benötigten Elemente wie gefordert in einer Linie mit kurzen Wegen anordnet. Des Weiteren ist zu erkennen, das generell auf mehr Abstand zwischen Schleifenantenne und Bauelementen geachtet wurde.

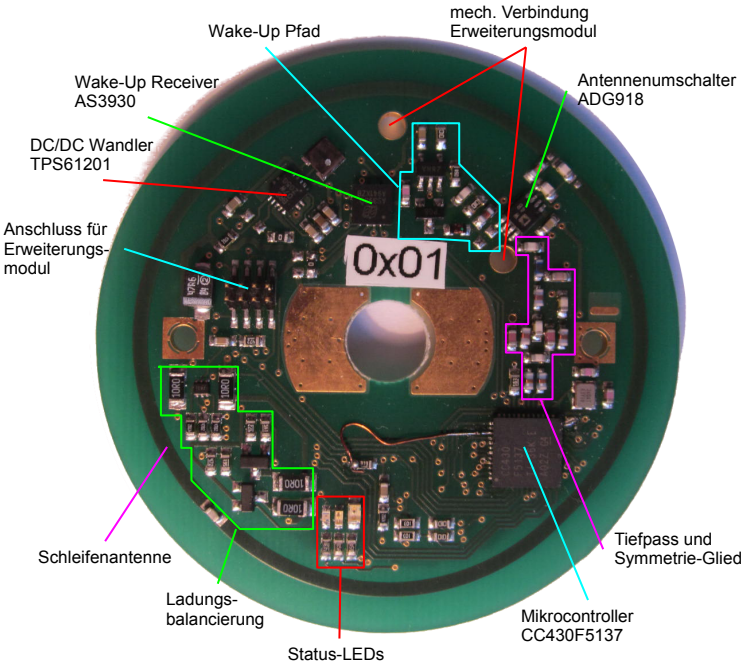


Abbildung 5.1: Bestückte Zellensensorplatine / Vorderseite

5.1.1 Korrektur von Designfehlern

Während der Inbetriebnahme wurden zwei Fehler im Design festgestellt. Beide konnte durch den Einsatz von Fädeldraht behoben werden. Für zukünftig zu fertigende Zellensensoren v0.4, wurde das in Abschnitt C.2 auf Seite 168 gezteigte Design und das in Abbildung D.2 auf Seite 174 gezeigte Layout vorbereitet, das diese Probleme nicht mehr aufweist. Im Folgenden werden die beiden Designfehler kurz vorgestellt.

JTAG_TEST Der in ZS k3 v0.3 verwendete Controller MSP430F235 nutzte zum Programmieren und Debuggen eine 4-Wire JTAG Verbindung. Die zusätzlich für Spy-by-Wire und andere Nutzungsmöglichkeiten zur Verfügung gestellten Signale RST/NMI und JTAG_TEST wurden zum Programmieradapter geführt, ohne in Nutzung zu sein. Beim Design des ZS k3 v0.4 wurde dabei übersehen, dass der CC430F5137 eine Shared-JTAG Schnittstelle besitzt, die vom Programmiergerät über den dedizierten JTAG_TEST/SBWCLK Pin konfiguriert wird. Hier wurde beim Design das JTAG_TEST Signal mit einem beliebigen GPIO-Pin verbunden, wie es auch im Vorgänger der Fall war. Dadurch kann der Programmieradapter MSP-FET430UIF nicht ohne Weiteres auf den Controller zugreifen. Um mit den gefertigten Platinen dennoch arbeiten zu können, wurde ein Fädeldraht vom dedizierten JTAG_TEST Pin zur Schnittstelle gezogen.

GDO0 Bei den Zellensensoren, welche den dedizierten CC1101 als Transceiver benutzen, werden alle Interrupts dem Mikrocontroller über zwei Interrupt-Leitungen signalisiert. Dabei ist die Bedeutung des jeweiligen Interrupt-Signals abhängig von der Konfiguration des CC1101 und wird für die verschiedenen Betriebsmodi umgestellt. Der Radio-Core des CC430 hat die Möglichkeit, beim Erkennen bestimmter Ereignisse, direkt auf den Interrupt-Controller des CC430 zuzugreifen und diese zu signalisieren. Dies wird so in der aktuellen Software z. B. genutzt, um den vollständigen Empfang oder Versand einer Nachricht zu signalisieren. Während der Konzeptionierung wurde davon ausgegangen, dass die Interrupts, die im asynchronen Modus die fallende Flanke des Burst-Signal anzeigen, ebenfalls an den Interrupt-Controller des CC430 geleitet werden können. Es zeigt sich, dass dies zwar möglich ist, dafür aber ein Umweg über die Timer Interrupts genommen werden muss. Das bedeutet, das 'Generell Data OUT' (GDO) des Radio-Core signalisiert dem Timer den Start der Zählung, der bei einer eingestellten Zähldauer von Null sofort den Timer-Interrupt auslöst. Innerhalb der Interrupt-Service-Routine muss dann in Software unterschieden werden, ob es sich um einen Radio-Core Interrupt oder einen Timer Interrupt gehandelt hat.

Da für dieses Vorgehen ein Timer umprogrammiert werden muss, kann er in dieser Zeit nicht anderweitig verwendet werden. Daher wurde entschieden stattdessen das GDO0-Signal des Radio-Core auf einem Pin auszugeben und mit einem anderen interruptfähigen Pin wieder einzulesen. Zwar war der Pin 1.0, dem das GDO0 Signal standardmäßig zugeordnet ist, zufällig nicht mit einer anderen Funktion versehen, allerdings waren beide benachbarten PINs belegt, sodass es einen gewissen Aufwand erfordert hätte, eine Verbindung zu einem interruptfähigen Eingang herzustellen. Stattdessen wurde die Port-Map-Matrix des CC430 genutzt, um das GDO0 Signal auf den Pin 2.7 zu legen, der bis dahin für den einzigen Testpoint der Sensorplatine verwendet wurde. Der direkt danebenliegende Pin 2.6 wurde dann als Interrupt-Eingang genutzt, der nun signalisiert, das im asynchronen Modus während der Burstmessung eine fallende Flanke detektiert wurde. Der Quellcode zum Einstellen der PortMap ist im Listing 5.1 gezeigt.

```
1  PMAPPWD = 0x02D52;                          // Get write-access to port mapping regs
   CC430_GDO2_OUT_PMAP = PM_RFGDO2;            // Connect GDO0 with P2.7
3  PMAPPWD = 0;                                // Lock port mapping registers
   CC430_GDO2_OUT_PxDIR |= CC430_GDO2_OUT_PIN; // P2.7 output
5  CC430_GDO2_OUT_PxSEL |= CC430_GDO2_OUT_PIN; // P2.7 Select special function
   CC43_BURST_SIG_DIR_IN;                      // P2.6 as inut
7  CC430_BURST_SIG_IRQ_FALLING_EDGE;           // P2.6 Hi/Lo edge
   CC430_BURST_SIG_IRQ_DISABLE;                // Disable until window opening
9  CC430_BURST_SIG_CLEAR_IRQ;
```

Listing 5.1: Einstellen der Port-Map-Matrix und Konfiguration der Interrupts für die asynchrone Burst-Detektion

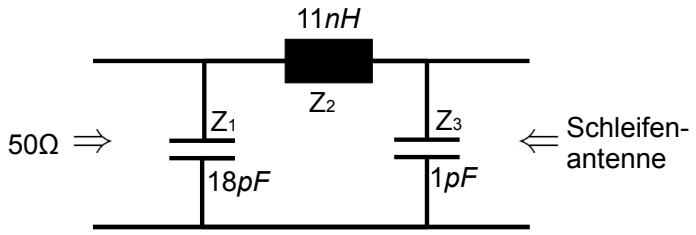


Abbildung 5.2: Anpassnetzwerk zwischen Schleifenantenne und Antennenumschalter.

5.1.2 Impedanzanpassung der Schleifenantenne

Um das Netzwerk für die Anpassung der Schleifenantenne auf 50 Ω zu bestimmen, wurde ein vollständig bestückter Zellensensor verwendet, um den Einfluss aller Elemente berücksichtigen zu können. Für die Messung der Parameter konnte auf die Unterstützung von Herrn B. Eng. Nico Sassano zurückgegriffen werden, der eine derartige Messung bereits für seinen eigenen Zellensensor durchgeführt hatte und daher Erfahrung und den Zugang zu entsprechendem Messequipment besaß. Daher soll an dieser Stelle nur auf die von ihm ermittelten Parameter zur Antennenanpassung verwiesen werden, die in Abbildung 5.2 dargestellt sind.

5.1.3 Allgemeiner Funktionstest

Der Funktionstest für den neuen Zellensensor orientiert sich an dem Funktionstest der letzten Generation aus [9]. Allerdings war es nicht möglich, den Wake-Up des Chips zu testen, da die neue Basisstation die dafür notwendige Funktion noch nicht besitzt. Der Funktionstest umfasst deshalb die folgenden Punkte:

- Sicherstellen der packetorientierten Funk-Kommunikation
- Konfiguration des Zellensensors durch das Batteriesteuergerät
- Abruf einer einzelnen Spannung
- Abruf eines Spannungs- und Temperaturwerts (TMP102)
- Abruf der Temperatur (TMP102)
- Abruf der Temperatur (MSP430)
- Burstmessung
- Burstdaten abrufen

5.2 Erweiterungsmodul v0.1

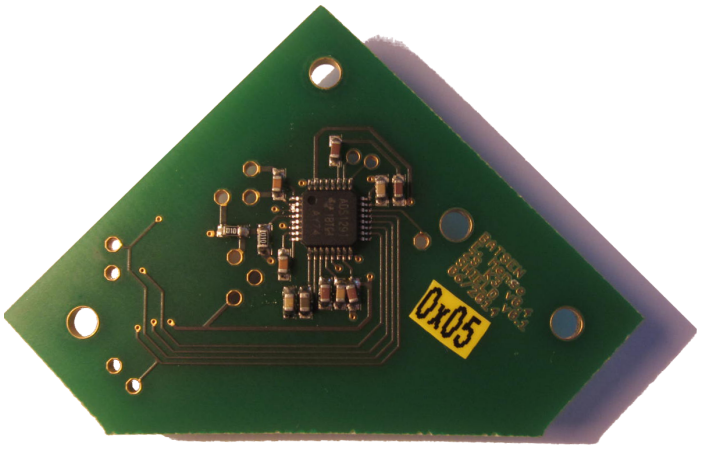
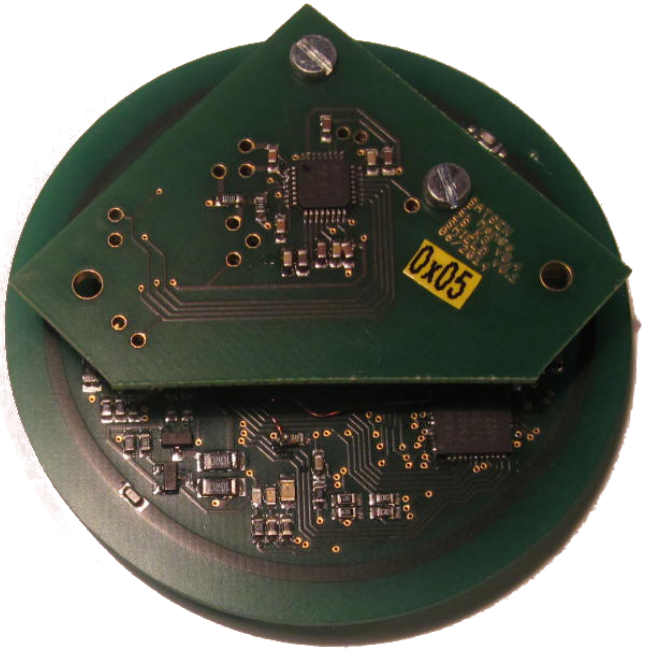


Abbildung 5.3: Oberseite des bestückten Erweiterungsmoduls.

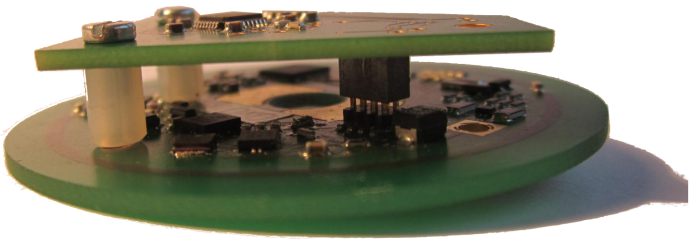
Abbildung 5.3 zeigt die Oberseite des fertigen Erweiterungsmoduls. Die Fertigung kann aufgrund der gewählten Gehäuseform auch manuell erfolgen. Abbildung 5.4 auf der nächsten Seite zeigt die Kombination aus Zellensensor und Erweiterungsmodul.

Der Funktionstest des Erweiterungsmoduls umfasst die folgenden Punkte:

- Prüfen der Kommunikation über SPI durch schreiben und rücklesen der Konfiguration.
- Spannungsmessung bei allen Sampleraten



(a) Draufsicht



(b) Seitenansicht

Abbildung 5.4: Zellensensor mit aufgestecktem Erweiterungsmodul.

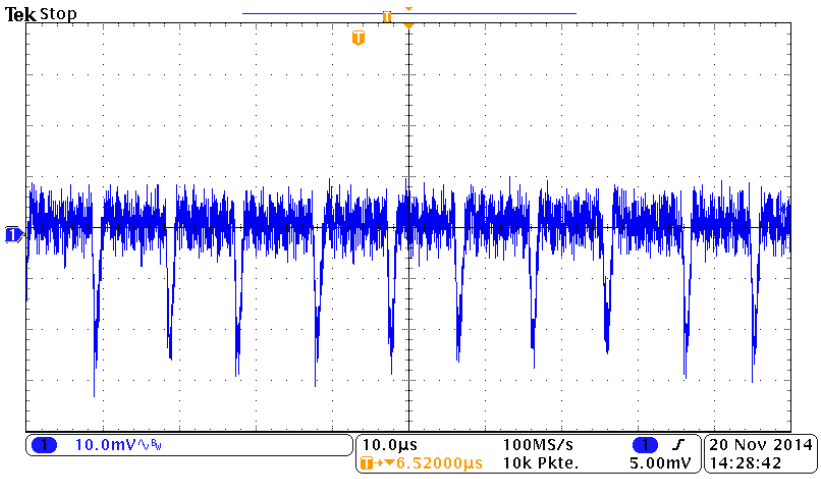


Abbildung 5.5: Periodische Störung der Zellenspannung durch den DC/DC-Schaltregler TPS61201. Um die Störung besser erkennen zu können, ist nur der Wechselanteil dargestellt. Die Zellenspannung beträgt 3.3 V.

5.2.1 Einfluss des DC/DC-Wandlers

Für die Versorgung des Zellensensors kommt mit dem TPS61201, wie in den bisherigen Versionen auch, ein Schaltregler zum Einsatz. Dieser koppelt periodische Störungen in seine Energiequelle ein. In unserem Fall ist das die zu messende Zellenspannung. Abbildung 5.5 zeigt, dass diese Störungen etwa mit einer Periodendauer von etwa 10 μs auftreten. In [9] wurde dieses Problem gelöst, indem der Spannungswandler vor der Messung kurzzeitig ausgeschaltet wird, sodass sich die Spannung beruhigt. Dadurch fällt die Versorgungsspannung langsam ab und stabilisiert sich erst wieder, wenn der Spannungsregler ausreichend lange eingeschaltet ist. Dadurch ist die Messfrequenz begrenzt, bis zu der dieses Verfahren möglich ist.

Im Fall einer Spannungsmessung mit dem Erweiterungsmodul ist dieses Verfahren nicht anwendbar, da der Δ-Σ-ADC seine Auflösung durch Überabtastung über einen großen Zeitraum gewinnt. Würde der Spannungswandler über den gesamten Zeitraum abgeschaltet bleiben, so würde die Versorgungsspannung derart weit abfallen, dass der Zellensensor sich abschalten würde. Daher ist es notwendig, den Spannungswandler während der Messung eingeschaltet zu lassen. Allerdings ergeben sich daraus nicht die zu erwartenden Probleme. Wie oben genannt tritt die Störung periodisch auf. Die Periodendauer ist abhängig von der durch den DC/DC-Wandler versorgten Last und bewegt sich, bei durchgeführten Messungen, zwischen 5 μs und 50 μs. Der Einfluss wird dementsprechend im Spektrum durch Störungen bei etwa 20 kHz bis 200 kHz repräsentiert. Da die Abtastrate des Quantisierers bei 500 kHz liegt, wird das Nyquist-Kriterium nicht verletzt. Durch die Tiefpassfilterung bei der Dezimierung wird der Einfluss der Störung unterdrückt. Daraus folgt, dass die Störung

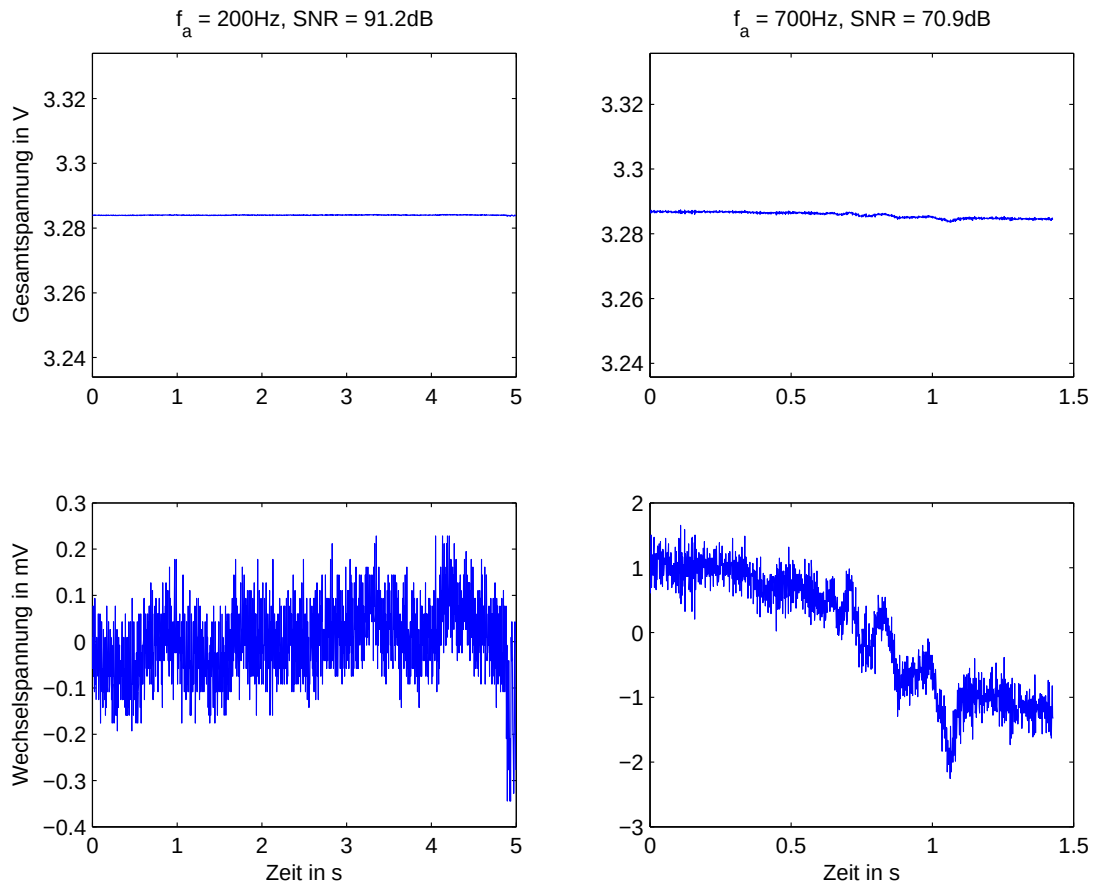


Abbildung 5.6: Einfluss des DC/DC-Wandlers auf das Messergebnis des 24 Bit-ADCs. Links das gemessene Ergebnis bei einer Abtastrate von 200Hz, rechts bei einer Abtastrate von 700 Hz. Um das Ergebnis besser mit der Oszilloskopmessung vergleichen zu können ist in der oberen Reihe dieselbe y-Achsenskalierung gewählt. Der angegebene SNR bezieht sich auf die Ruhespannung. Bezogen auf die für die EIS zu messenden Spannungsamplituden ergibt sich ein deutlich kleinerer SNR.

bei großen Dezimierungsraten weniger Einfluss hat als bei niedrigen Dezimierungsraten. Abbildung 5.6 zeigt das Ergebnis einer Burst-Messung einer stabilen Zellenspannung bei zwei unterschiedlichen Burst-Frequenzen. Es zeigt sich, dass eine ausreichende Unterdrückung der DC/DC-Wandler Störung erreicht wird, auch ohne das dieser abgeschaltet werden müsste.

5.2.2 Synchronisierte Messung phasenverschobener Sinussignale

Um die Synchronisierung der Messung mehrerer Zellensensoren mittels Burst-Signal zu überprüfen, wurden mit einem Funtkionsgenerator zwei phasenverschobene Sinus-Signale erzeugt. Um die Verhältnisse während einer Messung für die Elektrochemische Impedanz-Spektroskopie nachzubilden, wurden dabei Signale mit einem Offset von etwa 3.3 V und einer Wechselspannungsamplitude von etwa 10 mV verwendet. Da der Funktionsgenerator nicht in der Lage war, die von den Zellensensoren benötigte Leistung von etwa 15 mA bei 3.3 V zu liefern, mussten beide Signale durch einen Operationsverstärker gestützt werden. Dabei kamen zwei 'Bipolar Power Amplifier' von KEPCO zum Einsatz. Diese Art des Leistungs-Operationsverstärkers kann positive und negative Ausgangsspannungen im Bereich von $\pm$60 V liefern und sowohl als Stromquelle, wie auch als Stromsenke im Bereich mehrerer Ampere arbeiten. Diese Leistungsdaten sind für die gestellte Aufgabe überdimensioniert. Allerdings werden sie für spätere Messungen benötigt und standen daher zur Verfügung. Die Operationsverstärker werden als Spannungsfolger betrieben, sodass die Zellensenoren aus der auf diese Weise simulierten Zellenspannung versorgt werden können.

Es war nicht möglich, die Operationsverstärker so einzustellen, dass sie eine identische Verstärkung und Phasendrehung der beiden Signale verursachen. Dadurch entstehen am Ausgang der Verstärker nicht die im Funktionsgenerator eingestellten Signale. Daher wurden die am Funktionsgenerator eingestellte Phase so lange verändert, bis auf einem Oszilloskop beide Sinusschwingungen einen Phasenunterschied von 45 ° aufwiesen. Diese Spannungsverläufe wurden dann mit zwei Zellensensoren und Erweiterungsmodulen vermessen.

Dazu wurden über des Batteriesteuergerätes an beide Sensoren mittels Broadcast das Burst-Signal gesendet und anschließend nacheinander die Daten über die RS-232 Schnittstelle des Batteriesteuergerätes an einen PC gesendet. Hier wurden sie in eine Datei kopiert, um mit einem Skript ausgewertet werden zu können. Das Skript befindet sich auf der beiliegenden CD.

Die Abtastrate bei der in Abbildung 5.7 auf der nächsten Seite gezeigten Messung betrug 200 Hz und beide Sinusschwingungen wiesen eine Frequenz von 1 Hz auf. Damit ergibt sich die Phasenauflösung zu $^{360\,°}/_{200} = 1.8\,°$. Im auswertenden Skript wird mittels des in Abschnitt 2.2.2 auf Seite 55 beschriebenen Goertzel-Algorithmus die Wechselspannungsamplitude und die Phasenlage der Signale bei der vorgegebenen Frequenz berechnet. Daraus lässt sich der Wechselspannungverlauf beider Signale rekonstruieren. Die Rekonstruktion ist ebenfalls in der Abbildung zu sehen. Durch Dividieren der beiden komplexen Ausgangswerte des Goertzel-Algorithmus erhält man eine komplexe Zahl, die das Verhältnis der beiden Signale zueinander repräsentiert. Nutzt man zur Darstellung dieser Zahl die Form nach Betrag und Phase, so liefert die Phase das Ergebnis -45.2 °.

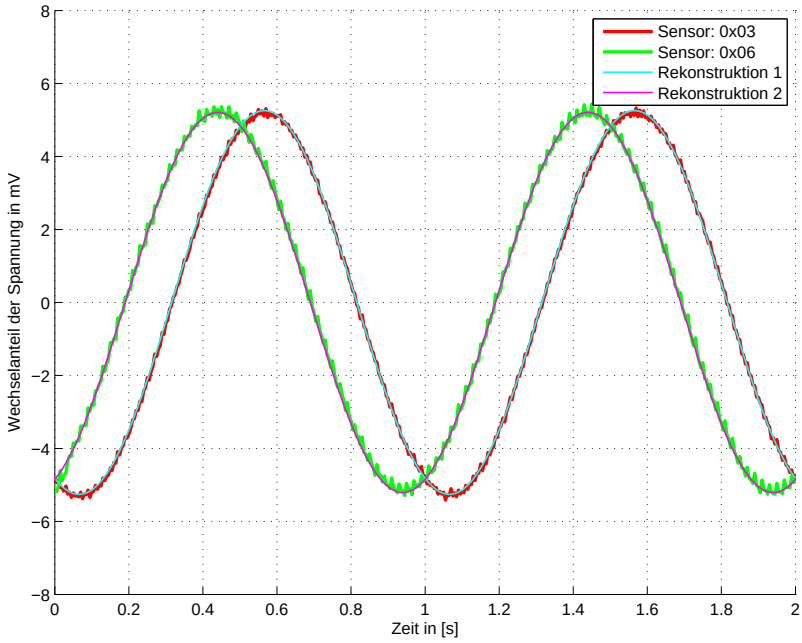


Abbildung 5.7: Gemessene Sinusschwingungen mit rekonstruierten Signalen aus dem Goertzel-Algorhytmus. Zu beachten ist, dass hier nur der Wechselanteil mit einer Amplitude von etwa 10 mV dargestellt ist. Das gemessenen Signale beinhalten auch den vollständigen DC-Offset von ca. 3.3 V. Die Phasendiferenz der beiden Signale wird zu -45.2 ° bestimmt.

Durch diese Messung konnten sichergestellt werden, dass folgende Anforderungen erfüllt sind.

- Es ist möglich mit dem Zellensensor Klasse 3 Version v0.4 und dem Erweiterungsmodul Wechselspannungsamplituden im Bereich weniger mV mit großen Offsetspannung zu erfassen.
- Der Signalverlauf kann rekonstruiert werden, auch wenn zur Reduzierung des Speicherbedarf ein fester Offset abgezogen wurde.
- Die mit dem Zellensensor aufgenommenen Spannungsverläufe zeigen die erwartete Frequenz.
- Der Goertzel-Algorhytmus ist geeignet, Wechselspannungsamplitude und Phasenlage eines sinusförmigen Signals bei einer vorgegeben Frequenz zu ermitteln.
- Aus zwei Phasenlagen lässt sich die Phasendifferenz ermitteln.

5.3 Impedanzsoektroskopie an Testimpedanz

Die Ermittlung eines Impedanzverlaufes entspricht der im letzten Abschnitt durchgeführten Messung, durchgeführt bei verschiedenen EIS-Frequenzen. Dabei muss die Abtastrate an die jeweils eingespeiste EIS-Frequenz angepasst werden, um auf der einen Seite eine ausreichende Phasenauflösung zu erreichen, auf der anderen Seite eine gute Rauschunterdrückung durch Oversampling im Δ-$Sigma$-ADC zu erreichen. Im Abschnitt 4.3.2 auf Seite 113 wurde gezeigt, dass maximal eine Abtastrate von 395 Hz zu realisieren ist, wenn die Spannungsauflösung von 10 μV eingehalten werden soll. Das Batteriesteuergerät staffelt die Auswahl der Messfrequenzen in 50 Hz Schritten, sodass sich die maximal nutzbare Frequenz auf 350 Hz reduziert. Die geforderte Phasenauflösung von 10 $^\circ$ erfordert mindestens 36 Abtastwerte pro Periode des Eingangssignals, sodass sich mit der nutzbaren Abtastrate nur EIS-Messungen bis etwa 10 Hz realisieren lassen.

5.3.1 Strommessung

Für eine EIS muss bei galvanostatischer Anregung neben der Spannungsantwort der Zellen auch die Stromanregung vermessen werden. Wird der Strom durch das Batteriesteuergerät gemessen, so muss beachtet werden, dass die Übertragung des Burst-Signals vom Steuergerät zu den Zellensensoren und die Verarbeitung des Signals einen gewissen Zeitversatz bedeuten. Dieser muss ausgeglichen werden, um die Synchronität von Spannungs- und Strommessung sicherzustellen.

Aus Zeitgründen wurde auf eine Berechnung der Ausgleichszeiten verzichtet, da diese sich für jedes Dezimierungsverhältnis ändert. Stattdessen wurde beschlossen, für die Strommessung einen der Zellensensoren umzubauen, indem der DC/DC-Wandler entfernt wird. Stattdessen wurde dieser Sensor über den 3.3 V Anschluss der JTAG-Schnittstelle versorgt. Dadurch kann der Sensor die Spannung an seinen Anschlusskontakten vermessen, ohne aus diesen gespeist zu werden. An der Software wurden geringe Änderungen vorgenommen, da die Ströme aufgrund der deutlich größeren Amplituden nicht mit 18 Bit Auflösung vermessen werden müssen. Stattdessen werden hier lediglich die oberen 16 Bit der 24 Bit Messung des ADC gespeichert. Die Messung des Stroms erfolgt als Messung des Spannungsabfalls über einen Shunt-Widerstand.

5.3.2 Aufbau der Testimpedanz

Kommerzielle Impedanz-Spektrographen verfügen für Funktionstest und Kalibrierung über Testimpedanzen mit bekanntem Frequenz-Impedanz-Verlauf. Die verfügbaren Testimpedanzen zeigen jedoch, in dem mit der aktuellen Ausbaustufe des Zellensensors erfassbaren Frequenzbereich, kein deutliches Verhalten, da sie eher im Bereich mehrerer Hz bis einigen kHz arbeiten.

Daher wurde die in Abbildung 5.8 auf der nächsten Seite gezeigte Testimpedanz entworfen und gefertigt. Diese zeigt einen halbkreisförmigen Verlauf, der zwischen 100 mHz und 25 Hz fast vollständig abgeschlossen ist. Daher wird beschlossen eine Messung bis 25 Hz durchzuführen, obwohl dabei die geforderte Phasenauflösung verletzt wird. Die unterste messbare Frequenz ergibt sich aus der Kombination der maximal speicherbaren 1900 Messwerte in einem Zellensensor und der langsamsten Burst-Frequenz von 100 Hz. Damit kann eine Messung maximal 19 Sekunden lang dauern, sodass bei einer EIS-Frequenz von 100 mHz nur noch knapp zwei Perioden der eingespeisten Spannung gemessen werden können.

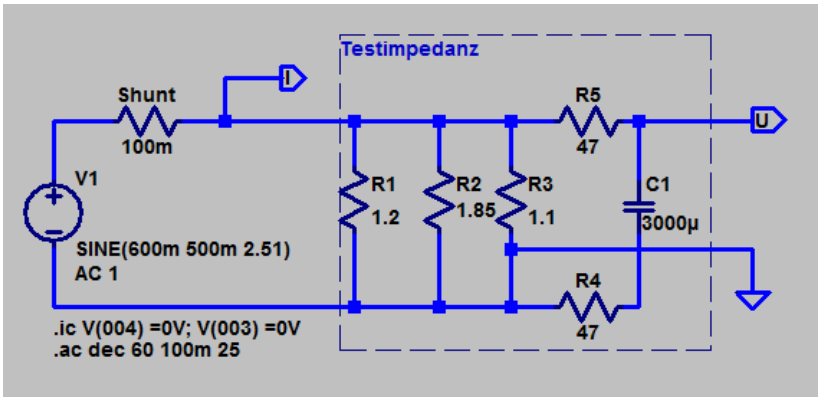
5.3.3 Anregung

Die Anregung der Testimpedanz soll über einen Strom erfolgen, der durch einen Shunt-Widerstand gemessen wird. Der Schaltplan der Testimpedanz zeigt, dass die Kapazität von den drei niederohmigen Widerständen am Eingang durch zwei 47 Ω Widerstände getrennt ist. Für eine Anregung mit Gleichstrom stellt die Kapazität einen Leerlauf dar, sodass sich der Folgende aus Richtung der Stromanregung gesehene Widerstand ergibt:

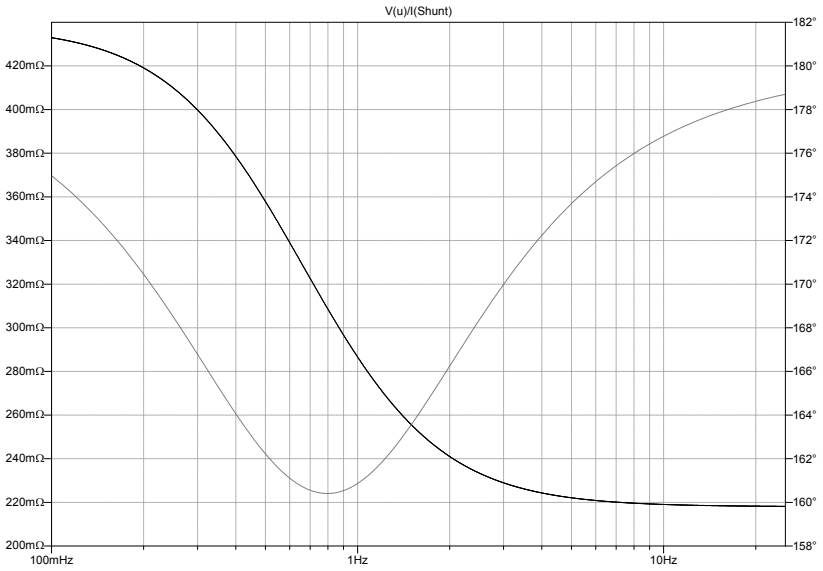
$$\begin{aligned} R_{DC} &= R_1 || R_2 || R_3 \\ &= 1.2\,\Omega || 1.85\,\Omega || 1.1\,\Omega \\ &\approx 0.438\,\Omega \end{aligned}$$

Mit steigender Frequenz verändert sich der Einfluss der Kapazität vom Leerlauf zum Kurzschluss. Für hohe Frequenzen ergibt sich eine Parallelschaltung der beiden 47 Ω Widerstände, sodass dann gilt:

$$\begin{aligned} R_{f\to\infty} &= R_1 || R_2 || R_3 || (R_4 + R_5) \\ &= 1.2\,\Omega || 1.85\,\Omega || 1.1\,\Omega || 92\,\Omega \\ &\approx 0.436\,\Omega \end{aligned}$$



(a) Schaltplan der Testimpedanz



(b) Bode-Diagramm des Impedanzverlaufes der Testimpedanz

Abbildung 5.8: Schaltplan und Bode-Diagramm der Testimpedanz mit deutlichen Änderungen im messbaren Frequenzinzerval.

Aus Sicht des Anregungssignals findet also nur eine geringe Änderung des Impedanzbetrages über den gesamten Frequenzbereich statt. Daher kann die Stromanregung mit einer Spannungsquelle erfolgen. Diese speist eine Wechselspannung im Bereich von 0 V bis 1 V in die Reihenschaltung aus 100 $m\Omega$ Shunt-Widerstand und 438 $m\Omega$ Testimpedanz ein, sodass sich ein sinusförmiger Stromfluss von ca. 2 A einstellt.

Die benötigte Spannung wird vom Funktionsgenerator erzeugt. Der Kepco 'BOP' dient erneut als Spannungsfolger, wobei jetzt seine Fähigkeit, Ströme im Amperebereich erzeugen zu können, genutzt wird.

5.3.4 Messergebnis

Innerhalb des ermittelten Frequenzbereiches von 100 mHz bis 25 Hz wurden fünf Messwerte pro Frequenzdekade aufgenommen. Für jeden Frequenzpunkt wurden fünf Messungen durchgeführt und über die daraus berechneten Impedanzen gemittelt.

Abbildung 5.9 auf der nächsten Seite zeigt ein Nyquist-Diagramm mit drei Bestandteilen. Zum einen ist eine Kurve abgebildet, die den simulierten Verlauf der Impedanz darstellt. Auf dieser Kurve sind die Impedanzen mit einem Kreis markiert, die zu den gemessenen Frequenzen gehören. Dies ist notwendig, da ein Nyquist-Diagramm ansonsten keine Informationen über die zugehörige Frequenz zu den dargestellten Punkten enthält. Farbig markiert ist jeweils der Mittelwert der fünf Messungen pro Frequenz. Es ist eine leichte Verschiebung der gemessenen Kurve hin zu größeren Realteilen und kleineren Imaginärteilen zu erkennen. Allerdings ist die Abweichung bezogen auf den Absolutwert gering. Außerdem ist zu erkennen, dass die verschiedenen Frequenzen unterschiedlich stark in ihrer Lage auf der Kurve variieren. Durch leichte Variation der Simulationsparameter erkennt man, dass der Verlauf der Kurve als solches sich kaum ändert, jedoch die Lage der einzelnen Frequenzpunkte auf der Kurve. Daher ist davon auszugehen, dass die Abweichung durch die Verwendung realer Bauteile zustande kommt, die leicht von den simulierten Bauteilwerten abweichen. Insgesamt kann festgestellt werden, dass die simulierte Kurve ausreichend gut durch Messung ermittelt werden konnte.

Im letzten Abschnitt wurde lediglich die korrekte Ermittlung der Phasendifferenz zweier manuell erzeugter Spannungen ermittelt. Diese Messung zeigt, dass durch eine Stromanregung an einer Impedanz eine Spannungsantwort erzeugt und gemessen werden kann. Mit dieser und der ebenfalls gemessenen Stromanregung konnte eine Ermittlung des Impedanzverlaufes über verschiedene Anregungsfrequenzen erfolgen, die sich ausreichend gut mit der Simulation deckt.

Es ist also möglich mit dem entwickelten System aus Zellensensor und Erweiterungsmodul eine Impedanzspektroskopie durchzuführen.

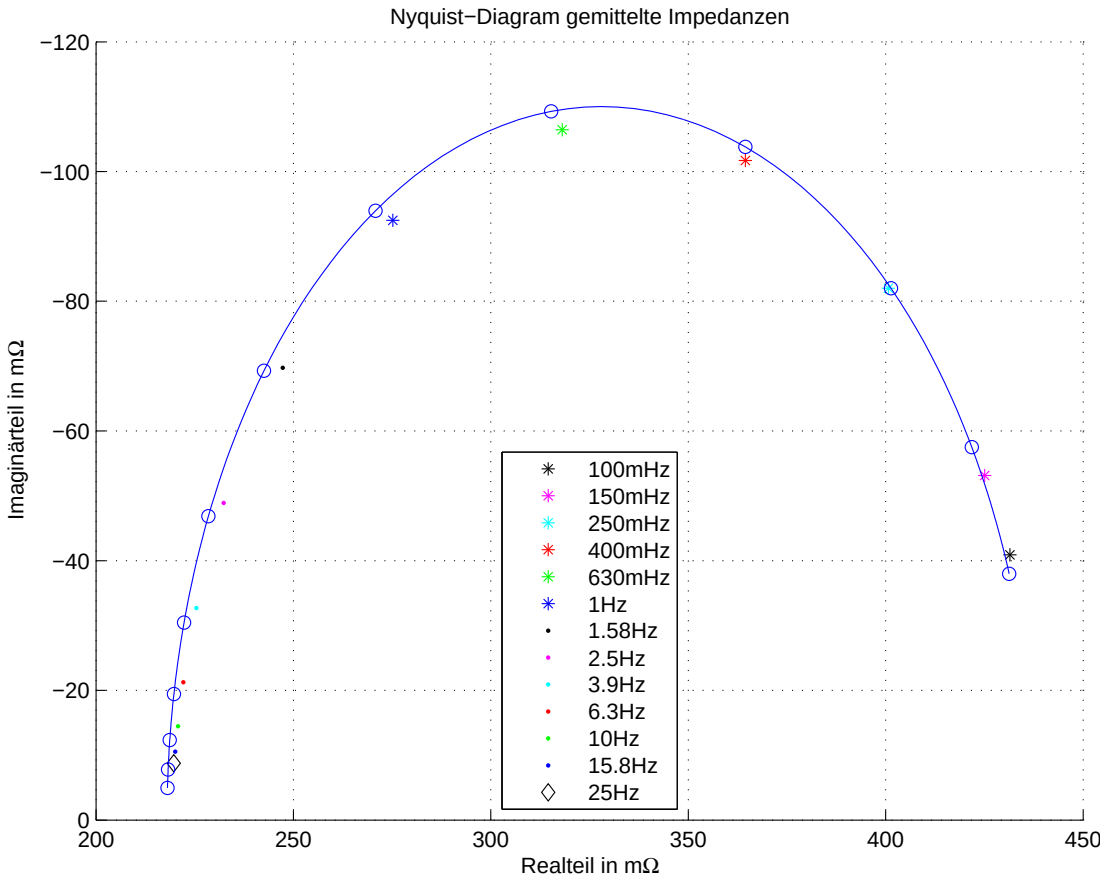


Abbildung 5.9: Vergleich zwischen simuliertem und gemessenem Impedanzverlauf der Testimpedanz in Nyquist-Darstellung.

5.4 Beispielhafte Elektrochemische Impedanzspektroskopie einer Zelle

In Abschnitt 5.2.2 wurde gezeigt, dass der Zellensensor in der Lage ist, kleine Wechselspannungen bei großem Offset synchron und phasenrichtig zu erfassen. In Abschnitt 5.3 wurde auf die geringe Signalamplitude verzichtet und eine Impedanzspektroskopie an einer Testimpedanz erfolgreich durchgeführt. In diesem Abschnitt sollen nun beide Anforderungen kombiniert werden, um eine Elektrochemische Impedanzspektroskopie an einer Lithium-Eisenphosphat Zelle durchzuführen. Dazu wird eine LiFePO4 Zelle der Firma A123-Systems mit der Nennkapazität 2.3 Ah verwendet. Der Impedanzverlauf solcher Zellen ist in [18] für eine neue Batterie vermessen worden, sodass diese Messwerte zum Vergleich herangezogen werden können. Die Strommessung erfolgt erneut mit dem umgebauten Zellensensor über einen Shunt-Widerstand. Dafür wurde aufgrund der zu erwartenden Ströme der 1 Ω Widerstand aus [6] verwendet, der bis zu 200 W Leistung abführen kann.

5.4.1 Anregung

Der verwendete ‚Bipolar Operational Power Supply Amplifier' der Firma Kepco lässt sich in allen vier Quadranten betreiben, d.h. er kann sowohl als Quelle für Strom und Spannung als auch als Senke für beides genutzt werden (vgl. [6]). Für den Betrieb als Stromsenke gilt die in Abbildung 5.10 gezeigte Grenze für den Strom der maximal aufgenommen werden kann. Aus diesen Informationen ergibt sich für den maximalen Strom bei Verwendung einer einzelnen $LiFePO_4$-Zelle mit der maximalen Ruhespannung 3.6 V:

$$\begin{aligned} I_{Cell} &= \frac{I_{Min} - I_{Max}}{U_{Max} - U_{Min}} \cdot U_{Cell} + I_{Max} \\ &= \frac{-1.7A - -5}{36V - 0V} \cdot U_{Cell} - 5A \\ &= 0.0917\Omega^{-1} \cdot 3.6V - 5A \\ &\approx -4.7A \end{aligned}$$

Die zu erwartenden Impedanzen liegen im Frequenzbereich, der mit den Zellensensor gemessen werden kann, oberhalb von 10 $m\Omega$, sodass mit diesem Strom Spannungsabfälle im niedrigen mV Bereich erzeugt werden können.

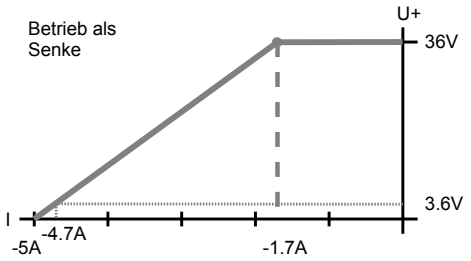


Abbildung 5.10: Maximale Stromaufnahme des BOP Amplifiers im Betrieb als Stromsenke.

5.4.2 Bestimmung der möglichen Wiederholungen je Frequenz

Durch Testmessungen wurde festgestellt, dass zur Erzeugung einer Wechselspannungsamplitude von 5 mV ein Wechselstrom von ca. 100 mA notwendig ist. Das deutet auf einen deutlich größeren Innenwiderstand als den erwarteten hin. Daher wurde die Messung mit einem Wechselstromanteil 100 mA bei einem Gleichstrom-Offset von 400 mA durchgeführt. Damit ist die Bedingung erfüllt, dass der Gleichstrom zum Einstellen des Arbeitspunktes deutlich größer sein soll als der Wechselstromanteil. Auf der anderen Seite wird der gesamte Entladungsstrom nicht zu groß. Der Effektivwert liegt bei $U_{eff} = 400\,\mathrm{mA} + 100\,\mathrm{mA}/\sqrt{2} \approx 471\,\mathrm{mA}$.

Während der Impedanzspektroskopie soll der SoC des Prüflings um nicht mehr als 10 % schwanken, weshalb bei der verwendeten Zelle mit 2.3 Ah eine Entladung von 230 mAh zulässig ist. Damit bleiben beim gegebenen Entladestrom ca. 30 Minuten um die Messung durchzuführen.

Die Dauer einer einzelnen Messung hängt von der Abtastrate der Messung und den aufzunehmenden Messpunkten ab. Die aktuelle Version des Batteriesteuergerätes erlaubt nur begrenzt langsame Messungen. 100 SPS ist die niedrigste derzeit wählbare Frequenz. Um die Messung bei einer Anregung mit 100 mHz durchzuführen, wird mit 100 SPS abgetastet, obwohl dies nicht notwendig wäre. Daraus resultiert, dass nach der Messung die vollen 1900 zur Verfügung stehenden Bytes an den Messwertspeicher übertragen werden müssen, obwohl jeder 20te Wert vollkommen ausreichen würde.

Die Übertragung der Messdaten erfolgt in Frames zu je 25 Messwerten. Es werden etwa zehn Messwert-Frames pro Sekunde vom Zellensensor über das Batteriesteuergerät an den PC gesendet. Ein solcher Frame geht mit einer gewissen Wahrscheinlichkeit, die hier mit 1 % angenommen werden soll, verloren. In einem solchen Fall gibt das Batteriesteuergerät eine Meldung aus und arbeitet weiter. Es besteht keine Möglichkeit den verlorenen Frame gezielt anzufordern, sodass der gesamte Datensatz erneut angefordert werden muss. Daraus resultiert, dass die Messungen für die besonders viele Messpunkte aufgenommen wurden doppelt kritisch sind. Mit der Anzahl der zu übertragende Frames steigt zum einen die Wahrscheinlichkeit, dass der gesamte Datensatz erneut angefordert werden muss exponentiell, zum anderen ist die Dauer der Datenübertragung erhöht. Um zu bestimmen wie viele Messwerte bei der gegeben Zeit von 30 Minuten aufgenommen werden können, wurde ein Skript erstellt, das die genannten Zeiten berücksichtigt. Außerdem wird pro Messung noch eine Zeit von 2 Sekunden für die manuelle Eingabe der Befehle und das Speichern der gesendeten Ergebnisse kalkuliert. Tabelle 5.1 auf der nächsten Seite gibt für die zuvor genannten Werte das Ergebnis des Skripts an. Es werden jeweils 173 Sekunden für einen Durchlauf über alle Messfrequenzen benötigt, sodass 5 Wiederholungen je EIS-Frequenz möglich sind, wenn sowohl Strom als auch Spannung von je einem Zellensensor gemessen werden sollen.

f_{EIS} in Hz	f_{Abtast} in Hz	Anzahl der Messwerte	$P_{Retransmission}$	Messdauer in s	Paketdauer in s	Gesamtdauer in s
0,15	100	1900	2,13	19,15	16,19	37,34
0,25	100	1900	2,13	19,15	16,19	37,34
0,4	100	1500	1,82	15,15	10,90	28,05
0,63	200	1500	1,82	7,65	10,90	20,55
1	200	1000	1,49	5,15	5,96	13,11
1,58	200	650	1,30	3,40	3,37	8,77
2,51	350	700	1,32	2,15	3,70	7,85
3,9	350	450	1,20	1,44	2,15	5,59
6,3	350	300	1,13	1,01	1,35	4,36
10	350	200	1,08	0,72	0,87	3,59
15,8	350	150	1,06	0,58	0,64	3,22
25	350	100	1,04	0,44	0,42	2,85
					Gesamtdauer für eine Wiederholung	172,61

Tabelle 5.1: Bestimmung der Messwertanzahl pro EIS-Frequenz. $P_{Retransmission}$ gibt an, wie oft ein Datensatz im Schnitt angefordert werden muss, bis er korrekt übertragen wird.

5.4.3 Messergebnis

Die Messung wurde mit den, in den letzten Abschnitten entwickelten, Rahmenbedingungen durchgeführt.Bei der Auswertung mittels des Goertzel-Algorithmus muss dabei darauf geachtet werden, dass der Prüfling und nicht die Anregeschaltung in diesem System die Quelle für den Strom ist. Daraus ergibt sich eine zusätzliche Phasendrehung von 180 ° zwischen Strom und Spannung, die bei der Berechnung kompensiert werden muss. Andernfalls erscheinen negative Realteile im Messergebnis.

Das Ergebnis von Messung und Auswertung ist in Abbildung 5.11 auf der nächsten Seite dargestellt. Die aufgezeichneten Daten und das Matlab-Skript, mit dem die Abbildungen erstellt wurden, finden sich auf der beigefügten CD. Zunächst fällt auf, dass sich über den betrachten Frequenzbereich die Impedanz nur wenig verändert.

Dies deckt sich mit der Messung aus [18], bei der ein kommerzielles EIS-Messgerät verwendet wurde. Aus dieser Arbeit geht hervor, dass für eine vollständige Erfassung der interessanten Teile des Spektrums ein Frequenzbereich von 5 mHz bis 20 kHz notwendig wäre.

Der mit dem Zellensensor beobachtbare Frequenzbereich liegt genau im Knick der Kurve nach dem Halbkreis (vgl. Abbildung 5.12 auf Seite 140), wo sich die Impedanz nur in sehr geringem Maße ändert.

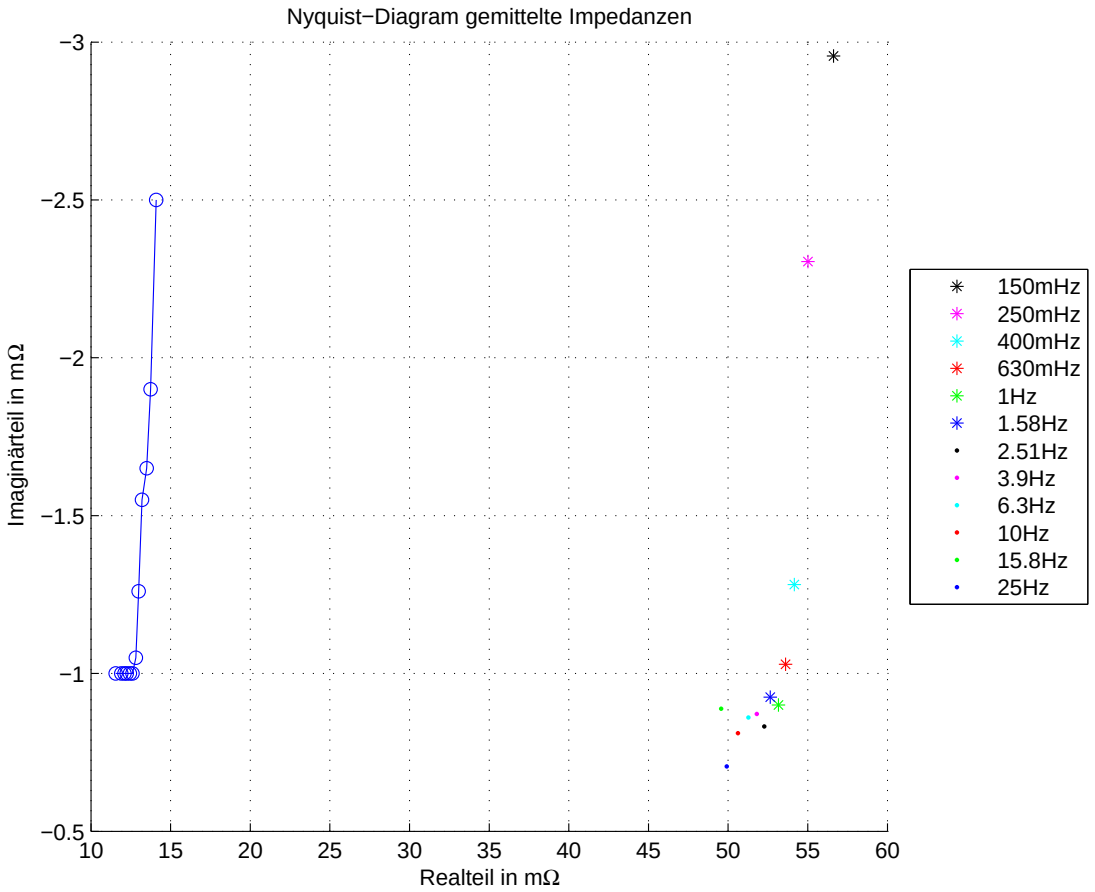


Abbildung 5.11: Ergebnis der EIS-Messung an einer LiFePO4-Zelle der Firma A123-Systems. In blau dargestellt der aus [18] entnommene Verlauf der Impedanzen, der mit einem kommerziellen EIS-Messgerät erstellt wurde. Dabei wurden aus dieser Messung die Impedanzen ausgewählt, die bei Frequezen liegen, die mit dem Zellensensor gemessen werden können. Farbig dargestellt sind die Ergebnisse der EIS-Messung mit dem Zellensensor. Dabei wurde für jeden Punkt der Mittelwert aus jeweils fünf Impedanzmessungen pro Frequenz gebildet.

Des Weiteren zeigt sich beim gemessenen Impedanzspektrum, dass die Realteile um den Faktor 4 größer sind als die mit einem kommerziellen EIS-Messgerät ermittelten. Die Abweichung der Imaginärteile von der Referenz ist deutlich geringer. Ebenfalls geringer fällt die Verbreiterung des Bereiches auf, über dem die Realteile verteilt sind.

Um zu erklären, wie die Abweichungen von den erwarteten Werten entstehen, soll nun betrachtet werden, wie sich das Impedanzspektrum einer alternden Lithium-Ionen-Zelle verhält. Abbildung 5.12 auf der nächsten Seite zeigt einen solchen Verlauf für eine Zelle mit einem anderen Kathodenmaterial als Lithium-Eisenphosphat. Da sich aber die Kurvenverläufe ähneln, ist zu erwarten, dass sich auch die Alterungserscheinungen ähneln.

Daraus folgt, dass mit steigendem Alter Real- und Imaginärteil bei den kleinen Frequenzen zunehmen, sodass sich der Radius des Halbbogens vergrößert. Für den mit dem Zellensensor messbaren Bereich des Knicks würde dies eine Verschiebung hin zu größeren Realteilen und Imaginärteilen bedeuten, wobei die Verschiebung des Realteil deutlich ausgeprägter ausfallen würde.

Wäre Alterung alleine für den gemessenen Effekt verantwortlich, so wäre auch die Verbreiterung des Impendanzbereiches über den Realteil mit dem gleichen Faktor behaftet wie die gesamte Verschiebung. Daher ist zu erwarten, dass Alterung nur für einen Teil der Abweichung verantwortlich ist. Während bei der Messung der Testimpedanz die Sensoren mit kurzen und dicken Leitungsstücken an der Testimpedanz, dem Shunt-Widerstand und dem Operationsverstärker verlötet waren, kamen bei dieser Messung aus Zeitgründen nur eine Verbindung mit Laborleitungen mit Bananensteckern zum Einsatz. Besonders kritisch muss in diesem Punkt die Kontaktierung der Zellensensoren selbst hinterfragt werden, hier wurden jeweils 2mm-Bananstecker in die Anschlußkontakte 'eingelegt', sodass keinerlei Anpresskontakt entstehen konnte. Daher ist davon auszugehen, dass ein Teil der Abweichung durch Übergangswiderstände aufgrund eines unzureichenden Messaufbaus entstanden ist.

Weder Alterung noch parasitäre Effekte erklären die relative Verschiebung der erfassten Punkte voneinander im Vergleich zum erwarteten Kurvenverlauf. Allerdings muss bedacht werden, dass für die einzelnen Frequenzen unterschiedliche Phasenauflösungen erreicht werden. Um den Einfluss der Phasenunsicherheit zu verdeutlichen zeigt Abbildung 5.13 die erfassten Impedanzen mit allen Positionen die innerhalb der Phasenungenauigkeit liegen.

In der Realität unterliegt auch die Amplitudenauflösung einer gewissen Unsicherheit, die aber in diesem Fall nicht berücksichtigt wurde. Es ist zu erkennen, dass die geringen Abweichungen der Positionen zueinander deutlich geringer ausfallen, als die Ungenauigkeit der Phase und somit als plausibel angesehen werden können. Aufgrund der geringen relativen Veränderung der Impedanz im erfassbaren Frequenzbereich ist es nicht sinnvoll, weitere Messungen an der Akkumulatorzelle vorzunehmen.

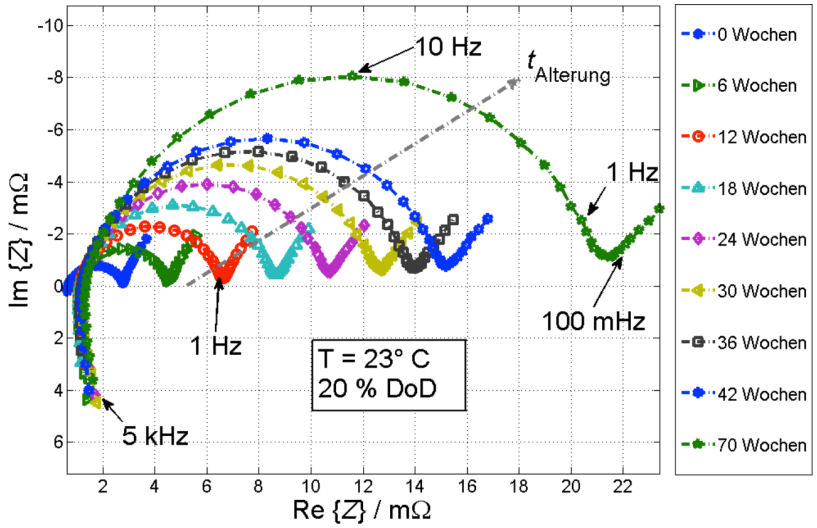


Abbildung 5.12: Veränderung des Impedanzspektrums einer Lithium-Ionen-Zelle. Das Kathodenmaterial der gezeigten Zelle ist nicht Lithium-Eisenphosphat. Entnommen aus [7]

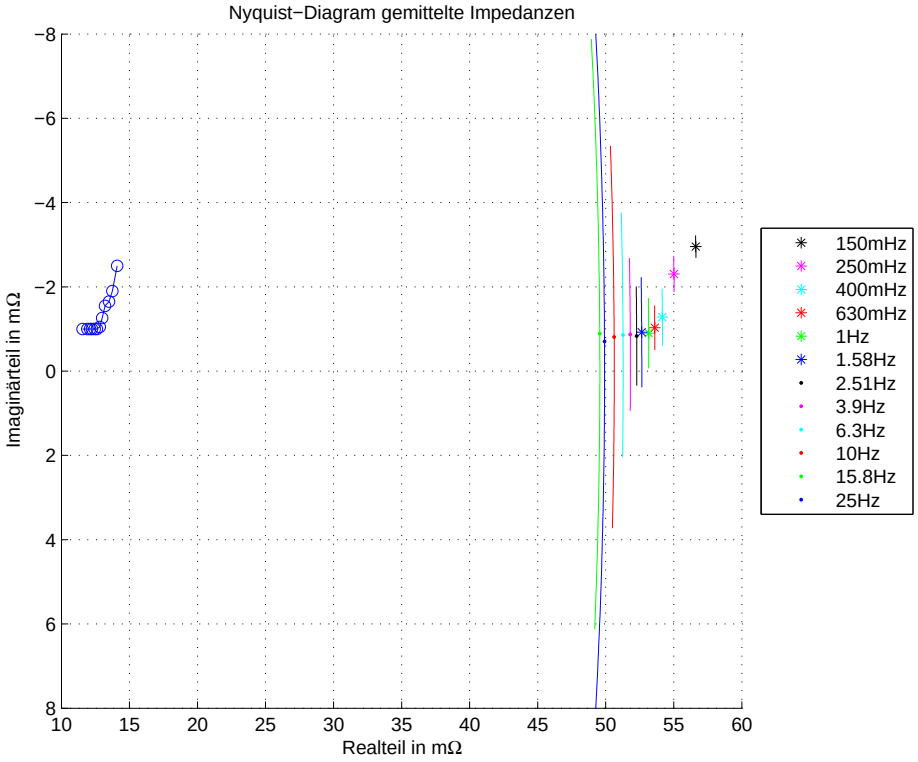


Abbildung 5.13: EIS-Messung mit eingezeichneter Phasenunsicherheit. Die Phasenauflösung ergibt sich aus der Anzahl der Messwerte pro Periode der jeweiligen Anregungsfrequenz.

Die erzielten Ergebnisse lassen einige Erkenntnisse zu:

- Bei ausreichend kleiner Anregung mit einem sinusförmigen Strom mit Offset nimmt die Spannungsantwort einer Akkumulatorzelle ebenfalls eine nahezu unverzerrte Sinusform an.
- Es ist mit dem Zellensensor möglich diesen Spannungsabfall an einer Zelles für Anregefrequenzen zwischen 100 mHz und 25 Hz zu messen.
- Der qualitative Verlauf der Impedanzspektroskopie entspricht der Referenzmessung.

5.4.4 Unterabtastung zur Erweiterung des Messbereichs

Eine Möglichkeit, um den Erfassbaren Frequenzbereich zu vergrößern, wäre die sogenannte Unterabtastung. Da bei der EIS-Messung nur eine einzige bekannte Frequenz neben dem Gleichanteil im Messsignal enthalten ist, kann vor der Messung bestimmt werden, was passiert, wenn das Nyquist-Kriterium verletzt wird. Dies passiert, wenn die Abtastfrequenz kleiner ist, als die doppelte Anregungsfrequenz.

Wenn die Frequenz der Anregung und die Frequenz der Abtastung entsprechend gewählt werden, verschiebt sich der Zeitpunkt der jeweiligen Abtastung innerhalb der Periode jeweils ein Stück, sodass über eine Reihe von Messungen der gesamte Verlauf der Kurve abgefahren wird. Es resultiert eine sinusförmige Schwingung, deren Frequenz eine andere als die Eingespeiste ist, die sich aber aufgrund der bekannten Anrege- und Messfrequenz mit der realen Anregung in Verbindung bringen lässt. Dieser Vorgang ist in Abbildung 5.14 dargestellt.

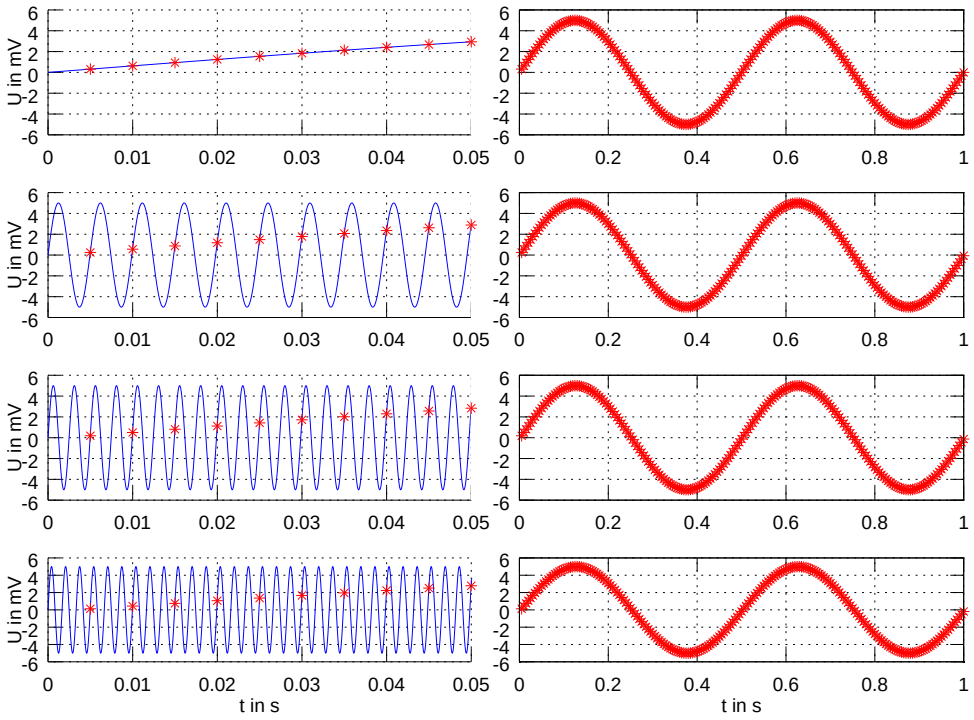


Abbildung 5.14: Unterabtastung bei monofrequentem Eingangssignal. Von oben nach unten wurden Signale mit der Frequenz 2, 202, 402 und 602 Hz mit 200 Hz abgestastet. Im obersten Fall ist das Nyquist-Kriterium also erfüllt. In allen anderen Fällen wird es verletzt und eine Alias-Frequenz entsteht, die für alle Messungen ebenfalls bei 2 Hz liegt. Rechts sieht man jeweils das resultierende Ausgangssignal, das für alle Eingänge identisch ist. Links ist ein Auschnitt der x-Achse abgebilded, und man sieht wie von oben nach unten immer mehr Perioden zwischen zwei Abtastzeitupunkten liegen.

Die Möglichkeit der Unterabtastung kann in unserem Fall nicht genutzt werden, da der Δ-Σ-ADC aus Sicht der Messwerterfassung keine echte Abtastung vornimmt. Der Quantisierer und der 1 Bit DAC sind zwar getaktet und nehmen somit eine Abtastung vor, allerdings wird in der Dezimierungs- und Filterstufe eine Mittelung über eine Reihe dieser Abtastwerte vorgenommen. Um mit dem vorhandenen ADC die gewünschten Messfrequenzen zu erreichen, müsste die Anregefrequenz um eine gute Größenordnung über der Messfrequenz liegen. Dabei wären also mehr als zehn vollständige Perioden des Anregesignals innerhalb des Bereiches, der vom ADC für die Mittelung herangezogen wird. Dadurch würde zwar weiterhin der qualitative Verlauf des Anregesignals nachvollzogen werden, durch den großen mittelwertfreien Anteil im gemittelten Signal würde aber die Amplitude des Ausgangssignals mit der Breite der Mittelung sinken.

Mit Hilfe einer Simulation in Matlab wurde untersucht wie sich die Ergebnisse einer echten Unterabtastung von einer Unterabtastung mit Mittelung über mehrere Perioden unterscheiden. Dazu wurde als Dezimierungsfilter eine einfache Mittelung nachgebildet. Es wurde eine Abtastrate von 200 Hz angenommen und die Eingangsfrequenz in 200 Hz Schritten ausgehend von 2 Hz erhöht. Somit liegen immer 100 Werte in einer Periode des Ausgangssignals, das für alle Eingangsfrequenzen als 2 Hz Ausgangssignal erscheint. Dabei wurde zum einen ein Ausgangssignal aufgenommen, das durch direkte Abtastung entsteht. Außerdem wurde die oben beschriebene Mittelung über einen Zeitbereich zwischen zwei Abtastzeitpunkten bestimmt. Abbildung 5.15 auf der nächsten Seite zeigt für die ersten sechs Frequenzschritte das Abnehmen der Amplitude bei Verwendung eines Δ-Σ-ADCs. Außerdem kann man erkennen, das es je nach Eingangsfrequenz auch zu einer Verschiebung der Phase kommt. Beide Effekte sind nicht zu beobachten, wenn eine echte Abtastung verwendet wird.

Es zeigte sich, dass die Amplitudenminderung bei etwa -20 dB pro Frequenzdekade liegt.

Da sowieso schon sehr kleine Amplituden untersucht werden, wird schnell klar, dass die Abnahme der Amplitude dazu führt, dass die Ausgangssignale für die Auswertung unbrauchbar werden. Die Unterabtastung ist mit dem Δ-Σ-ADC nicht durchführbar. Es wäre möglich, vor den Eingang des ADC ein Sample-And-Hold-Glied einzufügen, das mit dem Messtakt getaktet ist. Dann würde der Wert zum Abtastzeitpunkt über die Konversionsdauer gehalten werden, sodass bei der Unterabtastung kein Amplitudenverlust eintritt. Allerdings würde hierbei der Vorteil der Störungsunterdrückung verloren gehen, der beispielsweise dafür sorgt, dass die Störungen des DC/DC-Wandlers das Konversionsergebniss nur mäßig beeinflussen.

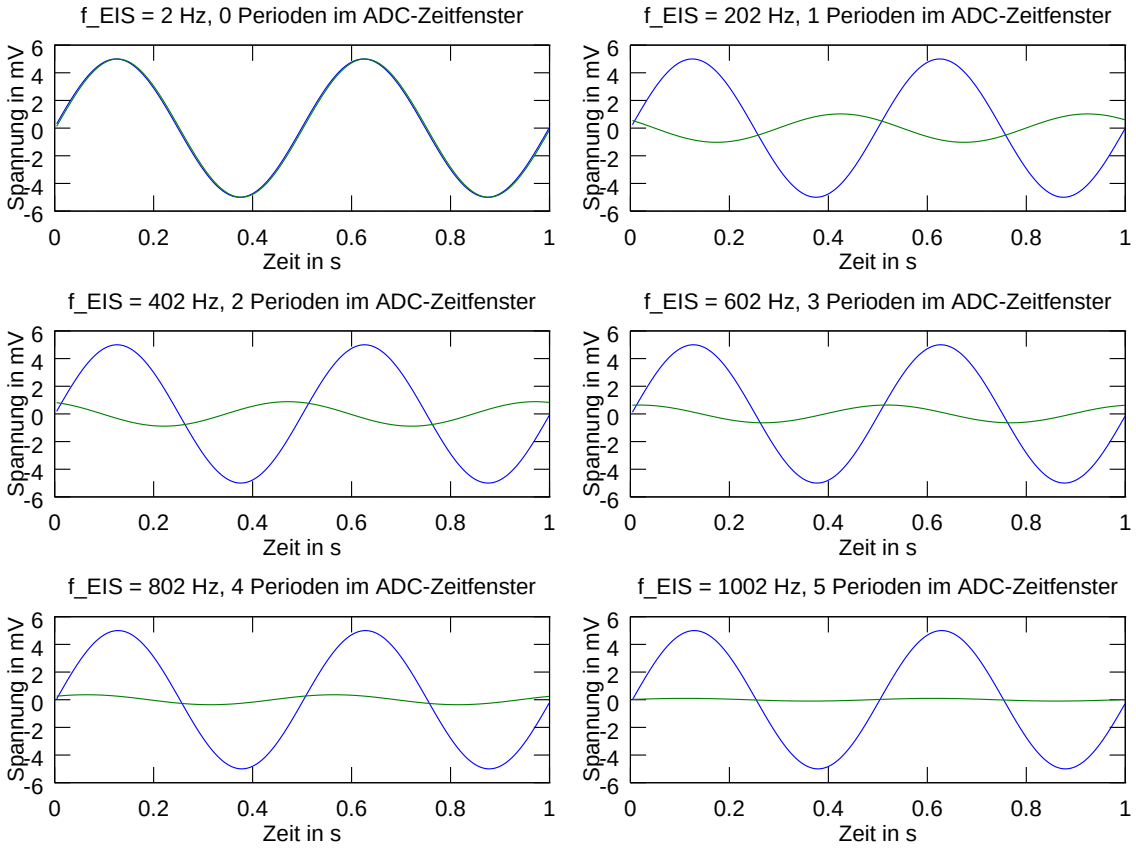


Abbildung 5.15: Unterabtastung mit und ohne Sample and Hold. Die Abtastfrequenz betrug immer 200 Hz. Durch Aliasing erscheinen alle Ausgangsignale als 2 Hz Signale. In blau wird das Ergebnis einer echten Abtastung dargestellt. In grün dargestellt ist jeweils das Ausgangsignal nach der Mittelung über die angegebene Anzahl Perioden.

6 Fazit

In diesem Kapitel wird zunächst ein kurzer Überblick über die im Rahmen der Arbeit behandelten Abschnitte gegeben. Anschließend wird Bezug auf die in der Einleitung erhobenen Fragestellungen genommen. Daraufhin wird ein Ausblick auf mögliche zukünftige Entwicklungen im Umfeld der Zellensensoren zur Elektrochemischen Impedanzspektroskopie gegeben.

6.1 Zusammenfassung

In Kapitel 1 wird, unter Bezugnahme auf Vorfälle in Flugzeugen eines Typs, der Lithium-Ionen-Akkumulatoren verwendet, dargestellt, welche Sicherheitsaspekte bei der Verwendung solcher Akkumulatortechnologien beachtet werden müssen. Es wurden Fragestellungen formuliert, deren Beantwortung innerhalb dieser Arbeit versucht wurde.

In Kapitel 2 werden notwendige Kenntnisse im Bereich der Lithium-Eisenphosphat-Akkumulatoren und der Elektrochemischen Impedanzspektroskopie vermittelt, auf die im Verlauf der Arbeit Bezug genommen wird.

In Kapitel 3 wird zunächst der aktuelle Stand der Zellensensoren der Klasse 3 des BATSEN-Projektes analysiert, um mit diesen Erkenntnissen festlegen zu können, welche Veränderungen notwendig sind, um eine Elektrochemische Impedanzspektroskopie mit den Zellensensoren durchführen zu können. Dazu wird eine Analyse der Anforderung, die eine solche Messung an einen Zellensensor stellt, durchgeführt. Mit diesen Erkenntnissen wurde ein Konzept erstellt, das beschreibt, wie mit einem drahtlosen Zellensensor die notwendigen Messungen des geringen Wechselspannungsanteils bei großem Offset eines EIS-Signals erfasst werden kann. Ebenfalls erstellt wurde ein Konzept zur Veränderung des Zellensensors der Klasse 3, sodass dieser die Möglichkeit erhält, mit einem zusätzlichen Erweiterungsmodul ausgestattet zu werden, das weitere Messaufgaben übernimmt.

In Kapitel 4 wird beschrieben, wie auf Grundlage des Konzeptes eine neue Version des Zellensensors und ein Erweiterungsmodul entworfen und gefertigt wurde. Anschließend wird in diesem Kapitel dargestellt, welche Software für die Inbetriebnahme von Zellensensor und Erweiterungsmodul erstellt wurde.

In Kapitel 5 wird beschrieben, wie die Probleme, die durch Fehler im Design des Zellensensors auftraten, bei der Inbetriebnahme gelöst wurden. Es wird weiter beschrieben, wie die neue Version des Zellensensors in Betrieb genommen und getestet wurde. Anschließend wird dargelegt, wie das Erweiterungsmodul in Betrieb genommen wurde. Dabei wird gezeigt, dass es mit dem Sensorsystem möglich ist, die geforderte synchrone Messung zweier phasenverschobener Signale auch dann durchzuführen, wenn diese Signale nur sehr geringe Amplituden haben.

Auf Grundlage dieser Erkenntnis wird eine Testimpedanz vermessen, die zu diesem Zweck entwickelt wurde. Mit dieser Messung wird gezeigt, dass eine Impedanzspektroskopie mit dem Sensorsystem, bestehend aus Zellensensor und Batteriesteuergerät, möglich ist. Dabei kann gezeigt werden, dass die Synchronität der Messung ausreichend gut ist, sodass mit Hilfe des bereits im zweiten Kapitel beschriebenen Goertzel-Algorithmus eine Auswertung der Messung möglich ist, die sich mit einer Simulation der Testimpedanz deckt.

Abschließend wird noch eine EIS-Messung an einer realen Lithium-Eisenphosphat Akkumulatorzelle durchgeführt. Dabei kann gezeigt werden, dass auch bei den hier wieder auftretenden kleinen Wechselspannung eine plausible Ermittlung der Zellenimpedanzen möglich ist.

6.2 Ergebnisse

In diesem Abschnitt wird dargestellt, inwieweit die im Abschnitt 1.1 aufgestellten Fragestellungen beantwortet werden konnten.

Es ist mit der entwickelten Erweiterung des Zellensensors der Klasse 3 gelungen, Wechselspannungsanteile im Bereich weniger mV bei großen Offsetspannungen zu messen. Dabei konnten Strom und Spannungsmessung synchron durchgeführt werden. Der dazu verwendete 24 Bit Δ-Σ-ADC kann die geforderte Spannungsauflösung bei der synchronen Messung nur bis zu einer Grenzfrequenz von 350 Hz erbringen. Dadurch, dass die Phasendifferenz von Strom- und Spannungsmessung ausgewertet werden soll, ist es nötig, mehrere Abtastwerte pro Periode der Anregung aufzunehmen, als zur Erfüllung des Nyquist-Kriteriums notwendig ist. Daher sind Impedanzspektren lediglich im Bereich bis 25 Hz aufgenommen worden.

Dennoch konnte gezeigt werden, dass es mit den Zellensensoren des BATSEN-Projektes möglich ist, eine Elektrochemische Impedanzspektroskopie durchzuführen. Dabei erfolgte die Auswertung der Messergebnisse in MATlab bzw. GNU Octave, da das Batteriesteuergerät aus einer anderen Abschlussarbeit verwendet wurde, deren Quellcode nicht angepasst wurde. Allerdings steht mit dem Goertzel-Algorithmus eine Berechnungsvorschrift zur Verfügung, die durch ihre rekursive Implementierung mit geringem Speicheraufwand die Berechnung einer einzelnen Spektrallinie aus dem DFT-Spektrum ermöglicht. Dieser Algorithmus ist bereits vielfach auf Mikrocontrollern implementiert worden, weshalb es als uneingeschränkt möglich anzusehen ist, ihn zukünftig auf dem Batteriesteuergerät der BATSEN-Zellensensoren einzusetzen.

Bezogen auf die in der Aufgabenstellung in Anhang A genannten Aufträge, konnten somit die Punkte eins bis drei abgearbeitet werden. Die in Punkt vier geforderte Aufnahme von Messreihen über verschiedene SoC und Temperaturen wurde nicht mehr durchgeführt. Der durch den erstellten Zellensensor mit Erweiterungsmodul messbare Frequenzbereich gibt nur einen sehr geringen Teil des relevanten Impedanzspektrums wieder. Daher wurde entschieden, dass es dem Projekt dienlicher ist, wenn Ansätze für Verbesserungen des Sensorsystems gesucht werden, die es zukünftig ermöglichen, einen breiteren Frequenzbereich zu analysieren.

In Bezug auf die in der Einleitung gestellten Fragen zum Thema Sicherheit von Lithium-Akkumulatoren in Flugzeugen kann Folgendes festgestellt werden:

Die Implementierung einer Elektrochemischen Impedanzspektroskopie in situ, das heißt am Betriebsort des Akkumulators, erscheint nach den gewonnenen Erkenntnissen möglich. Gerade eine Implementierung für Flugzeugbatterien lässt einen deutlich größeren Spielraum

für die Wirtschaftlichkeit eines solchen Vorhabens, da Batterien und Sicherheitstechnik prinzipiell deutlich teurer sind als beispielsweise in Kraftfahrzeugen. Außerdem werden Akkumulatoren in Flugzeugen nur selten belastet, da sie nur zum Start der Hilfsturbine oder für Notfälle verwendet werden. Deshalb stehen ausreichende Zeiten zur Verfügung, in denen eine EIS-Messung durchgeführt werden kann. Jede einzelne Batterie ist bereits mit einem eigenen Laderegler ausgestattet, sodass es möglich erscheint, über eine Modifikation dieses Ladereglers auch komplexere Anregungsformen zu realisieren.

Die Analyse der Quellenlage lieferte keine aussagekräftige Literatur, die beschreibt, wie mit Hilfe von Elektrochemischer Impedanzspektroskopie eine lokalisierte Veränderung innerhalb eines Akkumulators ermittelt werden kann. Vielmehr beschränken sich alle Arbeiten darauf, Parameter zu erfassen, die einen Gesamtzustand der Akkumulatorzelle beschreiben. Dies sind vor allem der Ladezustand (SoC) und der Gesundheitszustand (SoH). Letzterer beschreibt aber eben nicht die Veränderung einzelner Bereiche der Zelle, sondern allgemeine Alterungserscheinungen wie die Abnahme der Kapazität und Zunahme des Innenwiderstandes und die damit verbundene Leistungsabgabe.

Das Wachstum eines Dendriten, das als Ursache für mindestens einen der Vorfälle an Bord einer Boeing 787 ausgemacht wurde, lässt sich also nicht durch den Vergleich regelmäßig ausgeführter EIS-Messungen verfolgen. Dies liegt daran, dass ein solcher Dendrit nur über einem Teil der Elektrodenoberfläche wächst, der im Vergleich zur gesamten Elektrode sehr klein ist.

Dennoch kann die EIS eine erfolgreiche Erweiterung des Batterie-Managements darstellen, wenn es mit ihr möglich ist, die Parameter für eine Modellierung zu verbessern. Solche Modellparameter können Aufschluss darüber geben, wie die Leistungsgrenzen der Batterie sich mit der Zeit verändern, sodass eine unzulässige Belastung der Zellen verhindert wird. Gerade eine verbesserte Erkennung des Gesundheitszustandes kann die Sicherheit im Flugzeug erhöhen. Nämlich dann, wenn das Batterie-Management-System selbstständig feststellen kann, dass eine Batterie sich derart verschlechtert hat und eine sichere Funktion beim nächsten Bedarfsfall nicht gewährleistet ist. Da zumindest ein Teil der Batterien nur für die Notfallversorgung genutzt wird, ist eine solche Verschlechterung des Gesundheitszustandes ansonsten eher schwierig zu erkennen, da die Batterien selten einem Lastfall ausgesetzt werden.

Auch wenn derzeit keine Möglichkeit gesehen wird, wie mit Hilfe von Elektrochemischer Impedanzspektroskopie oder anderen Verfahren ein Zellenkurzschluss ausgeschlossen werden kann, so ist dennoch eine erhebliche Verbesserung der Sicherheit zu erreichen, wenn statt Akkumulatoren mit Lithium-Cobalt-Dioxid als Kathodenmaterial Lithium-Eisenphosphat zum Einsatz kommt. Kommt es im Fehlerfall zu einer Überhitzung der Akkumulatorzellen, so produziert dieses Material keinen Sauerstoff, der zur weiteren thermischen Zersetzung der

Zelle führt, da der Sauerstoff im Kathodenmaterial in fester Bindung mit dem Phosphor steht. Außerdem werden keine toxischen Stoffe wie beispielsweise Cobalt freigesetzt.

Durch die niedrigere Zellenspannung haben Akkumulatoren auf Basis von Lithium-Eisenphosphat ein geringeres Energiegewicht als andere Akkumulatortypen auf Lithium-Basis. Dadurch erhöht sich das Gewicht für eine Batterie mit gleichem Energieinhalt. Außerdem liegt die Zellruhespannung abweichend von der anderer Lithium-Ionen-Akkumulatoren bei 3.3 V anstatt 3.6 V. Das Gleichspannungsnetz an Bord von Flugzeugen arbeitet nominell bei 28 V, allerdings müssen alle angeschlossenen Geräte bis 20 V Eingangsspannung fehlerfrei arbeiten. Blei-Säure- und Nickel-Cadmium Akkumulatoren sind daher mit 12 bzw. 20 Zellen ausgestattet und erreichen eine Nennspannung von 24 V. Die in der Boeing 787 verwendeten Lithium-Cobalt-Oxid Akkumulatoren verwenden acht Zellen, sodass sie eine Nennspannung von 28.8 V erreichen. Die Generatoren an APU und Triebwerken erzeugen für das Gleichspannungsnetz eine Spannung von etwa 30 V, sodass diese Akkumulatoren noch problemlos geladen werden können. Werden Lithium-Eisenphosphat-Zellen verwendet, so könnten mit acht Zellen eine Nennspannung von 26.4 V oder mit neun Zellen eine Nennspannung von 29.7 V erreicht werden. Durch den sehr breiten Arbeitsbereich der Gleichspannungsgeräte im Flugzeug ist es also möglich einen Akkumulator auf Lithium-Eisenphosphat Basis zu entwerfen, der die aktuell verwendeten Akkumulatoren ersetzt.

Die Implementierung von Elektrochemischer Impedanzspektroskopie in situ ist nicht die Lösung aller Sicherheitsprobleme. Allerdings kann sie, in Kombination mit sichereren Akkumulatortechnologien, als Erweiterung der bereits vorhandenen Messmethoden zur Zellenüberwachung hilfreich sein. Dazu müssen die gewonnenen Daten sinnvoll in eine modellbasierte Simulation der Zellen eingebunden werden, um so sicherzustellen, dass alle Zellen immer innerhalb ihrer erlaubten Grenzen betrieben werden.

6.3 Ausblick

Mit der in dieser Arbeit entstandenen Hardware konnte der Nachweis erbracht werden, dass eine Impedanzspektroskopie mit den drahtlosen Zellensensoren der Klasse 3 möglich ist. Da diese Sensoren durch die Erweiterungsmodul-Schnittstelle leicht zu erweitern sind, sollten weitere Arbeiten zum Thema Impedanzspektroskopie mit drahtlosen Sensoren zunächst eine Überarbeitung des Erweiterungsmoduls vorsehen. Hierbei sollte nach Möglichkeiten gesucht werden, die Messung der Wechselspannung mit größerer Samplerate durchzuführen.

In dieser Arbeit wurde beschlossen, eine hochaufgelöste Messung durchzuführen, die auch den Spannungsoffset einschließt. Dazu wurde der 24 Bit Δ-Σ-ADC ADS1291 verwendet, da er der einzige leicht lieferbare ADC war, der bei den vorhandenen 3.3 V Versorgungsspannung betrieben werden kann und eine Messung mit 18 unverrauschten Bit ermöglicht. Dieser Sensor erfasst aufgrund der einstellbaren Abtastraten nicht das gesamte Impedanzspektrum einer Batterie. Deshalb soll an dieser Stelle diskutiert werden, welche Möglichkeiten bestehen, um den erfassbaren Frequenzbereich zu erhöhen.

Zunächst erscheint es am logischsten, nach einem Δ-Σ-ADC zu suchen, der eine geringere Settling-Time hat, und daher nach dem Anstoßen einer Messung schneller ein Ergebnis liefern kann. Es muss betrachtet werden, wie lang diese Zeit bei der Samplerate ist, die noch die benötigten 18 rauschfreien Bits zur Verfügung stellt. Aus dem Kehrwert dieser Zeit berechnet sich dann die maximale erfassbare Messdatenrate.

Ein Δ-Σ-ADC der eine ausreichend hohe Messdatenrate erreicht und bei 3,3 V arbeitet, konnte während der Bearbeitung dieser Arbeit nicht gefunden werden. Es gibt eine ausreichend große Anzahl dieser ADCs, die dann aber zusätzliche externe Beschaltung benötigen.

Eine Alternative könnte es sein, einen Δ-Σ-ADC mit einer Sample-and-Hold Schaltung zu versehen, um dann die in Abschnitt 5.4.4 vorgestellte Unterabtastung zu verwenden, um die benötigte zeitliche Auflösung zu erreichen. Dies könnte kombiniert werden, mit der Verwendung einer Spannungslupe wie sie in Anhang F gezeigt ist. Da hierbei die Verwendung eines ADCs mit nur 12 Bit möglich ist, könnte wieder auf andere Umsetzungstechniken wie SAR-ADCs und Flash-ADCs zurückgegriffen werden. Diese bieten von Haus aus Sample-And-Hold Schaltungen in ihren Signalpfaden, sodass eine Unterabtastung genutzt werden kann.

Eine zukünftige Nutzung von Unterabtastung ist auch deshalb wünschenswert, da bei einer Messdatenrate von 10 kHz der Zellensensor selbst an seine Grenzen stößt. Mit Hilfe von Unterabtastung wäre es dann möglich die EIS mit noch größeren Frequenzen zu nutzen.

Ein weiteres Ziel sollte die Implementierung des Goertzel-Algorithmus auf einem Mikrocontroller sein. Dabei steht prinzipiell auch die Möglichkeit zur Verfügung, den Algorithmus nicht nur auf dem Batteriesteuergerät einzusetzen, sondern auf jedem Zellensensor. Wenn diese die Berechnung der Spektrallinie ihres zuvor aufgenommenen Signals selber durchführen, so reduziert sich drastisch die Datenmenge, die versendet werden muss. Die gesamte Information jeder EIS-Messung, egal welcher Frequenz und Messdatenmenge, könnte dann durch Real- und Imaginärteil der Spektrallinie bei der Anregefrequenz repräsentiert werden. Dadurch müssten nur noch zwei Integer übertragen werden, was deutliche Zeiteinsparung bedeutet.

Wenn auf diese Weise ein System auf Basis der Zellensensoren gefunden ist, mit der man Elektrochemische Impedanzspektroskopie im geforderten Frequenzbereich mit der notwendigen Auflösung durchführen kann, so ist diese durch geeignete Messungen mit kommerziellen EIS-Metern zu vergleichen, um eine eine verlässliche Aussage über die Leistungsfähigkeit des Gesamtsystems zu treffen.

6.4 Schlussbemerkung

Als persönliches Fazit kann ich feststellen, dass mir die Arbeit im BATSEN-Projekt eine Reihe neuer Erfahrungen ermöglicht hat, von denen ich meiner Meinung nach in Zukunft profitieren kann. Zum einen konnte ich feststellen das die Arbeit in einem Projekt mit vielen Mitarbeitern große Vorteile bringt, auch wenn man als Einziger mit einer bestimmten Tätigkeit vertraut ist. Oft konnte ich Fragen, die ansonsten eine längerer Recherche erfordert hätten mit den anderern Projektteilnehmern erörtern und so zu einer Lösung kommen. Auf der anderen Seite war es erfrischend den Fortschritt anderer Arbeiten zu verfolgen und mit den Studierenden zu ihren Themen zu diskutieren.

Am Ende konnte ich nicht alle Arbeitspakete abarbeiten, die ich in Zusammenarbeit mit dem Erstprüfer vor der Anmeldung dieser Arbeit aufgelistet habe. Dennoch glaube ich, dass meine Ergebnisse innerhalb des Projektes Anwendung finden werden und somit einen wichtigen Beitrag leisten. Ob Elektrochemische Impedanzspektroskopie in situ zukünftig kommerzielle Anwendungsgebiete erobern kann, steht meiner Meinung nach noch nicht fest. Ich werde diesen Forschungszweig zukünftig im Auge behalten, um mich über neue Erkenntnisse und Anwendungen zu informieren.

Um meine Ausbildung zum Ingenieur abzurunden, hat mich diese Arbeit eine Reihe von Erkenntnissen und Fähigkeiten gelehrt, auf die ich nicht mehr verzichten möchte. So hatte ich beispielsweiße vor dem beginn meiner Tätigkeit im Projekt keine Erfahrungen mit den Design und Layout von Schaltungen mit derart kleinen Bauteilen. Außerdem konnte ich einen Einblick in die Löttechnik für sehr kleine SMD-Bauteile gewinnen und eine kommerzielle Fertigungsstätte für Protoypen besuchen.

Der Ausflug in die Chemie von Akkumulatoren war für mich interessant wie herausfordernd. Ich denke das es zu den Eigenschaften eines fertigen Ingenieurs gehört, sich auch in Themengebiete einarbeiten zu können, die fernab der Ausbildungsinhalte liegen.

Insgesamt gehe ich mit einem guten Gefühl aus der Bearbeitungszeit dieser Arbeit und sehe mich bereit und in der Lage alle während der letzten fünf Jahre gewonnenen Erkenntnisse in meinem zukünftigen Berufsleben anzuwenden.

Literaturverzeichnis

[1] M. Yay, *Elektromobilität - theoretische Grundlagen, Herausforderungen sowie Chancen und Risiken der Elektromobilität, diskutiert an den Umsetzungsmöglichkeiten in die Praxis*, 1. Aufl. Pieterlen: Peter Lang, 2010.

[2] F. A. Agency, *Boeing 787 Type Certification Special Conditions 25-359-SC*, [abgerufen am 22.10.2014]. [Online]. Available: http://www.gpo.gov/fdsys/pkg/FR-2007-10-11/html/E7-19980.htm

[3] National Transportation Safety Board, „Boeing 787-8 APU Batterie deffect," *Interim Factual Report*, 2013.

[4] J. Ostrower und A. Paztor. Microscopic 'Dendrites' a Focus in Boeing Dreamliner Probe. [abgerufen am 22.10.2014]. [Online]. Available: online.wsj.com/news/articles/SB10001424127887324880504578298673566960476

[5] L. Hillermann, „Starterbatterie in Lithium-Eisen-Phosphat- Technologie parallele Zellenmodule mit Überwachungs- und Leistungselektronik ," Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[6] A. Angold, „Verfahren zur aufwandsreduzierten Elektrochemischen Impedanzspektroskopie für Starterbatterien," Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[7] M. Kiel, „Impedanzspektroskopie an der Batterien unter besonderer Berücksichtigung von Batteriesensoren im Feldeinsatz," Dissertation, Rheinisch-Westfälische Technische Hochschule.

[8] P. Durdaut, „Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz ," Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[9] N. Sassano, „Hard- und Softwareentwicklung für einen drahtlos kommunizierenden Batterie-Zellensensor mit funksynchronisierter Messung," Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[10] J. B. Goodenough, K. . Mizushima, P. Jones, und P. Wiseman, „A new cathode material for batteries of high energy density." *Materials Research Bulletin.*, Oktober 1980.

[11] Wikipedia. (2014) Graphen. [abgerufen am 05.11.2014]. [Online]. Available: http://de.wikipedia.org/wiki/Graphen

[12] D. Repenning, „Stand und Grenzen der Lithium- und Nickel- Metallhydridtechnologie für die mobile Anwendung."

[13] *Nanophosphate High Power Lithium Ion Cell ANR26650M1-B*, A123 Systems, 2011, Datenblatt. [Online]. Available: http://www.a123systems.com/Collateral/Documents/English-US/A123%20Systems%20ANR26650%20Data%20Sheet.pdf.

[14] M. A. Roscher, J. Vetter, und D. U. Sauer, „Characterisation of charge and discharge behaviour of lithium ion batteries with olivine based cathode active material," *Journal of power sources*, Vol. 191, 2009.

[15] D. Repenning, „Entwicklung einer Kathode für einen Lithium-Eisenphosphat-Akku."

[16] O. Toprakei, O. Toprakei, L. Ji, und X. Zhang, „Fabrication and Electrochemical Characteristics of LiFePO4 Powders for Lithium-Ion Batteries," *KONA Powder and Particels Journal*, Vol. 28, 2010.

[17] P. Balakrishnan, R. Ramesh, und T. Prem Kumar, „Safety mechanisms in lithium-ion batteries," *Journal of power sources*, Vol. 155, 2005.

[18] W. Mielke, „Modellierung von Kennlinien, Impedanzspektren und thermischem Verhalten einer Lithium-Eisenphosphat-Batterie," Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[19] W. G. Bessler, „Electrochemistry and transport in solid oxid fuel cells."

[20] M. Ender. Mikrostrukturmodellierung von Lithiumbatterie-Elektroden. [abgerufen am 25.10.2014]. [Online]. Available: https://www.iwe.kit.edu/forschung_3025.php

[21] F. Quantmeyer, J. Kießling, und X. Liu-Henke, „Modellbildung und Identifikation der Energiespeicher für Elektrofahrzeuge ."

[22] M. A. Roscher, „Zustandserkennung von LiFePO4-Batterien füur Hybrid- und Elektrofahrzeuge," Dissertation, Rheinisch-Westfälische Technische Hochschule.

[23] Bremach Reisemobile.org. LiFePO-Akkus. [abgerufen am 16.09.2014]. [Online]. Available: http://www.bremach-reisemobile.org/technik/technik-nach-themen/elektrik/lifepo-akkus

[24] M. D. Levi und D. Aurbach, „Frumkin intercalation isotherm - a tool for the description of lithium insertion into host materials," *Electrochemica Acta*, 1999.

[25] M. Gaberscek, R. Dominko, und J. Jamnik, „The meaning of impedance measurements of LiFePO4 cathodes: A linearity study," *Journal of Power Sources*, 2007.

[26] V. Srinivasan und J. Newman, „Existence of Path-Dependence in the LiFePO4 Electrode ," *Electrochemical and Solid-State Letters*, 2007.

[27] G. K. Singh, M. Bazant, und G. Ceder. Anisotropic surface reaction limited phase transformation dynamics in LiFePO4. [Online]. Available: http://arxiv.org/abs/0707.1858

[28] D. H. Johnson, „Origins of the Equivalent Circuit Concept: The Voltage-Source Equivalent," *Proceedings of the IEEE*, Vol. 91, 2003.

[29] Y. Li, „State-of-Charge-Bestimmung für eine Starterbatterie mit einem Kalman-Filter-Modell," Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[30] P. Keil und A. Jossen, „Aufbau und Parametrierung von Batteriemodellen," *19. DESIGN UND ELEKTRONIK-Entwicklerforum Batterien und Ladekonzept*, 2012. [Online]. Available: https://mediatum.ub.tum.de/doc/1162416/1162416.pdf

[31] A. V. Oppenheim und R. W. Schafer, *Zeitdiskrete Signalverarbeitung* -, 2. Aufl. Oldenbourg, 1989.

[32] J. Nelles. Impedanzspektroskopie. Vorlesungsskript. [abgerufen am 15.11.2014]. [Online]. Available: skriptum.net/Forschung/impedanzspektroskopie.pdf

[33] B. S. Karden, Eckhard und R. W. D. Doncker, „A method for measurement and interpretation of impedance spectra for industrial batteries," *Journal of Power Sources*, 2000.

[34] S. Buller, „Impedance-Based Simulation Models for Energy Storage Devices in Advanced Automotive Power Systems."

[35] N. Sassano, „Unveröffentlichte Arbeit," Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[36] S. Püttjer, „Diagnosefunktion für Automobil-Starterbatterien mit drahtlosen Zellsensoren," Master's thesis.

[37] R. Kube, „Drahtloses Sensornetzwerk für Fahrzeugbatterien - Kanal, Antennen und Fehlerraten ," Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[38] M. Meinzer, „Hard- und Softwareentwicklung sowie Erprobung drahtloser Zellensensoren für Fahrzeugbatterien ," Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[39] N. Jegenhorst, „Entwicklung eines Zellensensors für Fahrzeugbatterien mit bidirektionaler drahtloser Kommunikation ," Masterthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[40] V. Roscher, M. Scheider, R. Karl-Ragmar, J. Vollmer, und G. Müller, „Vehicle Batteries with Wireless Cell Monitoring," *North Sea Region Electric Mobility Network*, 2012.

[41] S. Plaschke, „Experimentalsystem für drahtlose Batteriesensorik," Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[42] T. Krannich, „Experimentalsystem für einen Sensor-Controller mit drahtloser Energie- und Datenübertragung ," Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[43] T. Eger, „Entwicklung von Hard- und Software eines Readers für drahtlose Sensorik mit Resonanzabgleich," Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg.

[44] *MSP430F2XX - Mixed Signal Microcontroller*, Texas Instruments, Juni 2007, Datenblatt. [Online]. Available: http://www.ti.com/lit/gpn/msp430f235

[45] *CC1101 - Low-Power Sub-1 GHz RF Transceiver*, Texas Instruments, Septmeber 2013, Datenblatt. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc1101.pdf

[46] *TPS61201 -Low Input VOltage Synchronous Boost Converter With 1.3-A Switches*, Texas Instruments, März 2013, Datenblatt. [Online]. Available: www.ti.com/lit/ds/symlink/tps61201.pdf

[47] *AS3930 - Single Channel Low Frequency Wakeup Receiver*, AMS, 2013, Datenblatt. [Online]. Available: http://www.ams.com/eng/content/download/23692/414425/file/AS3930_Datasheet_EN_v4.pdf.

[48] *ADG918 - Wideband 4 GHz, 43 dB Isolation at 1 GHz, CMOS 2:1 Mux*, Analog Devices, August 2013, Datenblatt. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/ADG918_919.pdf

[49] *TMP102 - Low Power Digital Temperature SensorWith SMBus/Two-Wire Serial Interface in SOT563*, Texas Instruments, 2012, Datenblatt. [Online]. Available: www.ti.com.cn/cn/lit/ds/symlink/tmp102.pdf

[50] V. Roscher, M. Scheider, R. Karl-Ragmar, P. Durdaut, N. Sassano, E. Mense, und S. Periguda, „Synchronisierte Messung durch Trigger-Broadcast und weitere Funktionen für drahtlose Batteriesensorik," *Beitrag zum 13.GI/ITG Fachgespräch Sensornetze*, September 2014.

[51] T. Instruments, *How delta-sigma ADCs work Part 1*, [abgerufen am 05.11.2014]. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=slyt423

[52] ——, *How delta-sigma ADCs work Part 2*, [abgerufen am 05.11.2014]. [Online]. Available: http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=slyt438

[53] *AD7176-2 -24-Bit, 250 kSPS Sigma-Delta ADC with 20 us Settling*, Analog Devices, Dezember 2013, Datenblatt. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD7176-2.pdf

[54] *ADS129X - Low-Power, 2-Channel, 24-Bit Analog Front-End for Biopotential Measurements*, Texas Instruments, Septmeber 2012, Datenblatt. [Online]. Available: http://www.ti.com/lit/gpn/ads1291

[55] W. Nasimzada2013, „Hard- und Softwareentwicklung eines Lichtleiter-Sensors für die optische Analyse des Elektrolyten von Bleibatterien ," Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg.

[56] *CC430FX1XX - MSP430 SoC With RF Core*, Texas Instruments, Septmeber 2013, Datenblatt. [Online]. Available: http://www.ti.com/lit/gpn/msp430f235

A Aufgabenstellung

Auf den Folgenden beiden Seiten ist die Aufgabenstellung dargestellt, die während der Vorbereitung auf diese Arbeit zusammen mit dem Erstprüfer erarbeitet wurde.

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

25. April 2014

Masterthesis von Eike Mense
Hard- und Softwareentwicklung für einen drahtlosen Batterie- Zellen-Sensor zur elektrochemischen Impedanzspektroskopie

Motivation

Moderne Batterie-Technologien auf Lithium-Basis werden zukünftig Einzug in viele Einsatzgebiete haben. Zu ihren überlegenen Eigenschaften gehört die günstige Energiedichte und das hohe Leistungsgewicht. Als Beispiele sind Traktionsbatterien in Elektrofahrzeugen oder Notstrombatterien in den neuen Flugzeuggenerationen zu sehen. Batterien auf Lithium-Basis benötigen eine neue Betrachtung des Risikos im Fehlerfall. Die Überwachung der Betriebsparameter sowie des Lade- und Alterungszustandes gehört zu den Maßnahmen der Risikominimierung. Das vom Bundesministerium für Bildung und Forschung geförderte Forschungsvorhaben BATSEN (drahtlose Zellensensoren für Fahrzeugbatterien) befasst sich mit neuen Ansätzen zur Batterieüberwachung. Dafür werden invasive und nicht-invasive Verfahren untersucht, um die nötigen Kennwerte zu bestimmen. Für die Messung von Zellspannung und -temperatur wurden dafür eine Reihe drahtloser Sensoren entwickelt. An die messtechnische Erfassung schließt sich eine modellbasierte Berechnung des Ladezustandes an. Um zusätzlich Rückschluss auf den Gesundheitszustand einer Batterie treffen zu können, wurden Anstrengungen unternommen, um bei einem Hochstromereignis den ohmschen (reellen) Innenwiderstand zu bestimmen. Aus der Literatur ist bekannt, dass der komplexe Innenwiderstand stärker differenzierte Aussage zulässt. Hierfür wird das Verfahren der elektrochemischen Impedanzspektroskopie (EIS) eingesetzt. Es hat sich als Messverfahren bei der Entwicklung von neuen Batterie-Technologien etabliert. Die EIS erlaubt Rückschlüsse auf verschiedene Vorgänge innerhalb der Batterie. Bisher wird für die EIS aufwendige Labortechnik eingesetzt, welche für eine Betriebsüberwachung im Einsatzfall zu aufwändig ist. Das Projekt BATSEN strebt an, diesen Aufwand deutlich zu senken. Hierfür sollen die Zellensensoren befähigt werden, EIS-Messung zu unterstützen.

Aufgabe

Herr Eike Mense erhält die Aufgabe, einen Zellsensor für die Durchführung der EIS zu erweitern. Er kann dabei auf Vorabreiten zurückgreifen, deren Ziel es war, Strom und Spannung zeitlich genau synchronisiert messen zu können sowie die benötigten großen Wechselströme in die Batterie einzuprägen. Die Vorarbeiten haben gezeigt, das erhebliche Verbesserungen in der Messauflösung des Zellensensors für die EIS benötigt werden. Insbesondere soll der Zellsensor eine Messung der Wechselspannung mit einer Amplitude unter 5mV trotz des Batteriespannungs-Offsets von bis über 4V ermöglichen. Der Zellsensor soll ein Aufsteck-Platinen (Shield-Module) erhalten, um verschiedene Messmethoden realisieren zu können. Darüber soll die Hardware des Zellsensors optimiert werden (Antennenfreiraum durch Layoutoptimierung, neue Controllervariante u.a.) . Parallel soll die Erweiterung des Steuergerätes berücksichtigt werden, um die EIS-Berechnung dort in Software durch zuführen. Es sollen exemplarische Messungen die Funktionsfähigkeit der EIS bestätigen.

Die Aufgabenstellung umfasst:

1. **Analyse der Rahmenbedingungen und Konzeption einer Lösung**
   - Erläuterung der Ziele der zellbasierten Überwachung der Batterie
   - Bezugnahme auf Anwendungsaspekte der Luftfahrt
   - Darstellung des Standes der Vorarbeiten im Projekt
   - Einführung in die Impedanzspektroskopie
   - Konzeption und Strukturierung eines Lösungsvorschlags
2. **Erweiterung und Redesign der Hardware des Sensors**
   - Modifizierter Platinenentwurf mit einem Controller-Transmitter (Texas Instruments CC430) und Aufsteckplatinen-Anschluss sowie Berücksichtigung günstiger Antennenbeeinflussung
   - Aufbau und Inbetriebnahme einer Versuchsserie von Zellsensoren
   - Entwurf, Aufbau und Inbetriebnahme von Aufsteckplatinen
3. **Umsetzung und Implementierung der Impedanzspektroskopie**
   - Phasen-richtige Aufnahme von Strom und Zellspannungen mit Zellsensor und Steuergerät
   - Algorithmen- und Softwareentwicklung für Impedanzspektroskopie, insbesondere für die begrenzten Ressourcen der Controller der Sensoren und des Steuergerätes
   - Erstellen einer Software-Struktur
4. **Erprobung, Auswertung und Bewertung**
   - Erstellen von EIS-Messreihen einer Batterie aus $LiFePO_4$-Zellen über verschiedene SoC, Temperaturen
   - Auswertung im Hinblick auf die Aussagekraft für Batteriekennwerte
   - Diskussion von Vor- und Nachteilen, gelöste und offene Punkte, Ausblick

Dokumentation

Die Fachliteratur, die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Die gesetzten Rahmenbedingungen, gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die Messergebnisse sind in exemplarischem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch einordnend zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

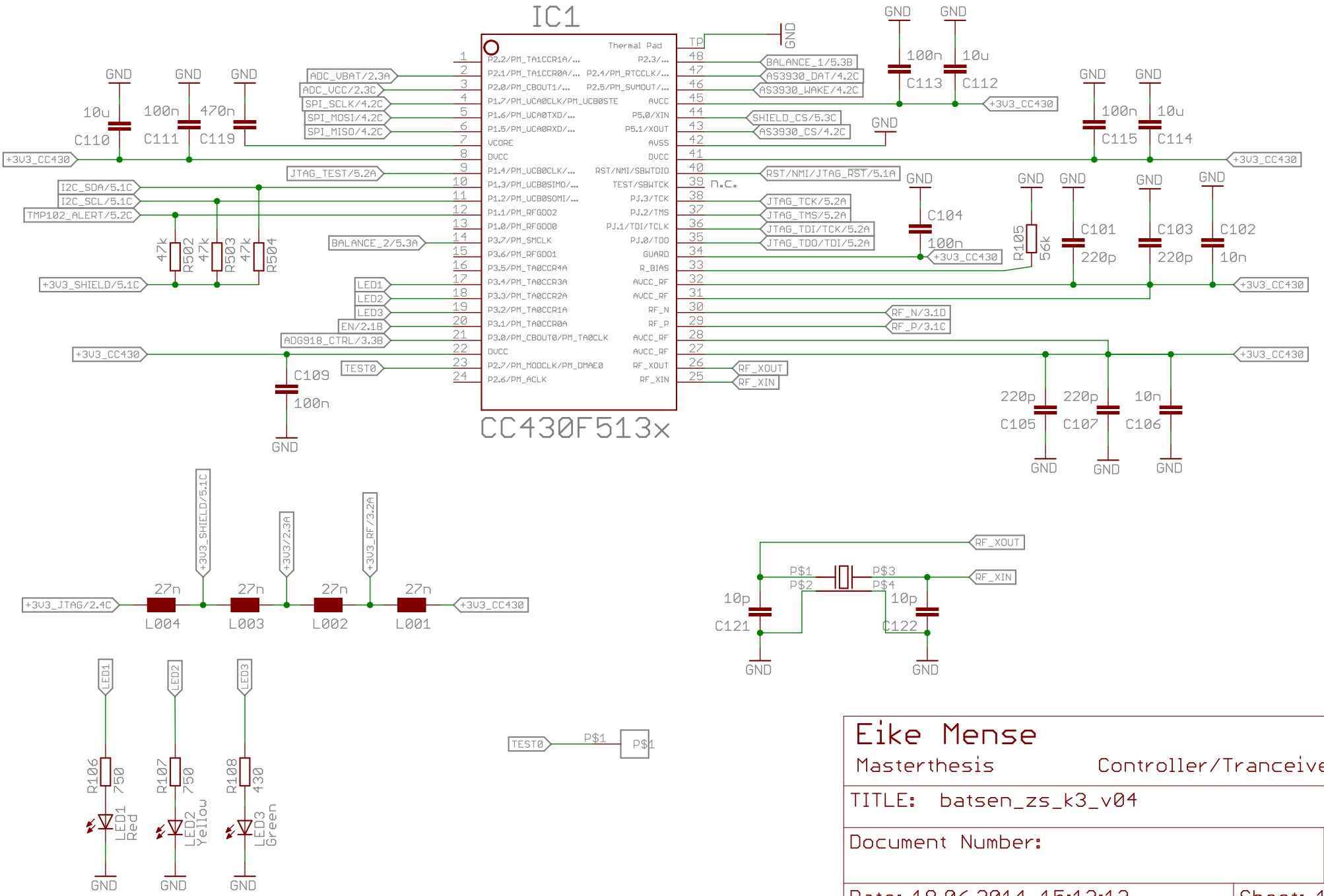
B Inhalt des Datenträgers

Der Inhalt des beigelegten Datenträgers ist wie folgt strukturiert:

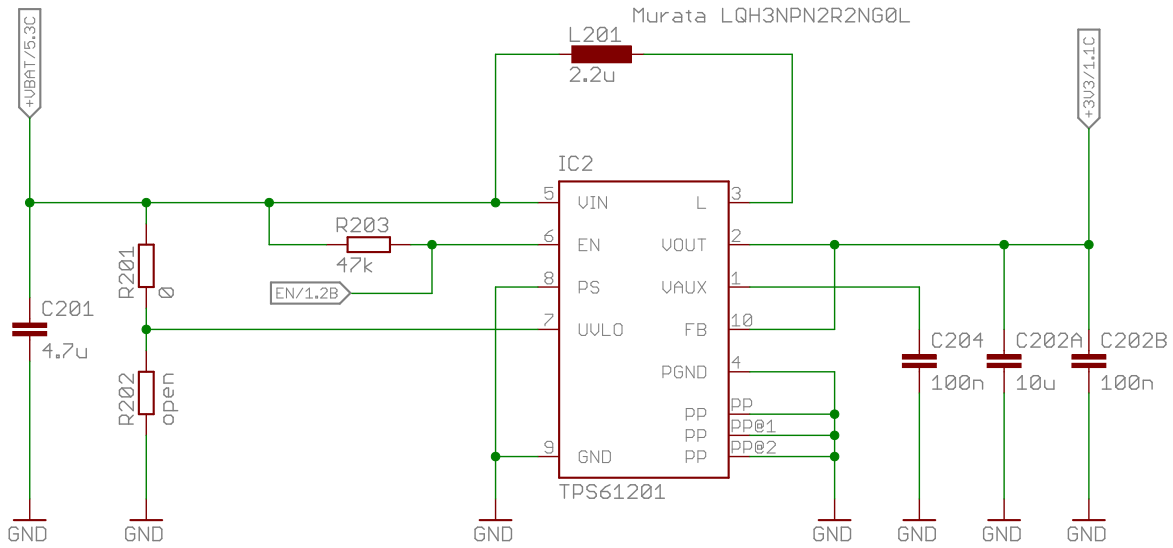
- **./01_MT_Mense2014.pdf**
  Hierbei handelt es sich um die vorliegende Arbeit im PDF
- **./02_Software**
  Dieser Ordner enthält den Quellcode der Software für Zellensensor und Erweiterungsmodul.
- **./03_Matlab**
  Diesr Ordner enthält alle Matlab-Skripte, die genutzt wurden, um Grafiken für die Arbeit zu erstellen. Außerdem sind die während der Verschiedenen Messungen aufgenommenen Datensätze enthalten.
- **./04_Modelle**
  Dieser Ordner enthält LT-Spice Modelle, mit denen simulierte Daten generiert wurden.
- **./05_Datenblätter**
  Dieser Ordner enthält Datenblätter der verwendeten ICs.

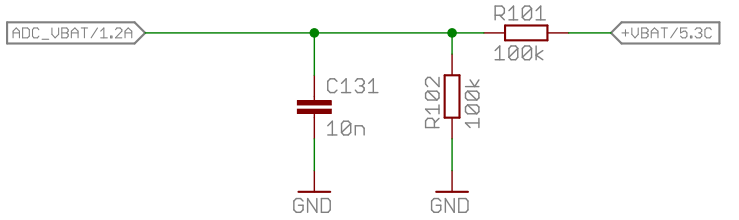
C Schaltpläne

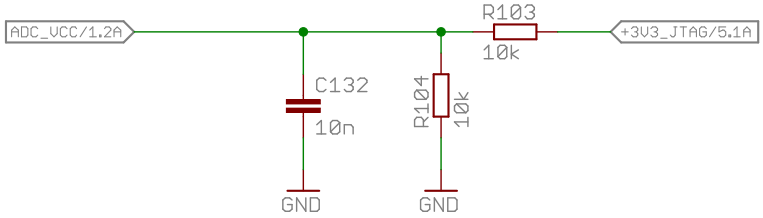
C.1 Zellensensor Klasse 3 Version 4

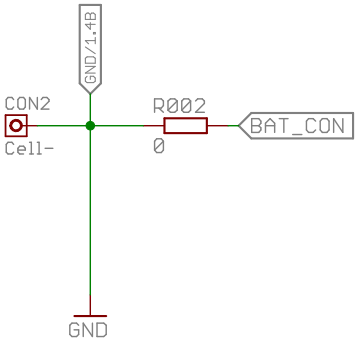


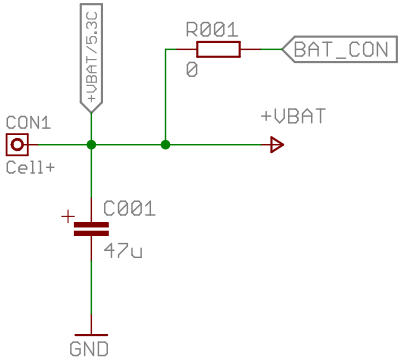
Eike Mense	
Masterthesis	Controller/Tranceiver
TITLE: batsen_zs_k3_v04	
Document Number:	REV: v0.1
Date: 18.06.2014 15:13:12	Sheet: 1/5

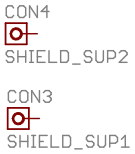












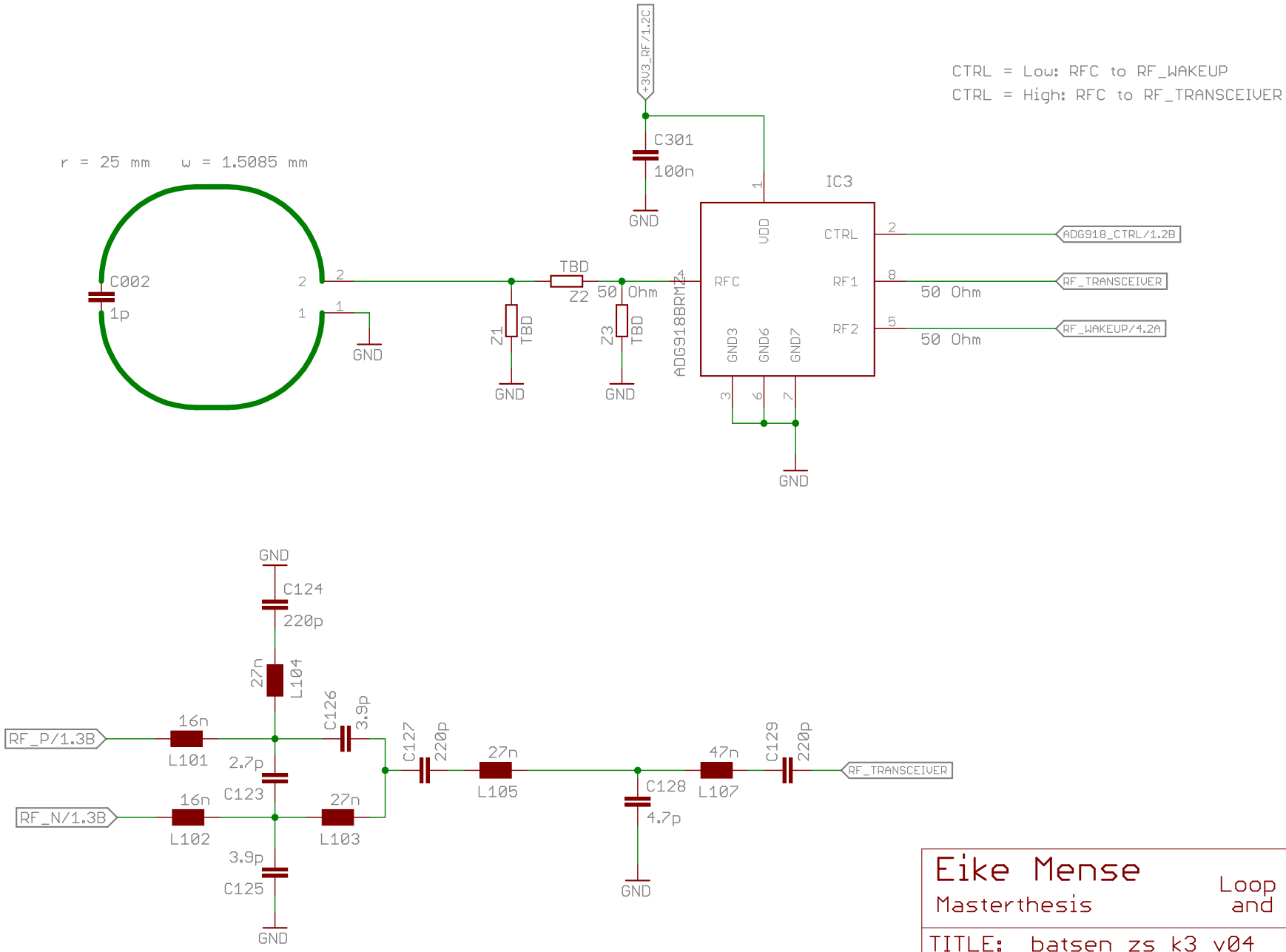
Eike Mense

Masterthesis DCDC, Cell connectors, ADCs

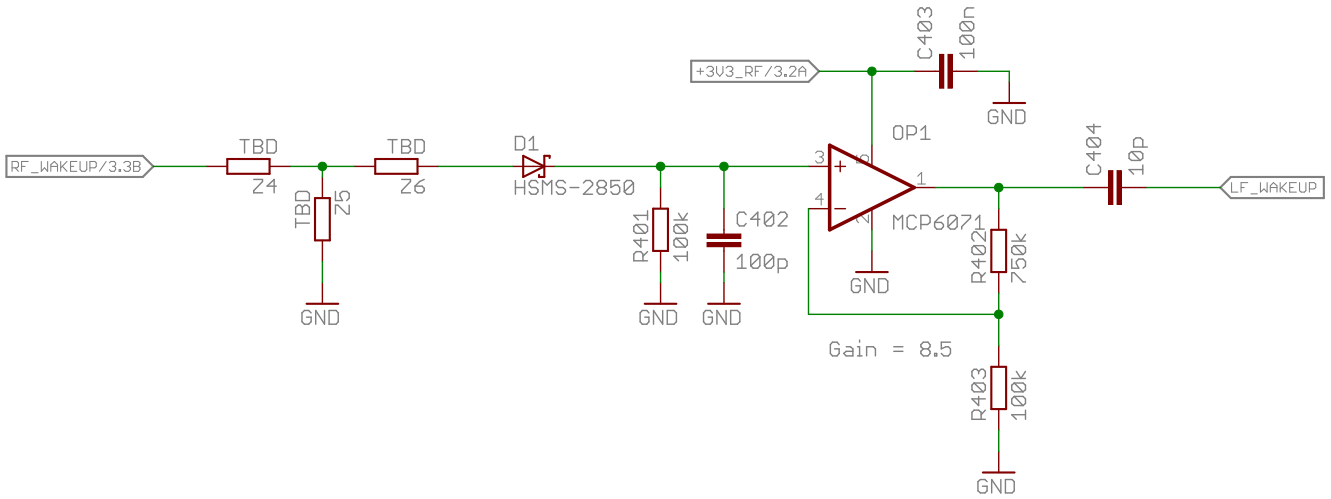
TITLE: batsen_zs_k3_v04

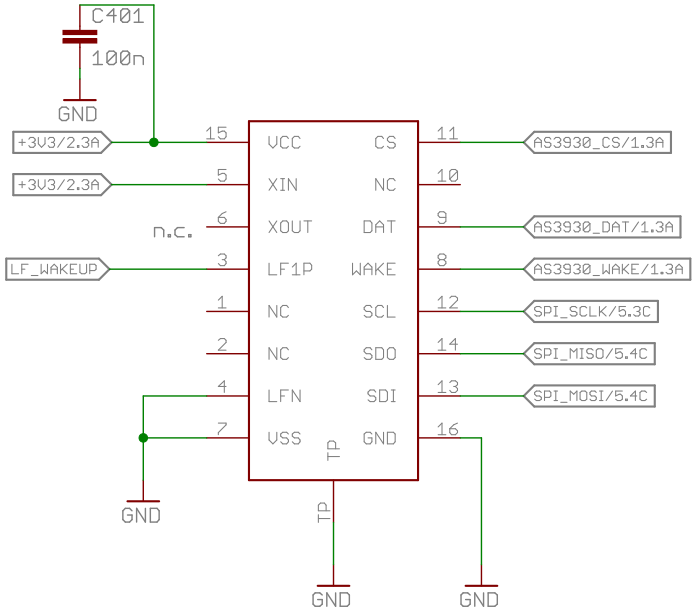
Document Number: REV: v0.1

Date: 18.06.2014 15:13:12 Sheet: 2/5

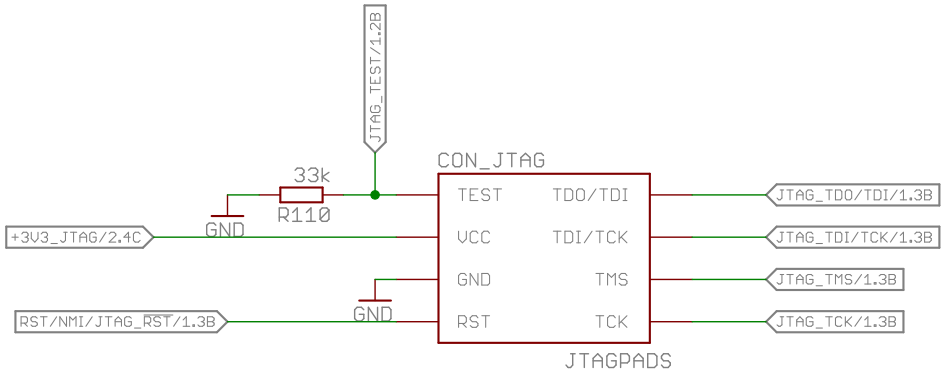


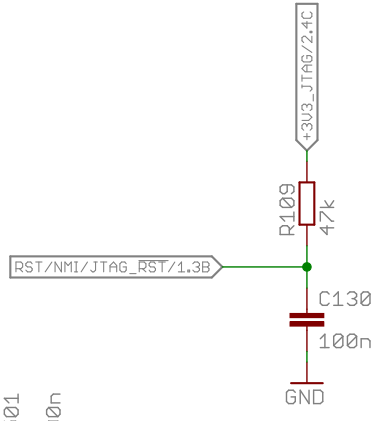
Eike Mense Masterthesis	Loop Antenna, RF Switch and Signal-Conditioning
TITLE: batsen_zs_k3_v04	
Document Number:	REV: v0.1
Date: 18.06.2014 15:13:12	Sheet: 3/5

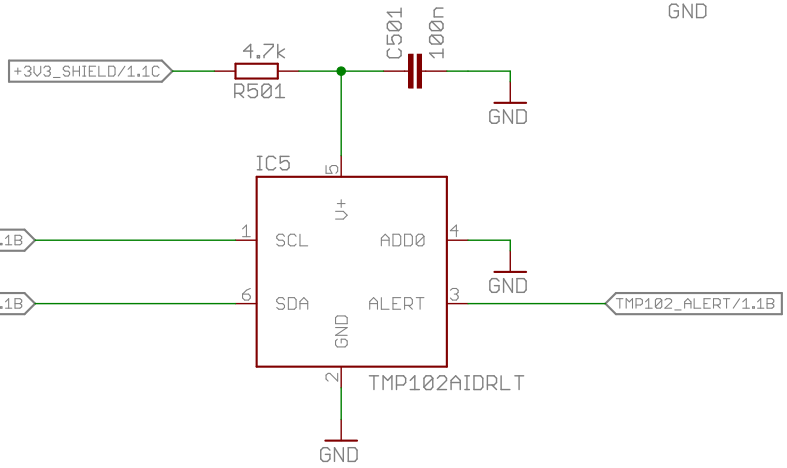




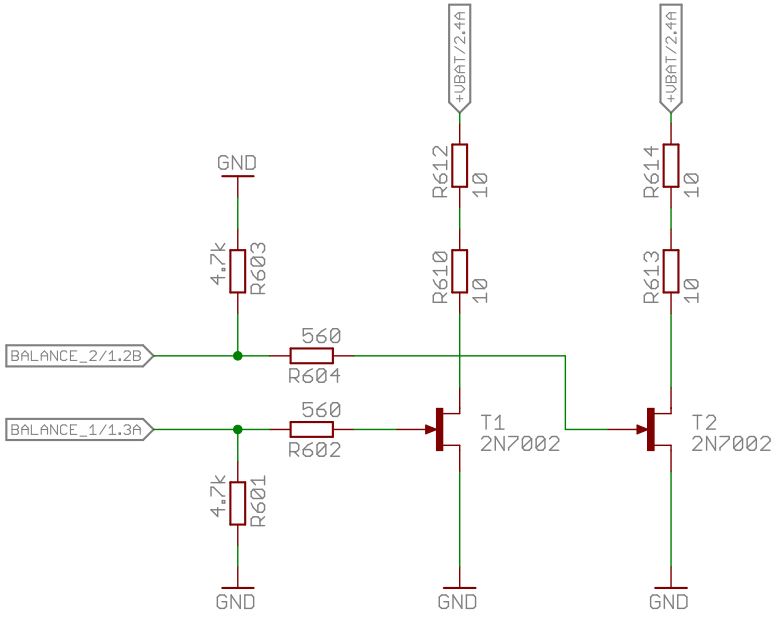
Eike Mense		
Masterthesis		Wakeup
TITLE: batsen_zs_k3_v04		
Document Number:		REV: v0.1
Date: 18.06.2014 15:13:12	Sheet: 4/5	

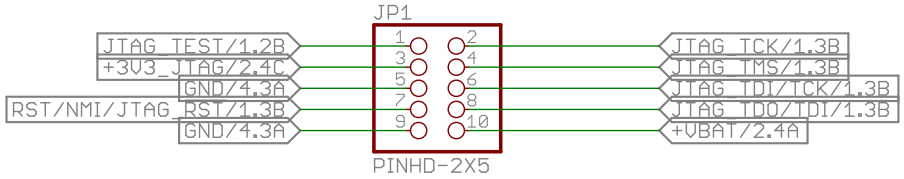


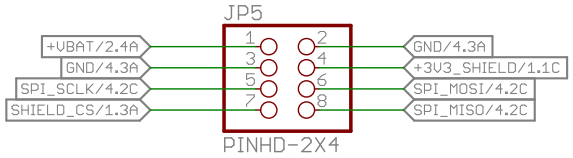




I2C Address: 0x48



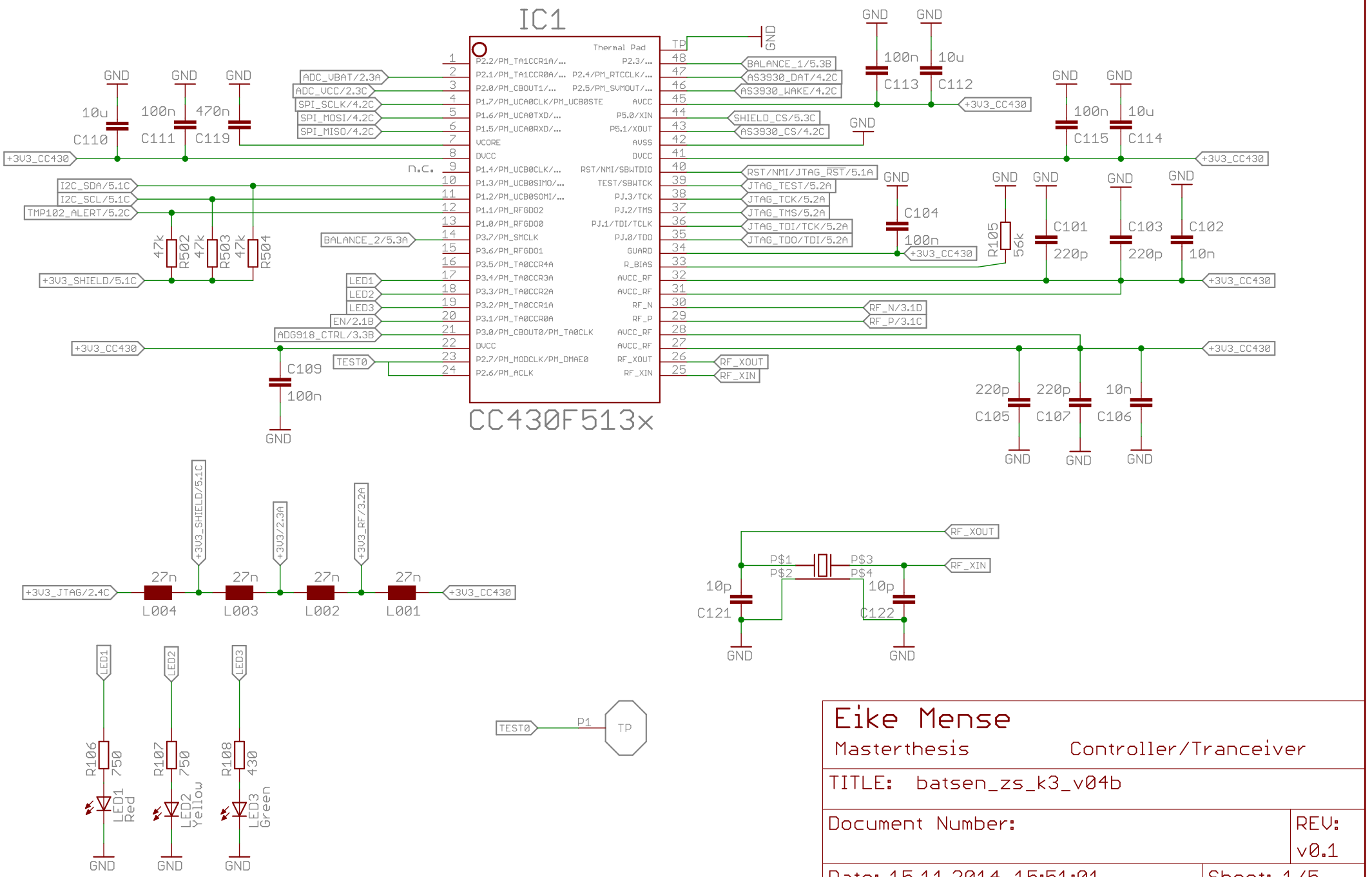




Eike Mense Masterthesis	JTAG, Temperature, Balancing, Shield
TITLE: batsen_zs_k3_v04	
Document Number:	REV: v0.1
Date: 18.06.2014 15:13:12	Sheet: 5/5

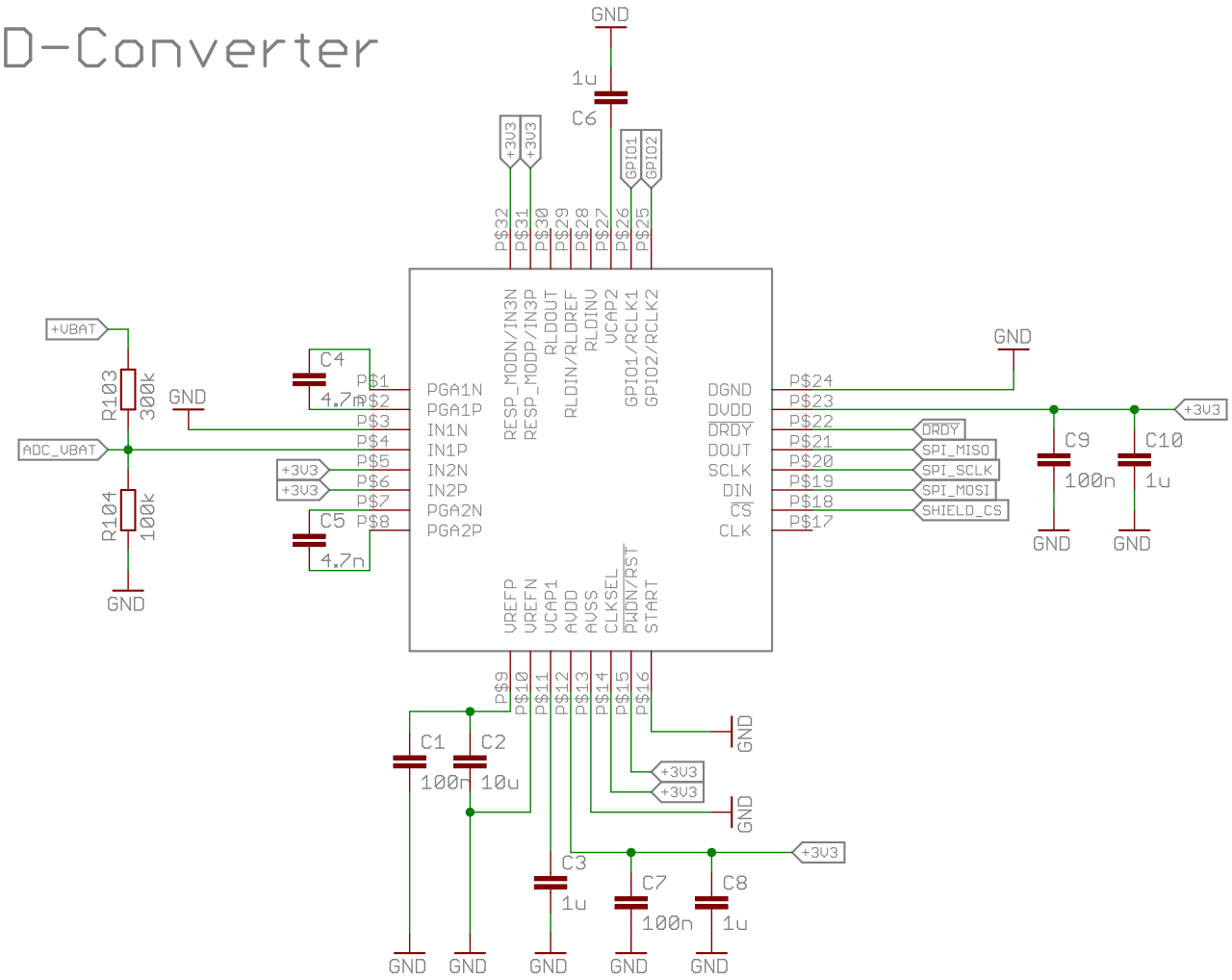
C.2 Zellensensor mit korrigierten Designfehlern

Die Seiten 2-5 des korrigierten Designs sind identisch mit den Seiten 2-5 der Orginalversion.

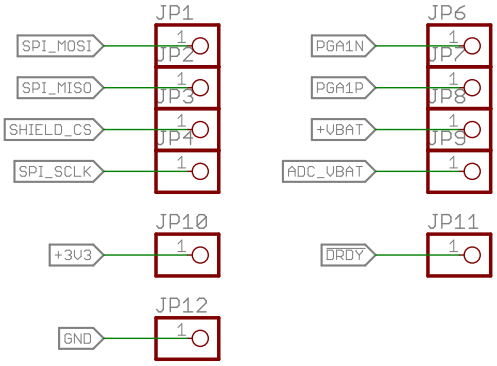

IC1
CC430F513x
Thermal Pad
TP
GND
ADC_VBAT/2.3A
ADC_VCC/2.3C
SPI_SCLK/4.2C
SPI_MOSI/4.2C
SPI_MISO/4.2C
n.c.
I2C_SDA/5.1C
I2C_SCL/5.1C
TMP102_ALERT/5.2C
BALANCE_2/5.3A
+3V3_SHIELD/5.1C
LED1
LED2
LED3
EN/2.1B
ADG918_CTRL/3.3B
TEST0
BALANCE_1/5.3B
AS3930_DAT/4.2C
AS3930_WAKE/4.2C
SHIELD_CS/5.3C
AS3930_CS/4.2C
RST/NMI/JTAG_RST/5.1A
JTAG_TEST/5.2A
JTAG_TCK/5.2A
JTAG_TMS/5.2A
JTAG_TDI/TCK/5.2A
JTAG_TDO/TDI/5.2A
RF_N/3.1D
RF_P/3.1C
RF_XOUT
RF_XIN
+3V3_CC430
+3V3_JTAG/2.4C
+3V3/2.3A
+3V3_RF/3.2A
C110
C111
C119
C109
C113
C112
C115
C114
C104
C101
C103
C102
C105
C107
C106
C121
C122
R502
R503
R504
R105
R106
R107
R108
L004
L003
L002
L001
P1
P2
P3
P4
Eike Mense
Masterthesis
Controller/Tranceiver
TITLE: batsen_zs_k3_v04b
Document Number:
REV: v0.1
Date: 15.11.2014 15:51:01
Sheet: 1/5

C.3 Erweiterungsmodul-Platine Version 1

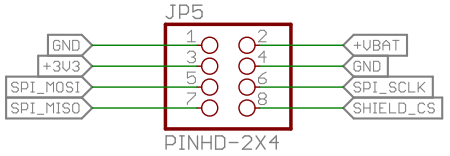
AD-Converter



Test-Points



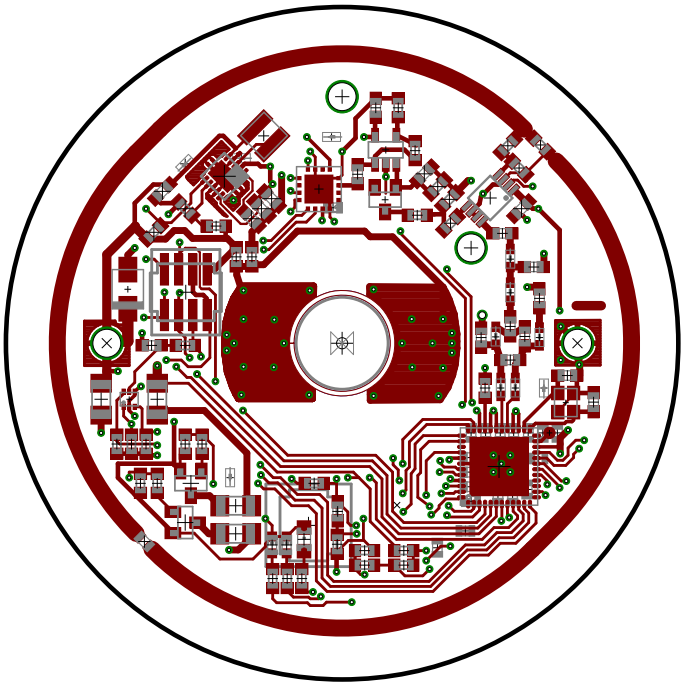
Electrical connections



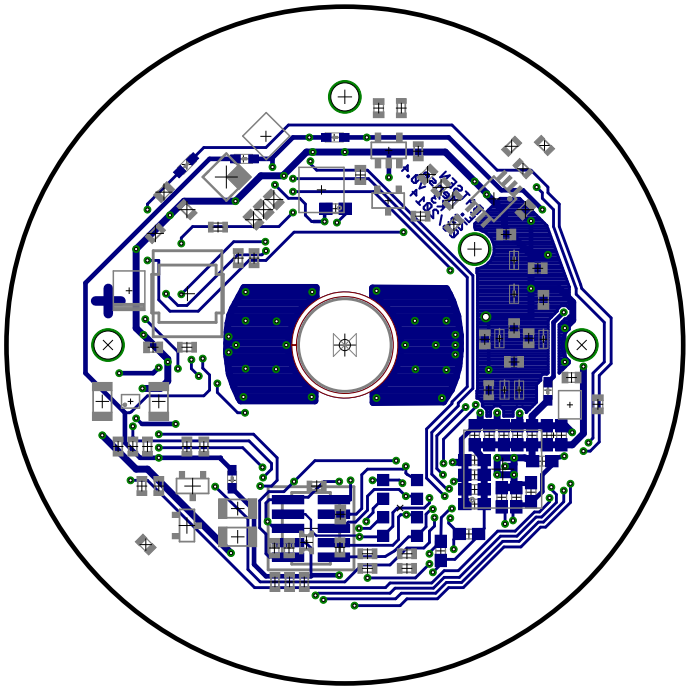
Mechanical connections

Eike Mense	
Masterthesis	SHIELD 24-BIT ?Σ ADC
TITLE: batsen_zs_k3_shield_v01	
Document Number:	REV: v0.1
Date: 05.09.2014 16:29:53	Sheet: 1/1

D Platinenlayouts

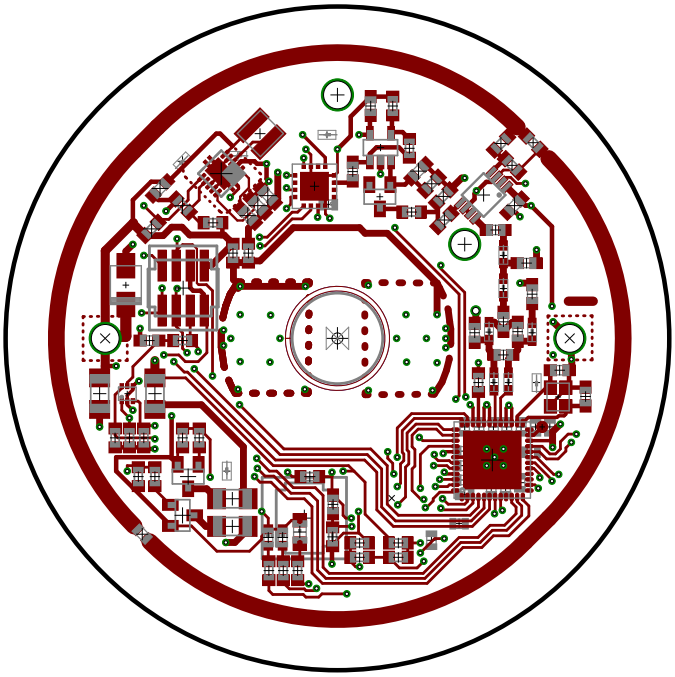


(a) Oberseite

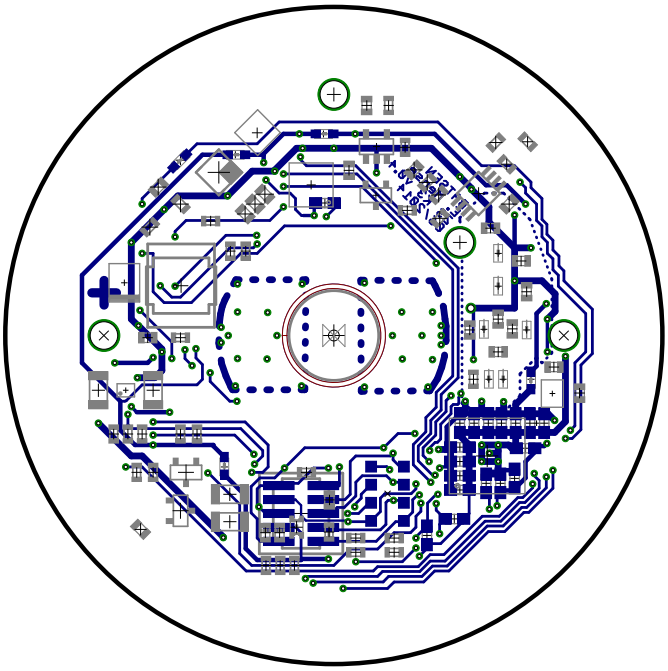


(b) Unterseite

Abbildung D.1: Platinenlayout 'Batsen ZS Klasse 3 v0.4'

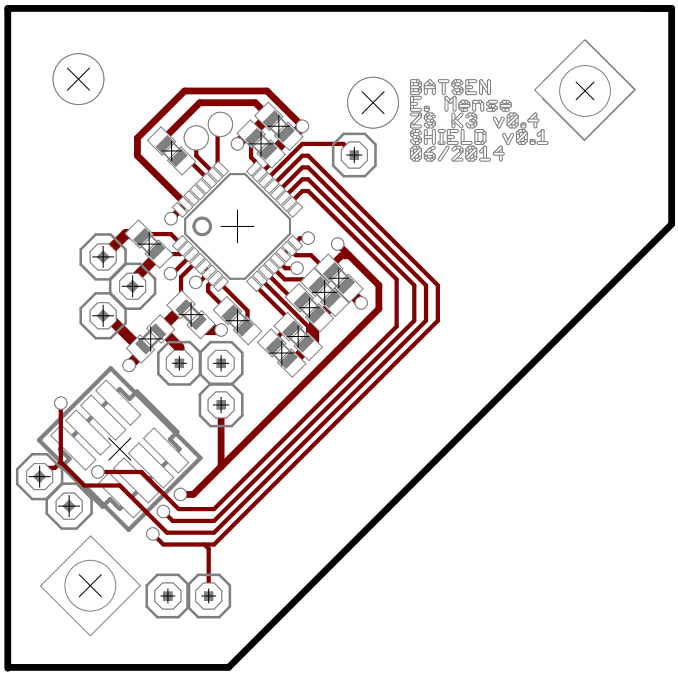


(a) Oberseite

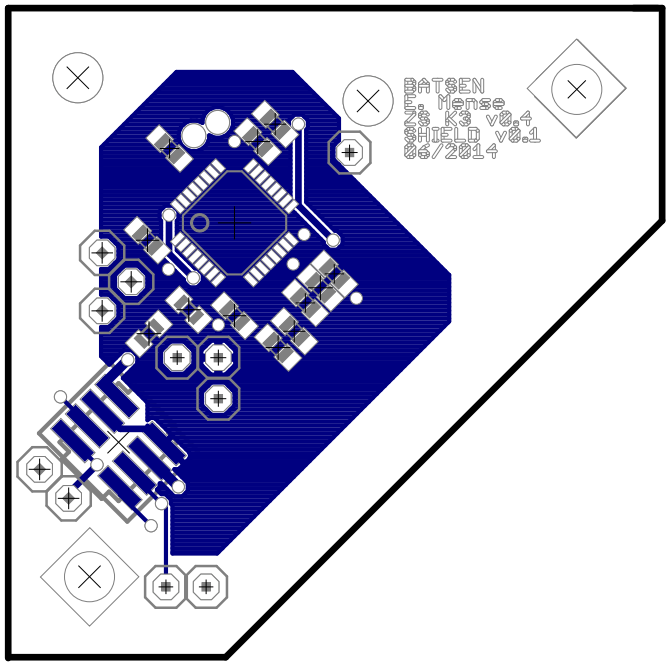


(b) Unterseite

Abbildung D.2: Platinenlayout 'Batsen ZS Klasse 3 v0.4 Revision 1' mit korrigierter JTAG-Test Verbindung und fester Verbindung von GDO0 zu einem Interrupt-Pin



(a) Oberseite



(b) Unterseite

Abbildung D.3: Platinenlayout 'Batsen ZS Klasse 3 Erweiterungsmodul v0.1'

E Quellcode

In den Tabellen E.1 und E.2 sind alle Quellcode-Dateien aufgelistet, die für den Zellensensor Klasse 3 Version v0.4 notwendig sind. Es ist jeweils aufgelistet ob die Dateien für diese Arbeit neu erstellt wurden, oder auf bereits vorhandenem Quellcode-Material beruhen. In der letzten Spalte sind jeweils die Änderungen im Vergleich zur Softwareversion aus der Arbeit von Sassano [9] aufgeführt. Im Quellcode selber ist bei allen Dateien, die geändert wurden jede Änderung mit Datum und dem Kürzel EM versehen, um zu Kennzeichnen welche Änderungen für diese Arbeit gemacht wurden.

Dateiname	Autor	Änderungs -datum	Änderungen für Zellsensor Klasse 3 Version v0.4
adc12.c	NS	04.08.14	Funktionsdeklaration in H-Datei Verschoben.
adc12.h	NS	04.08.14	Pin Definition geändert. Zwei neue Funktionsdeklarationen aus C-Datei übernommen.
adc24.c	EM	08.10.14	Neu Angelegt.
adc24.h	EM	08.10.14	Neu Angelegt.
adg918.c	PD		Keine Änderungen.
adg918.h	PD	24.07.14	Pin Definition geändert.
as3930.c	PD		Keine Änderungen.
as3930.h	PD	31.07.14	Pin Definition geändert.
balancing.c	NS		Keine Änderungen.
balancing.h	NS	24.07.14	Pin Definition geändert.
cc1101.c	NS	26.09.14	Durch cc430.c ersetzt und entfernt.
cc1101.h	NS	26.09.14	Durch cc430.h ersetzt und entfernt.
cc430.c	EM	26.09.14	Neu Angelegt. Codestruktur angelehnt an cc1101.c .
cc430.h	EM	26.09.14	Neu Angelegt. Codestruktur angelehnt an cc1101.h .
clk.c	EM	06.08.14	Alle Funktionen neu geschrieben, da CC430 anderen Takt-Kern nutzt.
clk.h	NS	06.08.14	Definition des DCO Multiplikators hinzugefügt.
delay.c	NS	01.10.14	Funkionsnamen angepasst da bei 1MHz schnellste Zählbare Zeit 20us.
delay.h	NS	01.10.14	Teilerwerte für Zählverzögerer durch Messung angepasst. Funkionsnamen angepasst da bei 1MHz schnellste Zählbare Zeit 20us.
hal_pmm.c	TI		Übernommen aus den CC430-RF-Examples
hal_pmm.h	TI		Übernommen aus den CC430-RF-Examples
i2c.c	NS	31.07.14	Generische Auswahl der I2C Pins eingefügt. War zuvor nicht notwendig, da USART-Pins beim MSP430F235 fest verdrahtet waren.
i2c.h	NS	31.07.14	Definitionen für generische Pinauswahl hinzugefügt.

Tabelle E.1: ChangeLog der Zellsensor-Software Teil 1 - PD - Phillip Durdaut [8]; NS - Nico Sassano [9]; EM - Eike Mense; TI - Texas Instruments

Dateiname	Autor	Änderungs -datum	Änderungen für Zellsensor Klasse 3 Version v0.4
init.c	NS	26.09.14	Aufruf der cc430 Funktionen anstelle der cc1101 Funktionen.
			Initialisierung des Externen ADC
init.h	NS		Keine Änderungen
isr.c	EM	08.10.14	Vollständig neu Angelegt, da Code Composer Studio 5 andere Interrupt-Vektor definitionen benötigt als die zuvor verwendete Kombination aus MPS-GCC und Eclipse.
led.c	PD		Keine Änderungen
led.h	PD	24.07.14	Pin Definition geändert
main.c	NS		Globale Variablen für ADC24 erstellt. Feste Paketgröße eingestellt. cc1101-Aufrufe in cc430-Aufrufe geändert
main.h	NS	08.10.14	cc430.h includiert cc1101.h entfernt. adc24.h includiert. Definitionen für 1000,1500 und 1900 Burst-Werte hinzugefügt. Definitionen für ADC-Auswahl hinzugefügt.
RF1A.c	TI		Übernommen aus den CC430-RF-Examples
RF1A.h	TI		Übernommen aus den CC430-RF-Examples
tempsensor.c	NS	30.07.14	i2c_init() wird nun durch te,ü_sensor_init() aufgerufen, für den Fall das dies nicht während der Sensor Initialisierung geschieht.
tempsensor.h	NS	30.07.14	Pin Definition geändert
timer.c	NS	05.08.14	Fallunterscheidung zwischen Internem 12-Bit und Externem 24-Bit für die Timereinstellung des Zeitfensters eingeführt.
timer h	NS		Keine Änderungen
types.c	PD		Keine Änderungen

Tabelle E.2: ChangeLog der Zellsensor-Software Teil 2 - PD - Phillip Durdaut [8]; NS - Nico Sassano [9]; EM - Eike Mense; TI - Texas Instruments

E.1 main.c

```
/*******************************************************************************
**     Description:     main.c
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            06/03/2013
**     Last Update:     25/09/2014 - EM
**     Author:          Nico Sassano
*******************************************************************************/

#include "main.h"

volatile uint8_t state = 0;

/*********************************************
** CLK
*********************************************/
uint8_t clk_set = 1;

/*********************************************
** RX
*********************************************/
volatile uint8_t rx_command = COMMAND_WAIT;
volatile uint8_t rx_data_length         = 0;
volatile uint8_t tx_data_length         = 0;

/*********************************************
** Balancing
*********************************************/
volatile uint16_t balancing_volt        = 0x500;  // Zielwert der Balancierung
volatile uint16_t balancing_time        = 0x000;
volatile uint8_t balanc_state           = OFF;
volatile uint8_t temp_state             = TEMP_NORMAL;
volatile uint16_t upper_balanc_temp     = 0x250;
volatile uint16_t lower_balanc_temp     = 0x200;
volatile uint16_t upper_alarm_temp      = 0x1A00;
volatile uint16_t lower_alarm_temp      = 0x1900;

/*********************************************
** Data buffer
*********************************************/
uint16_t config_value;
volatile uint16_t sample_burst_buf[BURST_VALUES];
volatile uint16_t sample_buf_volt;          // The latest cell voltage sample
volatile uint16_t sample_buf_temp;
volatile uint16_t sample_buf_temp_msp;

/*********************************************
** Burst Mode
*********************************************/
volatile uint8_t burst_freq             = 0;    // Burst Frequenz
volatile uint16_t burst_values          = 0;    // Anzahl der erwarteten Burst Werte
volatile uint16_t burst_counter         = 0;    // Counter der Burst-Werte
volatile uint8_t burst_frame_counter    = 0;
volatile uint8_t burst_frame_lenght     = 50;   // 50 -> 25 Werten
volatile uint8_t frame_number           = 0;
volatile uint8_t burst_error            = 0;
volatile uint8_t burst_error_flag       = 0;

/*********************************************
** Kalibrierung
*********************************************/
volatile uint8_t cali_pos_offset_tmp102 = 0;
volatile uint8_t cali_neg_offset_tmp102 = 0;
volatile uint8_t cali_pos_offset_msp430 = 0;
volatile uint8_t cali_neg_offset_msp430 = 0;
volatile uint8_t cali_pos_offset_adc    = 0;
volatile uint8_t cali_neg_offset_adc    = 0;

/*********************************************
** ADC24  added 25/09/2014 - EM
*********************************************/
volatile uint16_t adc24_settling_time = 0;
volatile uint8_t  adc24_mode = ADC24_SINGLE_MODE;
//volatile uint8_t adc_select = ADC12_INT_SELECT;
volatile uint8_t adc_select = ADC24_EXT_SELECT;
/*********************************************
** Interruptflags
*********************************************/
volatile uint8_t irq_mode               = TX_MODE;  // PORT1 IRQ Status
volatile uint8_t irq_send_done_flag     = 0;        // cc430 senden
volatile uint8_t irq_alert              = 0;        // TMP102 Alarm
volatile uint8_t irq_timera             = 0;        // Status TimerA
volatile uint8_t irq_timerb             = 0;        // Status TimerB
```

```
83
84  /*********************************************
85  ** Communication states
86  *********************************************/
87  volatile uint8_t command_recived        = 0;
88  volatile uint8_t wakeup_state           = 0x00;
89
90  volatile uint8_t brate_start            = 100;  // changed 25/09/2014 - EM
91  volatile uint8_t brate                  = 100;  // changed 25/09/2014 - EM
92
93  void tx_carrier(void);
94  void tx_packet(uint8_t, uint8_t *);
95  uint8_t Test_Paket[4] = {0, 0xaa, 0, 0xaa};
96  uint8_t packet[64] = {0};       // changed 25/09/2014 - EM  Alloziierung mit
97                      // Variabler Größe zur Laufzeit nicht möglich.
98                      // 64-Byte maximale Länge der FIFO
99
100
101 int main(void) {
102   WDTCTL = WDTPW+WDTHOLD;                     // Stop watchdog timer
103
104 //     P5DIR |= 0x0001;
105 //     P5DIR |= 0x0002;
106 //     P5DIR |= 0x0004;
107 //     P5DIR |= 0x0008;
108 //     P5DIR |= 0x0010;
109 //     P5DIR |= 0x0020;
110 //
111 //     P5OUT |= 0x0001;
112 //     P5OUT |= 0x0002;
113 //     P5OUT |= 0x0004;
114 //     P5OUT |= 0x0008;
115 //     P5OUT |= 0x0010;
116 //     P5OUT |= 0x0020;
117 //
118 //
119 //     while(1);
120
121     //P5SEL |= BIT5;
122     //P5DIR |= BIT5;
123
124     /*************************************************
125      ** Initialisierung
126      *************************************************/
127     init();
128
129 //    /*** Soll nicht einschlafen ***/
130 //    EXIT_LPM4;
131     // Disable and clear the wake interrupt as the sensor is awake now
132     AS3930_WAKE_IRQ_DISABLE;
133     AS3930_WAKE_CLEAR_IRQ;
134
135     // Configure packet received interrupt
136     CC430_END_OF_PKT_CLEAR_IRQ;    // changed 25/09/2014 - EM
137     CC430_END_OF_PKT_IRQ_ENABLE;   // changed 25/09/2014 - EM
138
139     // Turn on the red LED
140     led_on(LED_AWAKE);
141     //TIMER_B_START;
142
143     // Change to RX state
144     adg918_transceiver();           // Connect the loop antenne with the transceiver
145     adc12_volt_init(clk_set);       // Initialize ADC
146
147     state = S_WAKEUP;
148     //tx_packet(4, Test_Paket);
149
150
151
152     rx_data_length = 0;             // Standart Datenlänge empangen
153     cc430_set_rx(brate_start);
154     rx(HEADER_LENGTH + rx_data_length);
155
156
157 //    /*******************************/
158
159   /*************************************************
160    ** Endless loop
161    **
162    *************************************************/
163   while(1) {
164
165       if(command_recived > 0)
166         {           // Kommando wurde Empfangen
167           // Determine what the base station wants this sensor to do now
```

```
            switch(rx_command) {
169             /************************************************************
                ** Unbekanntes Kommando empfangen
171             ************************************************************/
                case COMMAND_DOWNLINK_UNKWOWN: {
173
                    rx(HEADER_LENGTH + rx_data_length);
175             } break;

177             /************************************************************
                ** Kommando zum WAKEUP empfangen
179             ************************************************************/
                case COMMAND_WAKEUP: {
181                 state = S_WAKEUP_RX;

183                 rx(HEADER_LENGTH + rx_data_length);
                } break;
185             /************************************************************
                ** Kommando WAKUP DONE empfangen
187             ************************************************************/
                case COMMAND_WAKEUP_DONE: {
189                 state = S_WAKEUP_DONE;

191                 rx(HEADER_LENGTH + rx_data_length);
                } break;
193
                /************************************************************
195             ** Senden des AWAKE Test
                ************************************************************/
197             case COMMAND_DOWNLINK_IS_AWAKE: {

199 //                  uint8_t packet[HEADER_LENGTH] = {0}; // changed (s.o.) 25/09/2014 - EM

201                 packet[0] = ADDRESS_BASE_STATION;
                    packet[1] = ADDRESS_THIS_SENSOR;
203                 packet[2] = 0x01;
                    packet[3] = 0x00;
205

207                 tx_packet(HEADER_LENGTH, packet);

209
                    rx(HEADER_LENGTH + rx_data_length);
211
                } break;
213
                /************************************************************
215             ** Einstellung zum Empfang von Konfigurationen
                ************************************************************/
217             case COMMAND_DOWNLINK_CONFIG_SET: {

219                 rx(HEADER_LENGTH + rx_data_length);
                } break;
221
                /************************************************************
223             ** Konfiguration empfangen, danach neu Konfigurieren
                ************************************************************/
225             case COMMAND_DOWNLINK_CONFIG: {

227                 /**********************************
                    ** Temperatursensor konfigurieren
229                 **********************************/
                    temp_sensor_set_alert(lower_alarm_temp, upper_alarm_temp);
231
                    rx(HEADER_LENGTH + rx_data_length);
233             } break;

235             /************************************************************
                ** Kalibrierung des TMP102
237             ************************************************************/
                case COMMAND_DOWNLINK_CALIBRATION_TMP102: {
239
                    rx(HEADER_LENGTH + rx_data_length);
241             } break;

243             /************************************************************
                ** Kalibrierung des MSP430
245             ************************************************************/
                case COMMAND_DOWNLINK_CALIBRATION_MSP430: {
247
                    rx(HEADER_LENGTH + rx_data_length);
249             } break;

251             /************************************************************
                ** Aufnahme der einfachen Spannungs- und Temp.-Messung
```

```
253             ************************************************************/
254             case COMMAND_DOWNLINK_SAMPLE_VOLTAGE_TEMPERATURE: {
255
256                 // IRQ TimerA anhalten
257                 // Da Interrupt stört NS 27.06.13
258                 if(IRQ_TIMERA_IS_BALANCING) {
259                     TIMER_A_STOP;
260                     TIMER_A_0_CM_IRQ_DISABLE;
261                 }
262
263                 if (balanc_state == OFF)     //TEST
264                     sample_buf_volt = adc12_get_volt_sample(clk_set);
265
266                 sample_buf_temp = temp_sensor_get_temp();
267                 sample_buf_temp = sample_buf_temp + cali_pos_offset_tmp102;
268                 sample_buf_temp = sample_buf_temp - cali_neg_offset_tmp102;
269
270
271                 //  IRQ TimerA vortsetzen
272                 if(IRQ_TIMERA_IS_BALANCING) {
273                     TIMER_A_0_CM_IRQ_ENABLE;
274                     TIMER_A_START_UP_MODE;
275                 }
276
277                 rx(HEADER_LENGTH + rx_data_length);
278             } break;
279
280             /************************************************************
281             ** Senden der einfachen Spannungs- und Temp.-Messung
282             ************************************************************/
283             case COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE: {
284
285 //                uint8_t packet[8];     // changed (s.o.) 25/09/2014 - EM
286
287                 packet[0] = ADDRESS_BASE_STATION;
288                 packet[1] = ADDRESS_THIS_SENSOR;
289                 packet[2] = COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE;
290                 packet[3] = 0x00;
291
292                 packet[4] = (uint8_t)((sample_buf_volt >> 8) & 0x0F);
293                 packet[5] = (uint8_t)((sample_buf_volt >> 0) & 0xFF);
294
295                 packet[6] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
296                 packet[7] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);
297
298
299                 tx_packet(8, packet);
300
301
302                 rx(HEADER_LENGTH + rx_data_length);
303             } break;
304
305             /************************************************************
306             ** Aufnahme der einfachen Spannungs- und Temp.-Messung
307             ************************************************************/
308             case COMMAND_DOWNLINK_SAMPLE_VOLTAGE_TEMPERATURE_ALL: {
309
310                 // IRQ TimerA anhalten
311                 // Da Interrupt stört NS 27.06.13
312                 if(IRQ_TIMERA_IS_BALANCING) {
313                     TIMER_A_STOP;
314                     TIMER_A_0_CM_IRQ_DISABLE;
315                 }
316
317                 sample_buf_volt = adc12_get_volt_sample(clk_set);
318
319                 sample_buf_temp = temp_sensor_get_temp();
320                 sample_buf_temp = sample_buf_temp + cali_pos_offset_tmp102;
321                 sample_buf_temp = sample_buf_temp - cali_neg_offset_tmp102;
322
323                 sample_buf_temp_msp = adc12_get_temp_sample(clk_set);
324
325
326                 //  IRQ TimerA vortsetzen
327                 if(IRQ_TIMERA_IS_BALANCING) {
328                     TIMER_A_0_CM_IRQ_ENABLE;
329                     TIMER_A_START_UP_MODE;
330                 }
331
332                 rx(HEADER_LENGTH + rx_data_length);
333             } break;
334
335             /************************************************************
336             ** Senden der einfachen Spannungs- und Temp.-Messung
337             ************************************************************/
```

```
            case COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE_ALL: {
339
//              uint8_t packet[10]; // changed (s.o.) 25/09/2014 - EM
341
                packet[0] = ADDRESS_BASE_STATION;
343             packet[1] = ADDRESS_THIS_SENSOR;
                packet[2] = COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE;
345             packet[3] = 0x00;

347             packet[4] = (uint8_t)((sample_buf_volt >> 8) & 0x0F);
                packet[5] = (uint8_t)((sample_buf_volt >> 0) & 0xFF);
349
                packet[6] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
351             packet[7] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);

353             packet[8] = (uint8_t)((sample_buf_temp_msp >> 8) & 0x0F);
                packet[9] = (uint8_t)((sample_buf_temp_msp >> 0) & 0xFF);
355
                tx_packet(10, packet);
357

359             rx(HEADER_LENGTH + rx_data_length);
            } break;
361
            /*************************************************************
363         ** Aufnahme der einfachen Temperaturmessung
            ** mit TMP102
365         *************************************************************/
            case COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_TMP102: {
367
                // IRQ TimerA anhalten
369             // Da Interrupt stört NS 27.06.13
                if(IRQ_TIMERA_IS_BALANCING) {
371                 TIMER_A_STOP;
                    TIMER_A_0_CM_IRQ_DISABLE;
373             }

375             sample_buf_temp = temp_sensor_get_temp();
                sample_buf_temp = sample_buf_temp + cali_pos_offset_tmp102;
377             sample_buf_temp = sample_buf_temp - cali_neg_offset_tmp102;

379             //  IRQ TimerA vortsetzen
                if(IRQ_TIMERA_IS_BALANCING) {
381                 TIMER_A_0_CM_IRQ_ENABLE;
                    TIMER_A_START_UP_MODE;
383             }

385               rx(HEADER_LENGTH + rx_data_length);
            } break;
387
            /*************************************************************
389         ** Senden der einfachen Temperaturmessung
            ** mit TMP102
391         *************************************************************/
            case COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102: {
393
//              uint8_t packet[6];    // changed (s.o.) 25/09/2014 - EM
395
                packet[0] = ADDRESS_BASE_STATION;
397             packet[1] = ADDRESS_THIS_SENSOR;
                packet[2] = COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102;
399             packet[3] = 0x00;

401             packet[4] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
                packet[5] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);
403
                tx_packet(6, packet);
405

                rx(HEADER_LENGTH + rx_data_length);
407         } break;

409         /*************************************************************
            ** Aufnahme der einfachen Temperaturmessung
411         ** mit TMP102 zur Kalibrierung
            *************************************************************/
413         case COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_TMP102_CALI: {

415             // IRQ TimerA anhalten
                // Da Interrupt stört NS 27.06.13
417             if(IRQ_TIMERA_IS_BALANCING) {
                    TIMER_A_STOP;
419                 TIMER_A_0_CM_IRQ_DISABLE;
                }
421
                sample_buf_temp = temp_sensor_get_temp();
```

```
423
                    //  IRQ TimerA vortsetzen
425                 if(IRQ_TIMERA_IS_BALANCING) {
                        TIMER_A_0_CM_IRQ_ENABLE;
427                     TIMER_A_START_UP_MODE;
                    }
429
                    delay_ms(1);
431
                    rx(HEADER_LENGTH + rx_data_length);
433             } break;

435             /*************************************************************
                ** Senden der einfachen Temperaturmessung
437             ** mit TMP102 zur Kalibrierung
                *************************************************************/
439             case COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102_CALI: {

441 //                  uint8_t packet[6];     // changed (s.o.) 25/09/2014 - EM

443                 packet[0] = ADDRESS_BASE_STATION;
                    packet[1] = ADDRESS_THIS_SENSOR;
445                 packet[2] = COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102_CALI;
                    packet[3] = 0x00;
447
                    packet[4] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
449                 packet[5] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);

451                 tx_packet(6, packet);

453                 rx(HEADER_LENGTH + rx_data_length);
                } break;
455
                /*************************************************************
457             ** Aufnahme der einfachen Temperaturmessung
                ** mit MSP430 zur Kalibrierung
459             *************************************************************/
                case COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_MSP430_CALI: {
461
                    sample_buf_temp_msp = adc12_get_temp_sample(clk_set);
463                 delay_ms(1);

465                 rx(HEADER_LENGTH + rx_data_length);
                } break;
467
                /*************************************************************
469             ** Senden der einfachen Temperaturmessung
                ** mit MSP430 zur Kalibrierung
471             *************************************************************/
                case COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430_CALI: {
473
474 //                  uint8_t packet[6];     // changed (s.o.) 25/09/2014 - EM
475
                    packet[0] = ADDRESS_BASE_STATION;
477                 packet[1] = ADDRESS_THIS_SENSOR;
                    packet[2] = COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430_CALI;
479                 packet[3] = 0x00;

481                 packet[4] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
                    packet[5] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);
483
                    tx_packet(6, packet);
485
                    rx(HEADER_LENGTH + rx_data_length);
487             } break;

489             /*************************************************************
                ** Aufnahme der einfachen Temperaturmessung
491             ** mit MSP430
                *************************************************************/
493             case COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_MSP430: {

495                 sample_buf_temp = adc12_get_temp_sample(clk_set);
                    sample_buf_temp = sample_buf_temp + cali_pos_offset_msp430;
497                 sample_buf_temp = sample_buf_temp - cali_neg_offset_msp430;
                    delay_ms(1);
499
                    rx(HEADER_LENGTH + rx_data_length);
501             } break;

503             /*************************************************************
                ** Senden der einfachen Temperaturmessung
505             ** mit MSP430
                *************************************************************/
507             case COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430: {
```

```
509 //                  uint8_t packet[6];// changed (s.o.) 25/09/2014 - EM
511                     packet[0] = ADDRESS_BASE_STATION;
                        packet[1] = ADDRESS_THIS_SENSOR;
513                     packet[2] = COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430;
                        packet[3] = 0x00;
515
                        packet[4] = (uint8_t)((sample_buf_temp >> 8) & 0x0F);
517                     packet[5] = (uint8_t)((sample_buf_temp >> 0) & 0xFF);

519                     tx_packet(6, packet);

521                     rx(HEADER_LENGTH + rx_data_length);
                    } break;
523
                    /*************************************************************
525                 ** Aufnahme der einfachen Spannungsmessung
                    *************************************************************/
527                 case COMMAND_DOWNLINK_SAMPLE_VOLTAGE: {

529                     sample_buf_volt = adc12_get_volt_sample(clk_set);
                        delay_ms(1);
531


533                     rx(HEADER_LENGTH + rx_data_length);
                    } break;
535
                    /*************************************************************
537                 ** Senden der einfachen Spannungsmessung
                    *************************************************************/
539                 case COMMAND_DOWNLINK_SEND_VOLTAGE: {

541
//                  uint8_t packet[6];     // changed (s.o.) 25/09/2014 - EM
543
                        packet[0] = ADDRESS_BASE_STATION;
545                     packet[1] = ADDRESS_THIS_SENSOR;
                        packet[2] = 0x00;//COMMAND_DOWNLINK_SEND_VOLTAGE;
547                     packet[3] = 0x00;

549                     packet[4] = (uint8_t)((sample_buf_volt >> 8) & 0x0F);
                        packet[5] = (uint8_t)((sample_buf_volt >> 0) & 0xFF);
551
                        tx_packet(6, packet);
553
                        rx(HEADER_LENGTH + rx_data_length);
555                 } break;

557                 /*************************************************************
                    ** Kommando zum Balancing empfangen
559                 *************************************************************/
                    case COMMAND_DOWNLINK_BALANCING_ON: {
561                     timer_a_init_balanc();  // TimerA wird initialisiert
                        TIMER_A_START_UP_MODE;
563                     balanc_state = ON;

565                     rx(HEADER_LENGTH + rx_data_length);
                    } break;
567
                    /*************************************************************
569                 ** Kommando zum Balancing ausschalten empfangen
                    *************************************************************/
571                 case COMMAND_DOWNLINK_BALANCING_OFF: {
                        TIMER_A_STOP;
573                     balancing_off();
                        balanc_state = OFF;
575


577                     rx(HEADER_LENGTH + rx_data_length);
                    } break;
579
                    /*************************************************************
581                 ** Kommando zur Burstmessung
                    *************************************************************/
583                 case COMMAND_DOWNLINK_BURST_MODE: {

585                     // DCO auf 16 MHz setzen
                      clk_init_MHz(16);
587                     clk_set = 16;                   //CLK ist auf 16MHz gesetzt
                        //
589
                        adc12_volt_init(clk_set);       // Initialize ADC auf 16 MHz
591                     burst_counter = 0;              // Zurücksetzen
```

```
593                 // Interrupt sperren
                    CC430_BURST_SIG_IRQ_DISABLE;   // changed 25/09/2014 - EM
595                 CC430_BURST_SIG_CLEAR_IRQ;     // changed 25/09/2014 - EM

597

                    // IRQ TimerA anhalten
599                 // Da Interrupt stört NS 27.06.13
                    if(IRQ_TIMERA_IS_BALANCING) {
601                     TIMER_A_STOP;
                        TIMER_A_0_CM_IRQ_DISABLE;
603                 }
                            // changed 25/09/2014 - EM
605                 cc430_init_burst();               // Auf Empfang konfigurieren
                    cc430_reset();                    // cc430 reset
607                 cc430_config_no_packet_rx();      // Auf asynchronen Empfang stellen
                    cc430_burst_rx();                 // Ab hier wird empfangen
609                 IRQ_SET_BURST;                    // IRQ auf Burst Mode stellen
                    delay_ms(32);                     // Wartezeit
611
                    // Interrupt freischalten
613                 CC430_BURST_SIG_CLEAR_IRQ;     // changed 25/09/2014 - EM
                    CC430_BURST_SIG_IRQ_ENABLE;    // changed 25/09/2014 - EM
615                 // Ab hier wird Interruptgesteuert empfangen

617                 command_recived = 0;
                }break;
619
                /************************************************************
621             ** Kommando wenn vorher BURST MODE war, um CLK unzustellen
                ************************************************************/
623             case COMMAND_BACK_FROM_BURST: {

625                 _DINT();                  // Globale Interrups ausschalten
                    clk_init_MHz(1);
627                 _EINT();                  // Globale Interrups einschalten

629                 // IRQ TimerA fortsetzen
                    // Da Interrupt stört NS 27.06.13
631                 if(IRQ_TIMERA_IS_BALANCING) {
                        TIMER_A_START_UP_MODE;
633                     TIMER_A_0_CM_IRQ_ENABLE;
                    }
635
                    rx(HEADER_LENGTH + rx_data_length);
637             } break;

639             /************************************************************
                ** Leersendung
641             ************************************************************/
                case COMMAND_DOWNLINK_BURST_CHECK: {
643
                    rx(HEADER_LENGTH + rx_data_length);
645             } break;


647

                /************************************************************
649             ** Anfrage der Burst Daten
                ************************************************************/
651             case COMMAND_DOWNLINK_BURST_DATA_RQ: {
                    //Berechnung der Anzahl der Frames
653                 uint16_t index = burst_counter;
                    //burst_counter = 0;          // Zurücksetzen
655                 burst_frame_counter = 0;    // Zurücksetzen

657                 while(index >= (burst_frame_lenght/2)) {
                        burst_frame_counter++;
659                     index = index - (burst_frame_lenght/2);
                    }
661                 //if(index != 0)  // Berechnung wenn weniger als (burst_frame_lenght/2) übrig bleibt
                    //    burst_frame_counter++;
663
//                    uint8_t packet[4];        // changed (s.o.) 25/09/2014 - EM
665
                    packet[0] = ADDRESS_BASE_STATION;
667                 packet[1] = ADDRESS_THIS_SENSOR;
                    packet[2] = burst_frame_counter;  // Anzahl der Frames
669                 packet[3] = burst_frame_lenght;   // Länge der Frames ohne Header

671

                    tx_packet(4,packet);
673

675                 rx(HEADER_LENGTH + rx_data_length);
                }break;
677
```

```
            /*************************************************************
            ** Senden der Burst Daten an das Batteriesteuergerät
            *************************************************************/
            case COMMAND_DOWNLINK_BURST_DATA_RX: {

                uint16_t seq_number      = 0;
                uint8_t sample_counter   = 0;
                uint8_t packages_counter = 0;
//                uint8_t packet[64];        // changed (s.o.) 25/09/2014 - EM

                packet[0] = ADDRESS_BASE_STATION;
                packet[1] = ADDRESS_THIS_SENSOR;
                packet[2] = seq_number;             // Anzahl der Frames
                packet[3] = frame_number;           // Länge der Frames ohne Header


                for(packages_counter=0;packages_counter < burst_frame_lenght;packages_counter++) {
                    packet[HEADER_LENGTH + packages_counter] = (uint8_t)((sample_burst_buf[(frame_number*(burst_frame_lenght/2))+
    sample_counter] >> 8) & 0xFF);
                    seq_number++;       // Sequenznummer hochzählen
                    packages_counter++;
                    packet[HEADER_LENGTH + packages_counter] = (uint8_t)((sample_burst_buf[(frame_number*(burst_frame_lenght/2))+
    sample_counter] >> 0) & 0xFF);
                    seq_number++;       // Sequenznummer hochzählen
                    sample_counter++;
                }

                tx_packet((HEADER_LENGTH + burst_frame_lenght),packet);

                rx(HEADER_LENGTH + rx_data_length);
            } break;

            /*************************************************************
            ** Kommando SLEEP empfangen
            *************************************************************/
            case COMMAND_DOWNLINK_SLEEP: {

                led_on(LED_ALL);
                delay_ms(1000);
                led_off(LED_ALL);

                init_for_sleep();

                ENTER_LPM4;

            } break;

            /*************************************************************
            **
            *************************************************************/
            case COMMAND_WAIT: {

                rx(HEADER_LENGTH + rx_data_length);
            } break;
        }
    }
  }
  return 0;
}

/*****************************************************************************************************
**
**
**
*****************************************************************************************************/
void rx(uint8_t packages) {
    cc430_set_rx(brate);      // changed 25/09/2014 - EM
    IRQ_SET_RX;
    CC430_END_OF_PKT_CLEAR_IRQ;      // Clear the sync word detected interrupt    // changed 25/09/2014 - EM
    CC430_END_OF_PKT_IRQ_ENABLE;     // Enable the sync word detected interrupt   // changed 25/09/2014 - EM
    CC430_END_OF_PKT_CLEAR_IRQ;                          // changed 25/09/2014 - EM
    cc430_rx(packages);                // Transceiver in RX state         // changed 25/09/2014 - EM
    command_recived = 0;
}
/*****************************************************************************************************
**
**
**
*****************************************************************************************************/
void tx_carrier(void) {
    led_on(LED_TX);
    adg918_transceiver();
    cc430_init();       // changed 25/09/2014 - EM
    cc430_reset();         // Reset chip and go to idle state
    SetVCore(2);
```

```
761
763     cc430_config_no_packet();      // changed 25/09/2014 - EM
        // Internal timer output (10.5 kHz) to Radio TX
765       // NOTE: SMARTF_CC430 IOCFG0 should = 0x2E. When IOCFG0 = 0x2D,
          // asynchronous data into the radio is taken from GDO0 and not the timer.
767       TA1CCR0 = 50;
          TA1CCR1 = 50;
769
          TA1CCTL0 = OUTMOD_4;
771       TA1CCTL1 = OUTMOD_4;
          TA1CTL = TASSEL__SMCLK + MC_1 + TACLR;
773
          //Transmit the TX waveform asynchronously
775
777     cc430_tx_carrier();      // changed 25/09/2014 - EM

779     while(1);
    }
781 /***********************************************************************************************
    **
783 **
    **
785 ***********************************************************************************************/
    void tx_packet(uint8_t packetes, uint8_t *txbuf) {
787     /************************************
         * Sendezeit optimierung 26.03.13 NS
789      ************************************/
        cc430_set_tx(brate);       // changed 25/09/2014 - EM
791     IRQ_SET_TX;
        CC430_END_OF_PKT_CLEAR_IRQ;       // Clear the sync word detected interrupt     // changed 25/09/2014 - EM
793     CC430_END_OF_PKT_IRQ_ENABLE;      // Enable the sync word detected interrupt

795     if(IRQ_TIMERA_IS_BALANCING) {
            TIMER_A_STOP;
797         TIMER_A_0_CM_IRQ_DISABLE;
        }
799
        led_on(LED_TX);
801     cc430_fill_tx_fifo(txbuf, packetes);       // changed 25/09/2014 - EM
        cc430_tx(packetes);                  // changed 25/09/2014 - EM
803
        while(!irq_send_done_flag);
805
        if(IRQ_TIMERA_IS_BALANCING) {
807         TIMER_A_0_CM_IRQ_ENABLE;
            TIMER_A_START_UP_MODE;
809     }

811     //delay_ms(10);
        led_off(LED_TX);
813
        irq_send_done_flag = 0;
815
        cc430_idle();                     // changed 25/09/2014 - EM
817
    }
```

code/ZSK3V04/main.c

E.2 main.h

```
/*******************************************************************************
**     Description:     main.h
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            06/03/2013
**     Last Update:     08/10/204 - EM
**     Author:          Nico Sassano
*******************************************************************************/

#ifndef MAIN_H_
#define MAIN_H_

#include <cc430f5137.h>
#include <signal.h>
#include <stdio.h>

#include "types.h"
#include "adc12.h"
#include "adg918.h"
#include "as3930.h"
#include "balancing.h"
#include "cc430.h"     // changed 28/08/204 - EM
#include "delay.h"
#include "led.h"
#include "i2c.h"
#include "temp_sensor.h"
#include "timer.h"
#include "clk.h"
#include "init.h"
#include "RF1A.h"
#include "hal_pmm.h"
#include "adc24.h"     // changed 28/08/204 - EM

/*------------------------------------------------------------------------------
  Fix error in msp430x23x.h
------------------------------------------------------------------------------*/

//#define __MSP430_HAS_ADC12__
//#include <msp430/adc12.h>

/*------------------------------------------------------------------------------
  Defines
------------------------------------------------------------------------------*/
#define ENABLE_LEDS

// TPS Controll

#define TPS61201_PxDIR   (P3DIR)
#define TPS61201_PxOUT   (P3OUT)
#define TPS61201_PIN   (BIT1)

#define TPS61201_INIT TPS61201_PxDIR |= TPS61201_PIN ;
#define TPS61201_ENABLE TPS61201_PxOUT |= TPS61201_PIN
#define TPS61201_DISABLE TPS61201_PxOUT &=~TPS61201_PIN

#define ON 0x01
#define OFF 0x00
// Testport deklaration

#define TEST_PORT_PxDIR                                 (P2DIR)
#define TEST_PORT_PxOUT                                 (P2OUT)
#define TEST_PORT_PIN                                   (BIT7)

#define TEST_PIN_ON          TEST_PORT_PxOUT |= TEST_PORT_PIN
#define TEST_PIN_OFF         TEST_PORT_PxOUT &=~TEST_PORT_PIN

#define ENABLE_LEDS

#define DOWNLINK_DATA_BYTES             4//6        // Incl. address byte
#define UPLINK_DATA_BYTES               7//11       // Incl. address byte

// Frequency offset added to the base frequency (in units of 1.59 kHz - 1.65 kHz)
#define FREQUENCY_OFFSET                0 // changed 28/08/204 - EM

// Address definitions
#define BROADCAST                                       0x00
#define ADDRESS_BASE_STATION                            0xFF
#define ADDRESS_THIS_SENSOR                             0x06

/*** Downlink commands ************************************/
#define COMMAND_WAKEUP                                  0x00
#define COMMAND_WAKEUP_DONE                             0x01
#define COMMAND_DOWNLINK_IS_AWAKE                       0x02
```

```
#define COMMAND_DOWNLINK_SAMPLE_VOLTAGE                 0x03
#define COMMAND_DOWNLINK_SEND_VOLTAGE                   0x04
#define COMMAND_DOWNLINK_SLEEP                          0x05
#define COMMAND_DOWNLINK_SAMPLE_TEMPERATURE             0x06
#define COMMAND_DOWNLINK_SEND_TEMPERATURE               0x07
#define COMMAND_DOWNLINK_BALANCING_ON                   0x08
#define COMMAND_DOWNLINK_BALANCING_OFF                  0x09
#define COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE       0x0A
#define COMMAND_DOWNLINK_SAMPLE_VOLTAGE_TEMPERATURE     0x0B
#define COMMAND_DOWNLINK_BURST_MODE                     0x0C
#define COMMAND_DOWNLINK_BURST_DATA_RQ                  0x0D
#define COMMAND_DOWNLINK_BURST_DATA_RX                  0x0E
#define COMMAND_DOWNLINK_BURST_CHECK                    0x0F
#define COMMAND_DOWNLINK_CONFIG_SET                     0x10
#define COMMAND_DOWNLINK_CONFIG                         0x11
#define COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_TMP102      0x12
#define COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102        0x13
#define COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_TMP102_CALI 0x14
#define COMMAND_DOWNLINK_SEND_TEMPERATURE_TMP102_CALI   0x15
#define COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_MSP430      0x16
#define COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430        0x17
#define COMMAND_DOWNLINK_SAMPLE_TEMPERATURE_MSP430_CALI 0x18
#define COMMAND_DOWNLINK_SEND_TEMPERATURE_MSP430_CALI   0x19
#define COMMAND_DOWNLINK_CALIBRATION_TMP102             0x1A
#define COMMAND_DOWNLINK_CALIBRATION_MSP430             0x1B
#define COMMAND_DOWNLINK_CALIBRATION_ADC                0x1C
#define COMMAND_DOWNLINK_SEND_VOLTAGE_TEMPERATURE_ALL   0x1D
#define COMMAND_DOWNLINK_SAMPLE_VOLTAGE_TEMPERATURE_ALL 0x1E


#define COMMAND_BACK_FROM_BURST                         0xFD
#define COMMAND_WAIT                                    0xFE
/*** TEST DOWNLINKS ***************************************/
#define COMMAND_DOWNLINK_ERROR_TEST                     0x99

#define COMMAND_DOWNLINK_UNKWOWN                        0xFF

#define HEADER_LENGTH                                   0x04

/************************************************************
** Frequenzen für die Burstmessung
************************************************************/
#define BURST_FREQ_10000HZ                              0x19
#define BURST_FREQ_8000HZ                               0x18
#define BURST_FREQ_6000HZ                               0x17
#define BURST_FREQ_4000HZ                               0x15
#define BURST_FREQ_2000HZ                               0x16
#define BURST_FREQ_1000HZ                               0x01
#define BURST_FREQ_950HZ                                0x02
#define BURST_FREQ_900HZ                                0x03
#define BURST_FREQ_850HZ                                0x04
#define BURST_FREQ_800HZ                                0x05
#define BURST_FREQ_750HZ                                0x06
#define BURST_FREQ_700HZ                                0x07
#define BURST_FREQ_650HZ                                0x08
#define BURST_FREQ_600HZ                                0x09
#define BURST_FREQ_550HZ                                0x0A
#define BURST_FREQ_500HZ                                0x0B
#define BURST_FREQ_450HZ                                0x0C
#define BURST_FREQ_400HZ                                0x0D
#define BURST_FREQ_350HZ                                0x0E
#define BURST_FREQ_300HZ                                0x0F
#define BURST_FREQ_250HZ                                0x10
#define BURST_FREQ_200HZ                                0x11
#define BURST_FREQ_150HZ                                0x12
#define BURST_FREQ_100HZ                                0x13
#define BURST_FREQ_50HZ                                 0x14


#define BURST_VALUES                                    1900
#define BURST_VALUES_50                                 0x01
#define BURST_VALUES_100                                0x02
#define BURST_VALUES_150                                0x03
#define BURST_VALUES_200                                0x04
#define BURST_VALUES_250                                0x05
#define BURST_VALUES_300                                0x06
#define BURST_VALUES_350                                0x07
#define BURST_VALUES_400                                0x08
#define BURST_VALUES_450                                0x09
#define BURST_VALUES_500                                0x0A
#define BURST_VALUES_550                                0x0B
#define BURST_VALUES_600                                0x0C
#define BURST_VALUES_650                                0x0D
#define BURST_VALUES_700                                0x0E
```

```
168 #define BURST_VALUES_750                        0x0F
    #define BURST_VALUES_800                        0x10
170 #define BURST_VALUES_850                        0x11
    #define BURST_VALUES_900                        0x12
172 #define BURST_VALUES_1000                       0x13   // added 08/10/204 - EM
    #define BURST_VALUES_1500                       0x14   // added 08/10/204 - EM
174 #define BURST_VALUES_1900                       0x15   // added 08/10/204 - EM

176 /*****************************************************
    ** Interrupt Modes
178 *****************************************************/
    #define TX_MODE                                 0x00
180 #define RX_MODE                                 0x01
    #define BURST_MODE                              0x02
182
    #define ALERT_HIGH                              0x00
184 #define ALERT_LOW                               0x01

186 #define TEMP_NORMAL                             0x00
    #define TEMP_HIGH                               0x01
188
    /*****************************************************
190 ** TIMER Macros
    *****************************************************/
192 #define TIMERA_BALANCING                        0x01

194 #define TIMERB_BURST                            0x01
    #define TIMERB_BURST_END                        0x02
196
    // Clock frequencies
198 #define DCOCLK                          1000000  // DCO Clock frequency
    #define MCLK                            1000000  // Main System Clock frequency
200 #define SMCLK                           1000000  // Sub System Clock frequency

202 /*****************************************************
    ** IRQ Macros
204 *****************************************************/
    #define IRQ_SET_RX                  (irq_mode = RX_MODE)
206 #define IRQ_IS_RX                   (irq_mode == RX_MODE)
    #define IRQ_SET_TX                  (irq_mode = TX_MODE)
208 #define IRQ_IS_TX                   (irq_mode == TX_MODE)
    #define IRQ_SET_BURST               (irq_mode = BURST_MODE)
210 #define IRQ_IS_BURST                (irq_mode == BURST_MODE)

212 /*****************************************************
    ** IRQ_ALERT_SET_HIGH    bei zu hoher Temperatur
214 ** IRQ_ALERT_IS_LOW      bei normaler Temperatur
    *****************************************************/
216 #define IRQ_ALERT_SET_HIGH          (irq_alert = ALERT_HIGH)
    #define IRQ_ALERT_IS_HIGH           (irq_alert == ALERT_HIGH)
218 #define IRQ_ALERT_SET_LOW           (irq_alert = ALERT_LOW)
    #define IRQ_ALERT_IS_LOW            (irq_alert == ALERT_LOW)
220
    #define TEMP_IS_NORMAL              (temp_state == TEMP_NORMAL)
222 #define TEMP_SET_NORMAL             (temp_state = TEMP_NORMAL)
    #define TEMP_IS_HIGH                (temp_state == TEMP_HIGH)
224 #define TEMP_SET_HIGH               (temp_state = TEMP_HIGH)

226 /*****************************************************
    ** IRQ TIMER
228 *****************************************************/
    #define IRQ_TIMERA_SET_BALANCING    (irq_timera = TIMERA_BALANCING)
230 #define IRQ_TIMERA_UNSET_BALANCING  (irq_timera = 0x00)
    #define IRQ_TIMERA_IS_BALANCING     (irq_timera == TIMERA_BALANCING)
232
    #define IRQ_TIMERB_SET_BURST        (irq_timerb = TIMERB_BURST)
234 #define IRQ_TIMERB_UNSET_BURST      (irq_timerb = 0x00)
    #define IRQ_TIMERB_IS_BURST         (irq_timerb == TIMERB_BURST)
236
    /*****************************************************
238 ** Burst Flag Macros
    *****************************************************/
240 #define BURST_ERROR_FLAG_SET        (burst_error_flag = 0x01)
    #define BURST_ERROR_FLAG_UNSET      (burst_error_flag = 0x00)
242 #define BURST_ERROR_FLAG_IS_SET     (burst_error_flag == 0x01)
    #define BURST_ERROR_FLAG_IS_UNSET   (burst_error_flag == 0x00)
244
    /*****************************************************
246  ** ADC AUSWAHL
     *****************************************************/
248 #define ADC12_INT_SELECT   0x00     // added 28/08/204 - EM
    #define ADC24_EXT_SELECT   0x01     // added 28/08/204 - EM
250
    /*-------------------------------------------------------------------
252   Types
```

```
254 ------------------------------------------------------------------------------*/

256 typedef enum { S_INIT,
                  S_SLEEP,
                  S_RX,
258               S_WAKEUP,
                  S_WAKEUP_RX,
260               S_WAKEUP_DONE,
                  S_TX_AWAKE,
262               S_SAMPLE,
                  S_TX_SAMPLE,
264               S_BALANC_FIRST_START,
                  S_BALANC_ON,
266               S_BALANC_OFF
                } state_t ;
268

270
    //uint16_t temp_high = 0x01A0; //0x01A0 -> 26°C
272 //uint16_t temp_low = 0x0190; // 0x0190 -> 25°C

274 /*------------------------------------------------------------------------------
      Macros
276 ------------------------------------------------------------------------------*/

278 #define ENTER_LPM4                      __bis_SR_register(LPM4_bits + GIE);
    #define EXIT_LPM4                       __bic_SR_register_on_exit(LPM4_bits);
280
    void rx(uint8_t);
282 void tx_carrier(void);

284 #endif /* MAIN_H_ */
```

code/ZSK3V04/main.h

E.3 adc24.c

```
/*******************************************************************************
**    Description:    adc24.c
**    Hardware:       BATSEN ADC24 Daughterbaord for ZS Klasse 3 v0.4 - Eike Mense - 08/2014
**    Date:           31/08/2014
**    Last Update:    02/09/2014
**    Author:         Eike Mense
*******************************************************************************/

#include "main.h"

extern volatile uint16_t adc24_settling_time;
extern volatile uint8_t  adc24_mode;


void adc24_spi_setup(void)
{
    ADC24_CS_PxDIR |= ADC24_CS_PIN;    // CS is output
    ADC24_CS_DISABLE;  // Chip disable

    UCA0CTL1 |= UCSWRST;                // Hold state machine in reset

    UCA0CTL0 &= ~UCCKPH;                    // Data is changed on rising edge
    UCA0CTL0 &= ~UCCKPL;                    // Inactive clock is low
    UCA0CTL0 |= (UCMST | UCMSB | UCSYNC);   // MSB first, Master mode, Synchronous mode
    UCA0CTL0 &= ~(UCMODE1 | UCMODE0);       // 3-pin SPI
    UCA0MCTL = 0;                           // No modulation

    // SMCLK / 2
    UCA0CTL1 |= (UCSSEL1 | UCSSEL0);
    UCA0BR0 = 2;
    UCA0BR1 = 0;

    // SPI functionality for pins
    ADC24_SPI_PxSEL |= (ADC24_SPI_MOSI_PIN | ADC24_SPI_MISO_PIN | ADC24_SPI_CLK_PIN);
    ADC24_SPI_PxDIR |= (ADC24_SPI_MOSI_PIN | ADC24_SPI_CLK_PIN); // MOSI and CLK are outputs
    ADC24_SPI_PxDIR &= ~ADC24_SPI_MISO_PIN; // MISO is input

    UCA0CTL1 &= ~UCSWRST;   // Initialize USART state machine
}

void ADC24_spi_write_register(u8_t address, u8_t value)
{
  ADC24_CS_ENABLE;            // Chip enable
    while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
    UCA0TXBUF = address | (ADC24_WREG);  // Send configuration mode and register
    while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
    UCA0TXBUF = 0x00;                          // Write Number of Bytes -1 to write
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    UCA0TXBUF = value;
    while (UCA0STAT & UCBUSY);
    ADC24_CS_DISABLE;           // Chip disable
}

char ADC24_spi_read_register(u8_t address)
{
    u8_t value;

    ADC24_CS_ENABLE;            // Chip enable
    while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
    UCA0TXBUF = address | (ADC24_RREG );   // Send configuration mode and register
    while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
    UCA0TXBUF = 0x00;                           // Write Number of Bytes -1 to read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    value = UCA0RXBUF;                          // Read data
    ADC24_CS_DISABLE;           // Chip disable

    return value;
}

uint32_t ADC24_spi_read_data()
{
    u8_t value1, value2, value3;
   volatile uint32_t buf;


    ADC24_CS_ENABLE;                    // Chip enable
    while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
    UCA0TXBUF = ADC24_RDATA;            // Send Read Data Request
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    UCA0TXBUF = 0x00;                           // Dummy write for Read
```

```
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
                            //    STATUS1
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
                            // Read STATUS2
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
                            // Read STATUS3
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    value1 = UCA0RXBUF;                          // Read CH1_1
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    value2 = UCA0RXBUF;                          // Read CH1_2
    UCA0TXBUF = 0x00;                           // Dummy write for Read
    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
    value3 = UCA0RXBUF;                          // Read CH1_3
//    UCA0TXBUF = 0x00;                           // Dummy write for Read
//    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
//    dummy = UCA0RXBUF;                          // Read CH2_1
//    UCA0TXBUF = 0x00;                           // Dummy write for Read
//    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
//    dummy = UCA0RXBUF;                          // Read Ch2_2
//    UCA0TXBUF = 0x00;                           // Dummy write for Read
//    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
//    dummy = UCA0RXBUF;                          // Read CH2_3
//    UCA0TXBUF = 0x00;                           // Dummy write for Read
//    while (UCA0STAT & UCBUSY);                  // Wait for TX complete
//    dummy = UCA0RXBUF;                          // Check for more

    ADC24_CS_DISABLE;           // Chip disable

    buf = (((uint32_t)(value1))<<16) | (((uint32_t)(value2))<<8) | (((uint32_t)(value3)));

    return buf;
}

void ADC24_spi_command(u8_t command)
{
  ADC24_CS_ENABLE;                // Chip enable
    while (UCA0STAT & UCBUSY);                      // Wait for TX to finish
    UCA0TXBUF = command;     // Send configuration mode and register
    while (UCA0STAT & UCBUSY);                      // Wait for TX to finish
    ADC24_CS_DISABLE;               // Chip disable
}

void adc24_volt_cont_init(uint8_t sps){
  adc24_settling_time = 0;
  adc24_mode = ADC24_CONT_MODE;
  adc24_spi_setup();

  ADC24_spi_command(ADC24_SDATAC);
  ADC24_spi_write_register(ADC24_CONFIG1 ,  (ADC24_CONTINOUS | sps));
  ADC24_spi_write_register(ADC24_CONFIG2 ,  (ADC24_UNLOCK_CONF_2 | ADC24_PDB_REFBUF));
  ADC24_spi_write_register(ADC24_CH1SET , ADC24_PGA_GAIN_1 |  ADC24_INPUT_DIFF);
  ADC24_spi_write_register(ADC24_CH2SET , ADC24_POWER_DOWN_CH);
  ADC24_spi_write_register(ADC24_GPIO , 0x00);
  ADC24_spi_command(ADC24_RDATAC);  // ~3.5us

}

void adc24_volt_init(uint8_t sps)
{
  switch (sps)
  {
  case ADC24_125_SPS :
    adc24_settling_time = ADC24_SET_TIME_125_SPS;
    break;
  case ADC24_250_SPS :
    adc24_settling_time = ADC24_SET_TIME_250_SPS;
    break;
  case ADC24_500_SPS :
    adc24_settling_time = ADC24_SET_TIME_500_SPS;
    break;
  case ADC24_1000_SPS :
    adc24_settling_time = ADC24_SET_TIME_1000_SPS;
    break;
  case ADC24_2000_SPS :
    adc24_settling_time = ADC24_SET_TIME_2000_SPS;
    break;
  case ADC24_4000_SPS :
    adc24_settling_time = ADC24_SET_TIME_4000_SPS;
    break;
  case ADC24_8000_SPS :
    adc24_settling_time = ADC24_SET_TIME_8000_SPS;
```

```
168       break;
    default :
170       adc24_settling_time = ADC24_SET_TIME_125_SPS;
          break;
172     }

174     adc24_spi_setup();
    adc24_mode = ADC24_SINGLE_MODE;
176
    ADC24_spi_command(ADC24_SDATAC);
178     ADC24_spi_write_register(ADC24_CONFIG1 ,   (ADC24_SINGLE_SHOT | sps));
    ADC24_spi_write_register(ADC24_CONFIG2 ,   (ADC24_UNLOCK_CONF_2 | ADC24_PDB_REFBUF));
180     ADC24_spi_write_register(ADC24_CH1SET , ADC24_PGA_GAIN_1 |   ADC24_INPUT_DIFF);
    ADC24_spi_write_register(ADC24_CH2SET , ADC24_POWER_DOWN_CH);
182     ADC24_spi_write_register(ADC24_GPIO , 0x00);
    ADC24_spi_command(ADC24_START); // ~3.5us
184 }

186 uint32_t adc24_volt_init_offset(uint8_t sps)
{ volatile uint32_t adc24_volt_buf;
188   switch (sps)
    {
190   case ADC24_125_SPS :
        adc24_settling_time = ADC24_SET_TIME_125_SPS;
192     break;
      case ADC24_250_SPS :
194     adc24_settling_time = ADC24_SET_TIME_250_SPS;
        break;
196   case ADC24_500_SPS :
        adc24_settling_time = ADC24_SET_TIME_500_SPS;
198     break;
      case ADC24_1000_SPS :
200     adc24_settling_time = ADC24_SET_TIME_1000_SPS;
        break;
202   case ADC24_2000_SPS :
        adc24_settling_time = ADC24_SET_TIME_2000_SPS;
204     break;
      case ADC24_4000_SPS :
206     adc24_settling_time = ADC24_SET_TIME_4000_SPS;
        break;
208   case ADC24_8000_SPS :
        adc24_settling_time = ADC24_SET_TIME_8000_SPS;
210     break;
    default :
212     adc24_settling_time = ADC24_SET_TIME_125_SPS;
        break;
214   }
    adc24_spi_setup();
216   ADC24_spi_command(ADC24_SDATAC);
    ADC24_spi_write_register(ADC24_CONFIG1 ,   (ADC24_SINGLE_SHOT | sps));
218   ADC24_spi_write_register(ADC24_CONFIG2 ,   (ADC24_UNLOCK_CONF_2 | ADC24_PDB_REFBUF));
    ADC24_spi_write_register(ADC24_CH1SET , ADC24_PGA_GAIN_1 |   ADC24_INPUT_DIFF);
220   ADC24_spi_write_register(ADC24_CH2SET , ADC24_POWER_DOWN_CH);
    ADC24_spi_write_register(ADC24_GPIO , 0x00);
222   ADC24_spi_command(ADC24_START);
    delay_10us_16MHz(adc24_settling_time*10);
224   //delay_ms(35);
    adc24_volt_buf = ADC24_spi_read_data();
226   adc24_volt_buf >>= 5;
    adc24_volt_buf -= 0xFFF0;
228   return adc24_volt_buf;
}
230
uint16_t adc24_get_volt_sample_off(uint32_t offset){
232   uint32_t adc24_volt_buf;
    ADC24_spi_command(ADC24_START);
234   delay_10us_16MHz(adc24_settling_time);
      adc24_volt_buf =(uint16_t) ((ADC24_spi_read_data() >> 5) - offset);
236   return adc24_volt_buf;
}
238
uint16_t adc24_get_volt_sample(){
240   if(adc24_mode == ADC24_SINGLE_MODE) {
        ADC24_spi_command(ADC24_START); // ~3.5us
242   }
    delay_10us_16MHz(adc24_settling_time);
244     return  (uint16_t) ((ADC24_spi_read_data() >> 5) - 0x22615);  //~ 20us
}
246
uint32_t adc24_get_volt_full(){
248   uint32_t adc24_volt_buf;
    //TPS61201_DISABLE;
250   ADC24_spi_command(ADC24_START);

252   delay_10us_16MHz(adc24_settling_time);
```

```
    //TPS61201_ENABLE;
254   adc24_volt_buf = ADC24_spi_read_data();
      return  adc24_volt_buf;
256
258 }
```

code/ZSK3V04/adc24.c

E.4 adc24.h

```
/*******************************************************************************
**    Description:    adc12.h
**    Hardware:       BATSEN ADC24 Daughterbaord for ZS Klasse 3 v0.4 - Eike Mense - 08/2014
**    Date:           31/08/2014
**    Last Update:    02/09/2014
**    Author:         Eike Mense
*******************************************************************************/


#ifndef ADC24_H_
#define ADC24_H_

#include "main.h"

#define ADC24_CS_PxDIR              P5DIR
#define ADC24_CS_PxOUT              P5OUT
#define ADC24_CS_PIN                BIT0

#define ADC24_SPI_PxSEL             P1SEL
#define ADC24_SPI_PxDIR             P1DIR
#define ADC24_SPI_PxIN              P1IN
#define ADC24_SPI_MOSI_PIN          BIT6
#define ADC24_SPI_MISO_PIN          BIT5
#define ADC24_SPI_CLK_PIN           BIT7

#define ADC24_CS_ENABLE         ADC24_CS_PxOUT &= ~ADC24_CS_PIN
#define ADC24_CS_DISABLE        ADC24_CS_PxOUT |= ADC24_CS_PIN
/*  COMMANDS  */

/*  SYSTEM COMMANDS */

#define ADC24_WAKEUP        0x02
#define ADC24_STANDBY       0x04
#define ADC24_RESET         0x06
#define ADC24_START         0x08
#define ADC24_STOP          0x0A
#define ADC24_OFFSETCAL       0x1A

/*  DATA READ COMMANDS  */

#define ADC24_RDATAC        0x10
#define ADC24_SDATAC        0x11
#define ADC24_RDATA         0x12

/*  REGISTER READ COMMANDS  */

#define ADC24_RREG          0x20
#define ADC24_WREG          0x40

/*  REGISTERS */

/*  DEVICE SETTINGS (READ-ONLY) */

#define ADC24_ID        0x00

/*  GLOBAL SETTINGS ACROSS CHANNEL  */

#define ADC24_CONFIG1       0x01
#define ADC24_CONFIG2       0x02
#define ADC24_LOFF          0x03

/*  CHANNEL SPECIFIG SETTINGS */

#define ADC24_CH1SET        0x04
#define ADC24_CH2SET        0x05
#define ADC24_RLD_SENS      0x06
#define ADC24_LOFF_SENS     0x07
#define ADC24_LOFF_STAT     0x8

/*  GPIO and Other Registers  */

#define ADC24_RESP1         0x09
#define ADC24_RESP2         0x0A
#define ADC24_GPIO          0x0B

/*  SETTINGS  */

#define ADC24_SINGLE_MODE   0x00
#define ADC24_CONT_MODE     0x01

/*  SAMPLING CONFIGURATION  */
```

```
83  #define ADC24_SINGLE_SHOT     0x80
    #define ADC24_CONTINOUS       0x00
85  #define ADC24_125_SPS         0x00
    #define ADC24_250_SPS         0x01
87  #define ADC24_500_SPS         0x02
    #define ADC24_1000_SPS        0x03
89  #define ADC24_2000_SPS        0x04
    #define ADC24_4000_SPS        0x05
91  #define ADC24_8000_SPS        0x06

93  /*  ADC CONFIGURATION */

95  #define ADC24_UNLOCK_CONF_2   0x80
    #define ADC24_PDB_LOFF_COMP   0x40  // ENABLES LEAD_OFF-Detection Komparators
97  #define ADC24_PDB_REFBUF      0x20  // Enables internal Reference Buffer for use of Internal Reference
    #define ADC24_VREF_4V         0x10  // Bit Chooses Between 2.42V Reference (UNSET) and 4.033V Reference (SET)
99  #define ADC24_CLK_EN          0x08  // If set the Internal Oszilator is handed to the CLK_Out pin
    #define ADC24_INT_TEST        0x02  // Enables Internal Test
101 #define ADC24_TEST_FREQ       0x01  // If set Internal Test Signal ist 1Hz Squarewave, if unset Testsignal is DC

103 /*  CHANNEL CONFIGURATION */

105 #define ADC24_POWER_DOWN_CH   0x80
    #define ADC24_PGA_GAIN_6      0x00
107 #define ADC24_PGA_GAIN_1      0x10
    #define ADC24_PGA_GAIN_2      0x20
109 #define ADC24_PGA_GAIN_3      0x30
    #define ADC24_PGA_GAIN_4      0x40
111 #define ADC24_PGA_GAIN_8      0x50
    #define ADC24_PGA_GAIN_12     0x60
113
    #define ADC24_INPUT_DIFF      0x00
115 #define ADC24_INPUT_SHORT     0x01
    #define ADC24_INPUT_RLD_MEAS  0x02
117 #define ADC24_INPUT_MVDD      0x03
    #define ADC24_INPUT_TEMP      0x04
119 #define ADC24_INPUT_TEST      0x05
    #define ADC24_INPUT_RLD_DRP   0x06
121 #define ADC24_INPUT_RLD_DRM   0x07
    #define ADC24_INPUT_RLD_DRPM  0x08
123 #define ADC24_INPUT_CH3       0x09

125 /*  GPIO CONFIGURATION  */

127 #define ADC24_GPIO1_CTL   0x04  // sets GPIO_Pin as Input
    #define ADC24_GPIO2_CTL   0x08  // sets GPIO_Pin as Input
129 #define ADC24_GPIO1_DATA  0x01
    #define ADC24_GPIO2_DATA  0x02
131

133 /*  SETTLING TIME [100 us]  */

135 #define ADC24_SET_TIME_125_SPS       3559
    #define ADC24_SET_TIME_250_SPS       1781
137 #define ADC24_SET_TIME_500_SPS       892
    #define ADC24_SET_TIME_1000_SPS      448
139 #define ADC24_SET_TIME_2000_SPS      226
    #define ADC24_SET_TIME_4000_SPS      115
141 #define ADC24_SET_TIME_8000_SPS      59

143 #define ADC24_ISR_TICKS_125_SPS      0
    #define ADC24_ISR_TICKS_250_SPS      35770
145 #define ADC24_ISR_TICKS_500_SPS      0
    #define ADC24_ISR_TICKS_1000_DIV2_SPS 36424
147 #define ADC24_ISR_TICKS_1000_DIV4_SPS 18212
    #define ADC24_ISR_TICKS_2000_DIV1_SPS 37296
149 #define ADC24_ISR_TICKS_2000_DIV2_SPS 18648
    #define ADC24_ISR_TICKS_4000_SPS     19520
151 #define ADC24_ISR_TICKS_8000_SPS     10624

153 void adc24_spi_setup(void);
    void ADC24_spi_write_register(u8_t, u8_t);
155 char ADC24_spi_read_register(u8_t);
    void ADC24_spi_command(u8_t);
157 uint32_t ADC24_spi_read_data();
    void adc24_volt_init(uint8_t);
159 void adc24_volt_cont_init(uint8_t);
    uint32_t adc24_volt_init_offset(uint8_t);
161 uint16_t adc24_get_volt_sample();
    uint16_t adc24_get_volt_sample_off(uint32_t);
163 uint32_t adc24_get_volt_full();

165 #endif /* ADC24_H_ */
```

code/ZSK3V04/adc24.h

E.5 adc12.c

```
/*******************************************************************************
**     Description:     adc12.c
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 01/07/2014 - EM
**     Date:            06/03/2013
**     Last Update:     18/07/2013
**     Author:          Nico Sassano
*******************************************************************************/

#include "main.h"

/*************************************************************
** Private Funktionen
*************************************************************/
//void adc12_init(void);

//void adc12_temp_init(uint8_t);
//void adc12_disable(void);

/*************************************************************
**  Public functions
*************************************************************/

uint16_t adc12_get_volt_sample(uint8_t clk_set) {
    uint16_t adc_value = 0;
   // TEST_PIN_OFF;
    //TPS61201_DISABLE;               // TPS Enable off

    if(clk_set == 16) {

//        delay_10us_16MHz(6);
//
//
//        TPS61201_ENABLE;     // TPS Enable on

        //TEST_PIN_ON;
        ADC12CTL0 |= (ENC | ADC12SC);               // Start conversion with sampling
        while ((ADC12IFG & 0x0001) != 0x0001) ;     // Wait while conversion is active

    } else if(clk_set == 1){
     //    TEST_PIN_OFF;

        ADC12CTL0 |= (ENC | ADC12SC);               // Start conversion with sampling
        while ((ADC12IFG & 0x0001) != 0x0001) ;     // Wait while conversion is active
        TPS61201_ENABLE;     // TPS Enable on
    }



    adc_value = (ADC12MEM0 & 0x0FFF);
  //  TEST_PIN_ON;

    //adc12_disable();     // ADC ausschalten

    return adc_value;
}

uint16_t adc12_get_temp_sample(uint8_t clk_set) {
    uint16_t temp_value = 0;

    adc12_temp_init(clk_set);

    ADC12CTL0 |= (ENC | ADC12SC);           // Start conversion with sampling
    while ((ADC12IFG & 0x0001) != 0x0001) ; // Wait while conversion is active
    temp_value = (ADC12MEM0 & 0x0FFF);

    //adc12_disable();     // Disable ADC again for saving energy
    adc12_volt_init(clk_set);

    return temp_value;
}

/*-------------------------------------------------------------------------
  Private functions
-------------------------------------------------------------------------*/
//void adc12_init(void) {
//    ADC12_VBAT_PxDIR &= ~ADC12_VBAT_PIN;     // ADC pin is input
//    ADC12_VBAT_PxSEL |= ADC12_VBAT_PIN;      // ADC functionality for pin
//
//    //ADC12CTL0 |= SHT02;      // S&H 64 ADC12CLK cyles (64 us > 37.56 us)
//    ADC12CTL0 |= (SHT02 | SHT01);    //20.03.13 NS
//    ADC12CTL0 |= REF2_5V;    // 2.5 V reference voltage
//    ADC12CTL0 |= REFON;      // Enable reference voltage
```

```
//    ADC12CTL0 |= ADC12ON;    // Enable ADC12
//
//    // Save conversion result to ADC12MEM0
//    ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD1 | CSTARTADD0);
//
//    // ADC12CLK = SMCLK / 1
//    ADC12CTL1 &= ~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0);
//    ADC12CTL1 |= (ADC12SSEL1 | ADC12SSEL1);
//
//    ADC12CTL1 &= ~(CONSEQ1 | CONSEQ0);  // Single-channel, single-conversion
//    ADC12CTL1 |= SHP;        // SAMPCON sourced from sampling timer
//
//    // Input channel A0, VR+ = VREF+, VR- = AVss
//    ADC12MCTL0 = SREF_1;
//}

void adc12_volt_init(uint8_t clk_set)
{
    ADC12CTL0 = 0;  // reset
    ADC12CTL1 = 0;  // reset

    ADC12_VBAT_PxDIR &= ~ADC12_VBAT_PIN;     // ADC pin is input
    ADC12_VBAT_PxSEL |= ADC12_VBAT_PIN;      // ADC functionality for pin

    /*****************************************************************************
    ** Sampel and Hold muss bei jeder CLK Änderung neu eingestellt werden.
    ** Die Hold-Zeit muss größer 37.56 us sein
    *****************************************************************************/
    switch(clk_set) {
        case 1: {
            // ADC12CLK = SMCLK / 1
            ADC12CTL1 &= ~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0);
            ADC12CTL1 |= (ADC12SSEL1 | ADC12SSEL1);

            ADC12_SHT0_CLK_64;       //(64 * (1/1MHz)) = 64sus (64 us > 37.56 us)
        }break;

        case 8: {
            // ADC12CLK = SMCLK / 1
            ADC12CTL1 &= ~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0);
            ADC12CTL1 |= (ADC12SSEL1 | ADC12SSEL1);

            ADC12_SHT0_CLK_512;      //(512 * (1/8MHz)) = 64us (64 us > 37.56 us)
        }break;

        case 16: {
            // ADC12CLK = SMCLK / 4
            ADC12CTL1 &= ~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0);
            ADC12CTL1 |=  (ADC12DIV1 | ADC12DIV0);
            ADC12CTL1 |=  (ADC12SSEL1 | ADC12SSEL1);

            ADC12_SHT0_CLK_256;     //(256 * (4/16MHz)) = 64 us (64 us > 37.56 us)

            // ADC12_SHT0_CLK_192;     //(192 * (4/16MHz)) = 48 us (48 us > 37.56 us)

        }break;

        default: break;
    }

    ADC12CTL0 |= REF2_5V;    // Interner 2.5V Referenzgenerator nutzen
    ADC12CTL0 |= REFON;      // Enable reference voltage
    ADC12CTL0 |= ADC12ON;    // Enable ADC12

    // Save conversion result to ADC12MEM0
    ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD1 | CSTARTADD0);

    ADC12CTL1 &= ~(CONSEQ1 | CONSEQ0);  // Single-channel, single-conversion
    ADC12CTL1 |= SHP;        // SAMPCON sourced from sampling timer

    // Input channel A0, VR+ = VREF+, VR- = AVss
    ADC12MCTL0 = SREF_1;
    ADC12_SET_INPUT_CH0;

}

void adc12_temp_init(uint8_t clk_set)
{
    ADC12CTL0 = 0;  // reset
    ADC12CTL1 = 0;  // reset

    /*****************************************************************************
    ** Sampel and Hold muss bei jeder CLK änderung neu eingestellt werden.
    ** Die Hold-Zeit muss größer 37.56 us sein
    *****************************************************************************/
```

```
//    switch(clk_set) {
//        case 1: {
//            ADC12_SHT0_CLK_128;  //(128 * (1/1MHz)) = 128us (128 us > 37.56 us)
//        }break;
//
//        case 8: {
//            ADC12_SHT0_CLK_512;  //(512 * (1/8MHz)) = 64us (64 us > 37.56 us)
//        }break;
//
//        default: break;
//    }

//    ADC12_REF_SET_2_5V; // Referenz auf 2.5V setzen
//    ADC12_SET_REF_ON;   // Referenz anschalten
//    ADC12_SET_ON;       // ADC anschalten

//    ADC12SSEL_SET_SCLK; // Taktquelle auf SCLK setzen
//    ADC12DIV_SET_1;     // Taktteiler 1
//    ADC12_SET_CONSEQ0;  // Mode Single-channel, single-conversion
//    ADC12_SET_SHP1;     // Takt von sourced from the sampling timer

    ADC12CTL0 = SHT0_4 + REFON + REF2_5V + ADC12ON;     // Internal ref = 1.5V
    ADC12CTL1 |= SHP;                           // enable sample timer
    ADC12MCTL0 = SREF_1 + INCH_10;             // ADC i/p ch A10 = temp sense i/p

    ADC12CTL0 |= ENC;
}

void adc12_disable(void)
{
    ADC12CTL0 &= ~ENC;        // Disable conversion
    ADC12CTL0 &= ~REFON;      // Disable reference voltage
    ADC12CTL0 &= ~ADC12ON;    // Disable ADC12
}
```

code/ZSK3V04/adc12.c

E.6 adc12.h

```
/*******************************************************************************
**     Description:     adc12.h
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 01/07/2014 - EM
**     Date:            06/03/2013
**     Last Update:     04/08/2014 - EM
**     Author:          Nico Sassano
*******************************************************************************/

#ifndef ADC12_H_
#define ADC12_H_

#include "main.h"

#define ADC12_VBAT_PxSEL            P2SEL // changed 04/08/2014 - EM
#define ADC12_VBAT_PxDIR            P2DIR // changed 04/08/2014 - EM
#define ADC12_VBAT_PIN              BIT1  // changed 04/08/2014 - EM

#define ADC12_DIR_IN                (ADC12_VBAT_PxDIR &= ~ADC12_VBAT_PIN)
#define ADC12_SET_PIN               (ADC12_VBAT_PxSEL |= ADC12_VBAT_PIN)


/*************************************************************
**  Bending Defines for CC430 use EM  added 4/08/2014 - EM
*************************************************************/

#define CONSEQ0 ADC12CONSEQ0
#define CONSEQ1 ADC12CONSEQ1
#define CSTARTADD0 ADC12CSTARTADD0
#define CSTARTADD1 ADC12CSTARTADD1
#define CSTARTADD2 ADC12CSTARTADD2
#define CSTARTADD3 ADC12CSTARTADD3
#define ENC ADC12ENC
#define REF2_5V ADC12REF2_5V
#define SHP ADC12SHP
#define INCH_0 ADC12INCH_0
#define INCH_10 ADC12INCH_10
#define SHT00 ADC12SHT00
#define SHT01 ADC12SHT01
#define SHT02 ADC12SHT02
#define SHT03 ADC12SHT03
#define SHT0_4 ADC12SHT0_4
#define SREF_1 ADC12SREF_1



/*************************************************************
** Sample-and-hold time for registers ADC12MEM8 to ADC12MEM15
*************************************************************/
#define ADC12_SHT1_CLK_4            (ADC12CTL0 &=~(SHT03 | SHT02 | SHT01 | SHT00))
#define ADC12_SHT1_CLK_8            (ADC12CTL0 &=~(SHT03 | SHT02 | SHT01)); \
                                    (ADC12CTL0 |= (SHT00))
#define ADC12_SHT1_CLK_16           (ADC12CTL0 &=~(SHT03 | SHT02 | SHT00)); \
                                    (ADC12CTL0 |= (SHT01))
#define ADC12_SHT1_CLK_32           (ADC12CTL0 &=~(SHT03 | SHT02)); \
                                    (ADC12CTL0 |= (SHT01 | SHT00))
#define ADC12_SHT1_CLK_64           (ADC12CTL0 &=~(SHT03 | SHT01 | SHT00)); \
                                    (ADC12CTL0 |= (SHT02))
#define ADC12_SHT1_CLK_96           (ADC12CTL0 &=~(SHT03 | SHT01)); \
                                    (ADC12CTL0 |= (SHT02 | SHT00))
#define ADC12_SHT1_CLK_128          (ADC12CTL0 &=~(SHT03 | SHT00)); \
                                    (ADC12CTL0 |= (SHT02 | SHT01))
#define ADC12_SHT1_CLK_192          (ADC12CTL0 &=~(SHT03)); \
                                    (ADC12CTL0 |= (SHT02 | SHT01 | SHT00))
#define ADC12_SHT1_CLK_256          (ADC12CTL0 &=~(SHT02 | SHT01 | SHT00)); \
                                    (ADC12CTL0 |= (SHT03))
#define ADC12_SHT1_CLK_384          (ADC12CTL0 &=~(SHT02 | SHT01)); \
                                    (ADC12CTL0 |= (SHT03 | SHT00))
#define ADC12_SHT1_CLK_512          (ADC12CTL0 &=~(SHT02 | SHT00)); \
                                    (ADC12CTL0 |= (SHT03 | SHT01))
#define ADC12_SHT1_CLK_768          (ADC12CTL0 &=~(SHT02)); \
                                    (ADC12CTL0 |= (SHT03 | SHT01 |SHT00))
#define ADC12_SHT1_CLK_1024         (ADC12CTL0 &=~(SHT01 | SHT00)); \
                                    (ADC12CTL0 |= (SHT03 | SHT02))

/*************************************************************
** Sample-and-hold time for registers ADC12MEM0 to ADC12MEM7
*************************************************************/
#define ADC12_SHT0_CLK_4            (ADC12CTL0 &=~(SHT03 | SHT02 | SHT01 | SHT00))
#define ADC12_SHT0_CLK_8            (ADC12CTL0 &=~(SHT03 | SHT02 | SHT01)); \
                                    (ADC12CTL0 |= (SHT00))
#define ADC12_SHT0_CLK_16           (ADC12CTL0 &=~(SHT03 | SHT02 | SHT00)); \
                                    (ADC12CTL0 |= (SHT01))
```

```
83  #define ADC12_SHT0_CLK_32           (ADC12CTL0 &=~(SHT03 | SHT02)); \
                                        (ADC12CTL0 |= (SHT01 | SHT00))
85  #define ADC12_SHT0_CLK_64           (ADC12CTL0 &=~(SHT03 | SHT01 | SHT00)); \
                                        (ADC12CTL0 |= (SHT02))
87  #define ADC12_SHT0_CLK_96           (ADC12CTL0 &=~(SHT03 | SHT01)); \
                                        (ADC12CTL0 |= (SHT02 | SHT00))
89  #define ADC12_SHT0_CLK_128          (ADC12CTL0 &=~(SHT03 | SHT00)); \
                                        (ADC12CTL0 |= (SHT02 | SHT01))
91  #define ADC12_SHT0_CLK_192          (ADC12CTL0 &=~(SHT03)); \
                                        (ADC12CTL0 |= (SHT02 | SHT01 | SHT00))
93  #define ADC12_SHT0_CLK_256          (ADC12CTL0 &=~(SHT02 | SHT01 | SHT00)); \
                                        (ADC12CTL0 |= (SHT03))
95  #define ADC12_SHT0_CLK_384          (ADC12CTL0 &=~(SHT02 | SHT01)); \
                                        (ADC12CTL0 |= (SHT03 | SHT00))
97  #define ADC12_SHT0_CLK_512          (ADC12CTL0 &=~(SHT02 | SHT00)); \
                                        (ADC12CTL0 |= (SHT03 | SHT01))
99  #define ADC12_SHT0_CLK_768          (ADC12CTL0 &=~(SHT02)); \
                                        (ADC12CTL0 |= (SHT03 | SHT01 |SHT00))
101 #define ADC12_SHT0_CLK_1024         (ADC12CTL0 &=~(SHT01 | SHT00)); \
                                        (ADC12CTL0 |= (SHT03 | SHT02))
103
105 /*************************************************************************
    ** Reference generator voltage
    *************************************************************************/
107 #define ADC12_REF_SET_1_5V          (ADC12CTL0 &=~REF2_5V)
    #define ADC12_REF_SET_2_5V          (ADC12CTL0 |= REF2_5V)
109
111 /*************************************************************************
    ** Reference generator on/off
    *************************************************************************/
113 #define ADC12_SET_REF_OFF           (ADC12CTL0 &=~REFON)
    #define ADC12_SET_REF_ON            (ADC12CTL0 |= REFON)
115
117 /*************************************************************************
    ** ADC12 on/off
    *************************************************************************/
119 #define ADC12_SET_OFF               (ADC12CTL0 &=~ADC12ON)
    #define ADC12_SET_ON                (ADC12CTL0 |= ADC12ON)
121
123 /*************************************************************************
    ** ADC12 clock divider
    *************************************************************************/
125 #define ADC12DIV_SET_1              (ADC12CTL1 &=~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0))
    #define ADC12DIV_SET_2              (ADC12CTL1 &=~(ADC12DIV2 | ADC12DIV1 )); \
127                                   (ADC12CTL1 |= (ADC12DIV0))
    #define ADC12DIV_SET_3              (ADC12CTL1 &=~(ADC12DIV2 | ADC12DIV0 )); \
129                                     (ADC12CTL1 |= (ADC12DIV1))
    #define ADC12DIV_SET_4              (ADC12CTL1 &=~(ADC12DIV2)); \
131                                     (ADC12CTL1 |= (ADC12DIV1 | ADC12DIV0))
    #define ADC12DIV_SET_5              (ADC12CTL1 &=~(ADC12DIV1 | ADC12DIV0 )); \
133                                     (ADC12CTL1 |= (ADC12DIV2))
    #define ADC12DIV_SET_6              (ADC12CTL1 &=~(ADC12DIV1)); \
135                                     (ADC12CTL1 |= (ADC12DIV2 | ADC12DIV0))
    #define ADC12DIV_SET_7              (ADC12CTL1 &=~(ADC12DIV0)); \
137                                     (ADC12CTL1 |= (ADC12DIV2 | ADC12DIV1))
    #define ADC12DIV_SET_8              (ADC12CTL1 |= (ADC12DIV2 | ADC12DIV1 | ADC12DIV0))
139
141 /*************************************************************************
    ** ADC12 clock source select
    *************************************************************************/
143 #define ADC12SSEL_SET_ADC12OSC      (ADC12CTL1 &=~(ADC12SSEL1 | ADC12SSEL0))
    #define ADC12SSEL_SET_ACLK          (ADC12CTL1 &=~(ADC12SSEL1)); \
145                                   (ADC12CTL1 |= (ADC12SSEL0))
    #define ADC12SSEL_SET_MCLK          (ADC12CTL1 &=~(ADC12SSEL0)); \
147                                     (ADC12CTL1 |= (ADC12SSEL1))
    #define ADC12SSEL_SET_SCLK          (ADC12CTL1 |= (ADC12SSEL1 | ADC12SSEL0))
149
151 /*************************************************************************
    ** ADC12 Conversion start address
    *************************************************************************/
153 #define ADC12MEM_SET_0              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD1 | CSTARTADD0))
    #define ADC12MEM_SET_1              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD1)); \
155                                   (ADC12CTL1 |=  (CSTARTADD0))
    #define ADC12MEM_SET_2              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD0)); \
157                                     (ADC12CTL1 |=  (CSTARTADD1))
    #define ADC12MEM_SET_3              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2)); \
159                                     (ADC12CTL1 |=  (CSTARTADD1 | CSTARTADD0))
    #define ADC12MEM_SET_4              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD1 | CSTARTADD0)); \
161                                     (ADC12CTL1 |=  (CSTARTADD2))
    #define ADC12MEM_SET_5              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD1)); \
163                                     (ADC12CTL1 |=  (CSTARTADD2 | CSTARTADD0))
    #define ADC12MEM_SET_6              (ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD0)); \
165                                     (ADC12CTL1 |=  (CSTARTADD2 | CSTARTADD1))
    #define ADC12MEM_SET_7              (ADC12CTL1 &= ~(CSTARTADD3)); \
167                                     (ADC12CTL1 |=  (CSTARTADD2 | CSTARTADD1 | CSTARTADD0))
```

```
#define ADC12MEM_SET_8              (ADC12CTL1 &= ~(CSTARTADD2 | CSTARTADD1 | CSTARTADD0)); \
                                    (ADC12CTL1 |=  (CSTARTADD3))
#define ADC12MEM_SET_9              (ADC12CTL1 &= ~(CSTARTADD2 | CSTARTADD1)); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD0))
#define ADC12MEM_SET_10             (ADC12CTL1 &= ~(CSTARTADD2 | CSTARTADD0)); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD1))
#define ADC12MEM_SET_11             (ADC12CTL1 &= ~(CSTARTADD2)); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD1 | CSTARTADD0))

#define ADC12MEM_SET_12             (ADC12CTL1 &= ~(CSTARTADD1 | CSTARTADD0)); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD2))
#define ADC12MEM_SET_13             (ADC12CTL1 &= ~(CSTARTADD1); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD2 | CSTARTADD0)))
#define ADC12MEM_SET_14             (ADC12CTL1 &= ~(CSTARTADD0); \
                                    (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD2 | CSTARTADD1)))
#define ADC12MEM_SET_15             (ADC12CTL1 |=  (CSTARTADD3 | CSTARTADD2 | CSTARTADD1 | CSTARTADD0))


/*****************************************************************************
** Conversion sequence mode select
** CONSEQ0 --> Single-channel, single-conversion
** CONSEQ1 --> Sequence-of-channels
** CONSEQ2 --> Repeat-single-channel
** CONSEQ3 --> Repeat-sequence-of-channels
*****************************************************************************/
#define ADC12_SET_CONSEQ0           (ADC12CTL1 &= ~(CONSEQ1 | CONSEQ0))
#define ADC12_SET_CONSEQ1           (ADC12CTL1 &= ~(CONSEQ1); \
                                (ADC12CTL1 |=  (CONSEQ0)
#define ADC12_SET_CONSEQ2           (ADC12CTL1 &= ~(CONSEQ0); \
                                    (ADC12CTL1 |=  (CONSEQ1)
#define ADC12_SET_CONSEQ3           (ADC12CTL1 |=  (CONSEQ1 | CONSEQ0))

/*****************************************************************************
** Sample-and-hold pulse-mode select
** SHP0 --> SAMPCON signal is sourced from the sample-input signal
** SHP1 --> SAMPCON signal is sourced from the sampling timer
*****************************************************************************/
#define ADC12_SET_SHP0              (ADC12CTL1  &= ~(SHP))
#define ADC12_SET_SHP1              (ADC12CTL1  |=  (SHP))

/*****************************************************************************
** Select reference
** REF0 --> VR+ = AVCC and VR- = AVSS
** REF1 --> VR+ = VREF+ and VR- = AVSS
** REF2 --> VR+ = VeREF+ and VR- = AVSS
** REF3 --> VR+ = VeREF+ and VR- = AVSS
** REF4 --> VR+ = AVCC and VR- = VREF-/ VeREF-
** REF5 --> VR+ = VREF+ and VR- = VREF-/ VeREF-
** REF6 --> VR+ = VeREF+ and VR- = VREF-/ VeREF-
** REF7 --> VR+ = VeREF+ and VR- = VREF-/ VeREF
*****************************************************************************/
#define ADC12_SET_REF0              (ADC12MCTL0 |= SREF_0)
#define ADC12_SET_REF1              (ADC12MCTL0 |= SREF_1)
#define ADC12_SET_REF2              (ADC12MCTL0 |= SREF_2)
#define ADC12_SET_REF3              (ADC12MCTL0 |= SREF_3)
#define ADC12_SET_REF4              (ADC12MCTL0 |= SREF_4)
#define ADC12_SET_REF5              (ADC12MCTL0 |= SREF_5)
#define ADC12_SET_REF6              (ADC12MCTL0 |= SREF_6)
#define ADC12_SET_REF7              (ADC12MCTL0 |= SREF_7)

/*****************************************************************************
** Input channel select
**
*****************************************************************************/
#define ADC12_SET_INPUT_CH0         (ADC12MCTL0 |= INCH_0)
#define ADC12_SET_INPUT_CH1         (ADC12MCTL0 |= INCH_1)
#define ADC12_SET_INPUT_CH2         (ADC12MCTL0 |= INCH_2)
#define ADC12_SET_INPUT_CH3         (ADC12MCTL0 |= INCH_3)
#define ADC12_SET_INPUT_CH4         (ADC12MCTL0 |= INCH_4)
#define ADC12_SET_INPUT_CH5         (ADC12MCTL0 |= INCH_5)
#define ADC12_SET_INPUT_CH6         (ADC12MCTL0 |= INCH_6)
#define ADC12_SET_INPUT_CH7         (ADC12MCTL0 |= INCH_7)
#define ADC12_SET_TEMPERATUR        (ADC12MCTL0 |= INCH_10)


/*-------------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------------*/

uint16_t adc12_get_volt_sample(uint8_t);
uint16_t adc12_get_temp_sample(uint8_t);
void adc12_init(void);
void adc12_volt_init(uint8_t);
void adc12_temp_init(uint8_t); // added 04/08/2014 - EM
```

```
253 void adc12_disable(void);     // added 04/08/2014 - EM

255 #endif /* ADC12_H_ */
```

code/ZSK3V04/adc12.h

E.7 adg918.c

```
/*-------------------------------------------------------------------
    Description:    ADG918 RF-switch driver.
    Date:           11/22/2012
    Last Update:    11/22/2012
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#include "main.h"

/*-------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------*/

void adg918_init(void)
{
    ADG918_CTRL_PxDIR |= ADG918_CTRL_PIN;
}

void adg918_wakeup(void)
{
    ADG918_CTRL_PxOUT &= ~ADG918_CTRL_PIN;
}

void adg918_transceiver(void)
{
    ADG918_CTRL_PxOUT |= ADG918_CTRL_PIN;
}
```

code/ZSK3V04/adg918.c

E.8 adg918.h

```
/*-------------------------------------------------------------------
    Description:    ADG918 RF-switch driver.
    Date:           11/22/2012
    Last Update:    24.07.2014 - EM
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#ifndef ADG918_H_
#define ADG918_H_
#include "main.h"
/*-------------------------------------------------------------------
  Defines
-------------------------------------------------------------------*/

#define ADG918_CTRL_PxDIR ( P3DIR ) // changed 24.07.2014 - EM
#define ADG918_CTRL_PxOUT (P3OUT) // changed 24.07.2014 - EM
#define ADG918_CTRL_PIN ( BIT0 )  // changed 24.07.2014 - EM

/*-------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------*/

void adg918_init(void);
void adg918_wakeup(void);
void adg918_transceiver(void);

#endif /* ADG918_H_ */
```

code/ZSK3V04/adg918.h

E.9 as3930.c

```
/*-----------------------------------------------------------------------
    Description:    Driver for austriamicrosystems AS3930 Single Channel
                    Low Frequency Wakeup Receiver.
    Date:           10/02/2012
    Last Update:    10/03/2012
    Author:         Phillip Durdaut
-----------------------------------------------------------------------*/

#include "main.h"

/*-----------------------------------------------------------------------
  Defines
-----------------------------------------------------------------------*/

/* Configuration Modes (bits 15-14) */

#define AS3930_WRITE                0x00
#define AS3930_READ                 0x01
#define AS3930_COMMAND              0x03

/* Configuration Registers (bits 13-8) */

#define AS3930_R00                  0x00
#define AS3930_R01                  0x01
#define AS3930_R02                  0x02
#define AS3930_R03                  0x03
#define AS3930_R04                  0x04
#define AS3930_R05                  0x05
#define AS3930_R06                  0x06
#define AS3930_R07                  0x07
#define AS3930_R08                  0x08
#define AS3930_R09                  0x09
#define AS3930_R10                  0x0a
#define AS3930_R11                  0x0b
#define AS3930_R12                  0x0c
#define AS3930_R13                  0x0d

/* Commands (bits 7-0) */

#define AS3930_CLEAR_WAKE           0x00
#define AS3930_RESET_RSSI           0x01
#define AS3930_TRIM_OSC             0x02
#define AS3930_CLEAR_FALSE          0x03
#define AS3930_PRESET_DEFAULT       0x04

/*-----------------------------------------------------------------------
  Prototypes of the private functions
-----------------------------------------------------------------------*/

void as3930_spi_setup(void);
void as3930_spi_write_register(u8_t address, u8_t value);
char as3930_spi_read_register(u8_t address);
void as3930_spi_command(u8_t command);

/*-----------------------------------------------------------------------
  Public functions
-----------------------------------------------------------------------*/

void as3930_init(void)
{
    as3930_spi_setup();

    AS3930_WAKE_DIR_IN;
    AS3930_WAKE_IRQ_RISING_EDGE;
    AS3930_WAKE_IRQ_DISABLE;
    AS3930_WAKE_CLEAR_IRQ;
}

void as3930_config_no_pattern(void)
{
    as3930_spi_write_register(AS3930_R01, BIT5);  // Data correlation disable, Crystal oscillator disable
    //as3930_spi_write_register(AS3930_R07, BIT5);  // Automatic time-out after 50 ms
}

void as3930_preset_default(void)
{
    as3930_spi_command(AS3930_PRESET_DEFAULT);
}

void as3930_clear_wakeup(void) {
    as3930_spi_command(AS3930_CLEAR_WAKE);
}
```

```
83
84  u8_t as3930_get_rssi(void)
85  {
86      return (as3930_spi_read_register(AS3930_R10) & 0x1F);
87  }
88
89  /*---------------------------------------------------------------------------
90    Private functions
91  ---------------------------------------------------------------------------*/
92
93  void as3930_spi_setup(void)
94  {
95      AS3930_CS_PxDIR |= AS3930_CS_PIN;    // CS is output
96      AS3930_CS_PxOUT &= ~AS3930_CS_PIN;   // Chip disable
97
98      UCA0CTL1 |= UCSWRST;                 // Hold state machine in reset
99
100     UCA0CTL0 &= ~UCCKPH;                     // Data is changed on rising edge
101     UCA0CTL0 &= ~UCCKPL;                     // Inactive clock is low
102     UCA0CTL0 |= (UCMST | UCMSB | UCSYNC);    // MSB first, Master mode, Synchronous mode
103     UCA0CTL0 &= ~(UCMODE1 | UCMODE0);        // 3-pin SPI
104     UCA0MCTL = 0;                            // No modulation
105
106     // SMCLK / 2
107     UCA0CTL1 |= (UCSSEL1 | UCSSEL0);
108     UCA0BR0 = 2;
109     UCA0BR1 = 0;
110
111     // SPI functionality for pins
112     AS3930_SPI_PxSEL |= (AS3930_SPI_MOSI_PIN | AS3930_SPI_MISO_PIN | AS3930_SPI_CLK_PIN);
113     AS3930_SPI_PxDIR |= (AS3930_SPI_MOSI_PIN | AS3930_SPI_CLK_PIN); // MOSI and CLK are outputs
114     AS3930_SPI_PxDIR &= ~AS3930_SPI_MISO_PIN; // MISO is input
115
116     UCA0CTL1 &= ~UCSWRST;    // Initialize USART state machine
117 }
118
119 void as3930_spi_write_register(u8_t address, u8_t value)
120 {
121     AS3930_CS_PxOUT |= AS3930_CS_PIN;           // Chip enable
122     while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
123     UCA0TXBUF = address | (AS3930_WRITE << 6);  // Send configuration mode and register
124     while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
125     UCA0TXBUF = value;                          // Send value
126     while (UCA0STAT & UCBUSY);                  // Wait for TX complete
127     AS3930_CS_PxOUT &= ~AS3930_CS_PIN;          // Chip disable
128 }
129
130 char as3930_spi_read_register(u8_t address)
131 {
132     u8_t value;
133
134     AS3930_CS_PxOUT |= AS3930_CS_PIN;           // Chip enable
135     while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
136     UCA0TXBUF = address | (AS3930_READ << 6);   // Send configuration mode and register
137     while (UCA0STAT & UCBUSY);                  // Wait for TX to finish
138     UCA0TXBUF = 0;                              // Dummy write so we can read data
139     while (UCA0STAT & UCBUSY);                  // Wait for TX complete
140     value = UCA0RXBUF;                          // Read data
141     AS3930_CS_PxOUT &= ~AS3930_CS_PIN;          // Chip disable
142
143     return value;
144 }
145
146 void as3930_spi_command(u8_t command)
147 {
148     AS3930_CS_PxOUT |= AS3930_CS_PIN;               // Chip enable
149     while (UCA0STAT & UCBUSY);                      // Wait for TX to finish
150     UCA0TXBUF = command | (AS3930_COMMAND << 6);     // Send configuration mode and register
151     while (UCA0STAT & UCBUSY);                      // Wait for TX to finish
152     AS3930_CS_PxOUT &= ~AS3930_CS_PIN;              // Chip disable
153 }
```

code/ZSK3V04/as3930.c

E.10 as3930.h

```
/*-------------------------------------------------------------------
    Description:    Driver for austriamicrosystems AS3930 Single Channel
                    Low Frequency Wakeup Receiver.
    Date:           10/09/2012
    Last Update:    10/09/2012
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#ifndef AS3930_H_
#define AS3930_H_

#include "main.h"

/*-------------------------------------------------------------------
  Defines -> Need to be changed depending on hardware
-------------------------------------------------------------------*/

#define AS3930_CS_PxDIR             P5DIR
#define AS3930_CS_PxOUT             P5OUT
#define AS3930_CS_PIN               BIT1

#define AS3930_SPI_PxSEL            P1SEL
#define AS3930_SPI_PxDIR            P1DIR
#define AS3930_SPI_PxIN             P1IN
#define AS3930_SPI_MOSI_PIN         BIT6
#define AS3930_SPI_MISO_PIN         BIT5
#define AS3930_SPI_CLK_PIN          BIT7

#define AS3930_WAKE_PxDIR           P2DIR
#define AS3930_WAKE_PxIES           P2IES
#define AS3930_WAKE_PxIE            P2IE
#define AS3930_WAKE_PxIFG           P2IFG
#define AS3930_WAKE_PIN             BIT5

/*-------------------------------------------------------------------
  Macros
-------------------------------------------------------------------*/

#define AS3930_WAKE_DIR_IN              (AS3930_WAKE_PxDIR &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_RISING_EDGE     (AS3930_WAKE_PxIES &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_FALLING_EDGE    (AS3930_WAKE_PxIES |= AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_ENABLE          (AS3930_WAKE_PxIE |= AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_DISABLE         (AS3930_WAKE_PxIE &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_PENDING         ((AS3930_WAKE_PxIFG & AS3930_WAKE_PIN) == AS3930_WAKE_PIN)
#define AS3930_WAKE_CLEAR_IRQ           (AS3930_WAKE_PxIFG &= ~(AS3930_WAKE_PIN))

/*-------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------*/

void as3930_init(void);
void as3930_config_no_pattern(void);
void as3930_preset_default(void);
void as3930_clear_wakeup(void);
u8_t as3930_get_rssi(void);

#endif /* AS3930_H_ */
```

code/ZSK3V04/as3930.h

E.11 balancing.c

```
/*******************************************************************************
**     Description:    blancing.c
**     Hardware:       BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:           06/03/2013
**     Last Update:    18/07/2013
**     Author:         Nico Sassano
*******************************************************************************/

#include "main.h"

/*-----------------------------------------------------------------------------
  Public functions
-----------------------------------------------------------------------------*/

void balancing_init(void) {
    BALANCE_PORT_1_PxDIR |= BALANCE_PORT_1_PIN;
    BALANCE_PORT_2_PxDIR |= BALANCE_PORT_2_PIN;
}

void balancing_on(void) {
    BALANCE_PORT_1_PxOUT |= BALANCE_PORT_1_PIN;
    BALANCE_PORT_2_PxOUT |= BALANCE_PORT_2_PIN;
}

void balancing_port_1_on(void) {
    BALANCE_PORT_1_PxOUT |= BALANCE_PORT_1_PIN;
}

void balancing_port_2_on(void) {
    BALANCE_PORT_2_PxOUT |= BALANCE_PORT_2_PIN;
}

void balancing_off(void) {
    BALANCE_PORT_1_PxOUT &=~BALANCE_PORT_1_PIN;
    BALANCE_PORT_2_PxOUT &=~BALANCE_PORT_2_PIN;
}

void balancing_port_1_off(void) {
    BALANCE_PORT_1_PxOUT &=~BALANCE_PORT_1_PIN;
}

void balancing_port_2_off(void) {
    BALANCE_PORT_2_PxOUT &=~BALANCE_PORT_2_PIN;
}
```

code/ZSK3V04/balancing.c

E.12 balancing.h

```
/*******************************************************************************
**    Description:    blancing.h
**    Hardware:       BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:           06/03/2013
**    Last Update:    24/07/2014 - EM
**    Author:         Nico Sassano
*******************************************************************************/

#ifndef BALANCING_H_
#define BALANCING_H_
#include "main.h"

/*-----------------------------------------------------------------------------
  Defines
-----------------------------------------------------------------------------*/

#define BALANCE_PORT_1_PxDIR  (P2DIR) // changed 24/07/2014 - EM
#define BALANCE_PORT_1_PxOUT  (P2OUT) // changed 24/07/2014 - EM
#define BALANCE_PORT_1_PIN    (BIT3)  // changed 24/07/2014 - EM

#define BALANCE_PORT_2_PxDIR  (P3DIR) // changed 24/07/2014 - EM
#define BALANCE_PORT_2_PxOUT  (P3OUT) // changed 24/07/2014 - EM
#define BALANCE_PORT_2_PIN    (BIT7)  // changed 24/07/2014 - EM
/*-----------------------------------------------------------------------------
  Public functions
-----------------------------------------------------------------------------*/

void balancing_init(void);
void balancing_on(void);
void balancing_off(void);

#endif /* BALANCING_H_ */
```

code/ZSK3V04/balancing.h

E.13 cc430.c

```
/*******************************************************************************
**     Description:     cc430.c
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            24.09.2014
**     Last Update:     24.09.2014
**     Author:          Eike Mense
*******************************************************************************/

#include "main.h"

#define PATABLE_LENGTH 2
#define  PATABLE_VAL         (0x51)
uint8_t cc430_patable[PATABLE_LENGTH] = { 0x12, 0xc0 };

void cc430_set_tx(uint8_t brate)
{
  cc430_init_tx();
  cc430_reset();          // Reset chip and go to idle state
  cc430_config_packet(brate); // Configure the transceiver for sending packets
}

void cc430_set_rx(uint8_t brate)
{
    cc430_init_rx();
    cc430_reset();          // Reset chip and go to idle state
    cc430_config_packet(brate); // Configure the transceiver for sending packets
}

void cc430_init_rx(void)
{
  CC430_END_OF_PKT_IRQ_EDGE;
  CC430_END_OF_PKT_IRQ_DISABLE; // TODO Check why Nico Uses Sync detect for Receiving.
  CC430_END_OF_PKT_CLEAR_IRQ;
//  CC430_SYNC_DETECT_IRQ_EDGE;
//  CC430_SYNC_DETECT_IRQ_DISABLE;
//  CC430_SYNC_DETECT_CLEAR_IRQ;
}

void cc430_init_burst(void)
{

  PMAPPWD = 0x02D52;                          // Get write-access to port mapping regs
  CC430_GDO2_OUT_PMAP = PM_RFGDO2;
  PMAPPWD = 0;                                // Lock port mapping registers
  CC430_GDO2_OUT_PxDIR |= CC430_GDO2_OUT_PIN;      // P2.7 output
  CC430_GDO2_OUT_PxSEL |= CC430_GDO2_OUT_PIN;      // P2.7 Port Map functions
  CC43_BURST_SIG_DIR_IN;                  // P2.6 as inut
                                  // P2.6 Hi/Lo edge
  CC430_BURST_SIG_IRQ_FALLING_EDGE;
  CC430_BURST_SIG_IRQ_DISABLE;
  CC430_BURST_SIG_CLEAR_IRQ;
}

void cc430_init_tx(void)
{
  CC430_END_OF_PKT_IRQ_EDGE;
  CC430_END_OF_PKT_IRQ_DISABLE;
  CC430_END_OF_PKT_CLEAR_IRQ;
}

void cc430_init(void)
{
  CC430_SYNC_DETECT_IRQ_EDGE;
  CC430_SYNC_DETECT_IRQ_DISABLE;
  CC430_SYNC_DETECT_CLEAR_IRQ;
}

void cc430_power_up_reset(void)
{
  cc430_reset();
}
void cc430_reset(void)
{
  Strobe (RF_SRES) ; // Resets Radio Core
}

void cc430_config_no_packet(void)
{
  WriteSingleReg(IOCFG2,0x0B);   //GDO2 Output Configuration
    WriteSingleReg(IOCFG0,0x2D);   //GDO0 Output Configuration
    WriteSingleReg(FIFOTHR,0x47); //RX FIFO and TX FIFO Thresholds
    WriteSingleReg(PKTCTRL0,0x32);//Packet Automation Control
```

```
    WriteSingleReg(FSCTRL1,0x06); //Frequency Synthesizer Control
    WriteSingleReg(FREQ2,0x10);   //Frequency Control Word, High Byte
    WriteSingleReg(FREQ1,0xB1);   //Frequency Control Word, Middle Byte
    WriteSingleReg(FREQ0,0x3B);   //Frequency Control Word, Low Byte
    WriteSingleReg(MDMCFG4,0xCA); //Modem Configuration
    WriteSingleReg(MDMCFG3,0x93); //Modem Configuration
    WriteSingleReg(MDMCFG2,0x30); //Modem Configuration
    WriteSingleReg(DEVIATN,0x34); //Modem Deviation Setting
    WriteSingleReg(MCSM0,0x10);   //Main Radio Control State Machine Configuration
    WriteSingleReg(FOCCFG,0x16);  //Frequency Offset Compensation Configuration
    WriteSingleReg(WORCTRL,0xFB); //Wake On Radio Control
    WriteSingleReg(FREND0,0x11);  //Front End TX Configuration
    WriteSingleReg(FSCAL3,0xE9);  //Frequency Synthesizer Calibration
    WriteSingleReg(FSCAL2,0x2A);  //Frequency Synthesizer Calibration
    WriteSingleReg(FSCAL1,0x00);  //Frequency Synthesizer Calibration
    WriteSingleReg(FSCAL0,0x1F);  //Frequency Synthesizer Calibration
    WriteSinglePATable(PATABLE_VAL);
}

void cc430_config_no_packet_rx(void)
{

     WriteSingleReg(IOCFG2, 0x0D);     // 0x0D -> Serial Data Output. Used for asynchronous serial mode
     WriteSingleReg(IOCFG1, 0x2E);     // 0x2E -> High impedance (3-state)
     WriteSingleReg(IOCFG0, 0x36);     // 0x36 -> CLK_XOSC/8
     WriteSingleReg(FIFOTHR, 0x47); // RX FIFO and TX FIFO Thresholds
     WriteSingleReg(PKTCTRL0, 0x32);  // Packet Automation Control 0
     WriteSingleReg(FSCTRL1, 0x06);    // Frequency Synthesizer Control 1
     WriteSingleReg(FREQ2, 0x10);    // Frequency Control Word, High Byte
     WriteSingleReg(FREQ1, 0xB1);    // Frequency Control Word, Middle Byte
     WriteSingleReg(FREQ0, 0x3B);    //Frequency Control Word, LOW Byte

     /*************************************************
     // Modem Configuration 4
     // 250kBaud             ->  0xxD
     // RX Filter 58.035714  ->  0xFx
     WriteSingleReg(MDMCFG4, 0x5D);
     // Modem Configuration 3
     // 250kBaud -> 0x3B
     WriteSingleReg(MDMCFG3, 0x3B);
     // Modem Configuration 2
     // OOK -> 0x30
     WriteSingleReg(MDMCFG2, 0x30);
     // Modem Configuration 1
     // Channel spacing 199.951172kHz -> 0x22
     WriteSingleReg(MDMCFG1, 0x22);
     // Modem Configuration 0
     // Channel spacing 199.951172kHz -> 0xF8
     WriteSingleReg(MDMCFG0, 0xF8);
     *************************************************/


     WriteSingleReg(DEVIATN, 0x15);    // Modem Deviation Setting
     WriteSingleReg(MCSM2, 0x07);      // Main Radio Control State Machine Configuration 2
     WriteSingleReg(MCSM1, 0x30);      // Main Radio Control State Machine Configuration 1
     WriteSingleReg(MCSM0, 0x18);      // Main Radio Control State Machine Configuration 0
     WriteSingleReg(FOCCFG, 0x16);  // Frequency Offset Compensation Configuration
     WriteSingleReg(BSCFG, 0x6C);      // Bit Synchronization Configuration
     WriteSingleReg(AGCCTRL2, 0x03);   // AGC Control 2
     WriteSingleReg(AGCCTRL1, 0x40);   // AGC Control 1
     WriteSingleReg(AGCCTRL0, 0x91);   // AGC Control 0
     WriteSingleReg(WORCTRL, 0x8F); // Wake On Radio Control
     WriteSingleReg(FREND1, 0x56);   // Front End RX Configuration 1
     WriteSingleReg(FREND0, 0x11);   // Front End RX Configuration 0

     WriteBurstPATable(cc430_patable, PATABLE_LENGTH);  // output power
}
void cc430_config_packet(uint8_t brate)
{
    WriteSingleReg(IOCFG0,0x06);  //GDO0 Output Configuration
    WriteSingleReg(IOCFG2,0x29);  //GDO0 Output Configuration
    WriteSingleReg(FIFOTHR,0x47); //RX FIFO and TX FIFO Thresholds
    WriteSingleReg(SYNC1,0x81); //   Sync-Word Hi-byte
    WriteSingleReg(SYNC0,0x81); //   Sync-Word Lo-byte
      WriteSingleReg(PKTCTRL0,0x04); // Flush RX packets when CRC is not OK, Address check and 0x00 broadcast.
    WriteSingleReg(PKTCTRL1,0x0A); //     // No whitening, FIFO mode, CRC enabled, Fixed packet length
    WriteSingleReg(ADDR,ADDRESS_THIS_SENSOR); //     // Adress of this Sensor
    WriteSingleReg(FSCTRL1,0x06); // IF frequency: 152.34375 kHz
    WriteSingleReg(FREQ2,0x10);   // Carrier frequency: 433.999969 MHz
    WriteSingleReg(FREQ1,0xB1);   //
    WriteSingleReg(FREQ0,0x3B);   //

      WriteSingleReg(AGCCTRL2, 0x05);
      WriteSingleReg(AGCCTRL1, 0x00);
      WriteSingleReg(AGCCTRL0, 0x91);
```

```
168     switch(brate) {
            /*****************************************************************************************
170         ** Übertragungstest
            ** 40 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
172         ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
            *****************************************************************************************/
174         case 40: {
                WriteSingleReg(MDMCFG4, 0x8A);
176             WriteSingleReg(MDMCFG3, 0x93);
                WriteSingleReg(MDMCFG2, 0x33);
178             WriteSingleReg(MDMCFG1, 0x22);
                WriteSingleReg(MDMCFG0, 0x7A);
180         }break;

182         /*****************************************************************************************
            ** Übertragungstest
184         ** 59.906 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
            ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
186         *****************************************************************************************/
            case 60: {
188             WriteSingleReg(MDMCFG4, 0x8B);
                WriteSingleReg(MDMCFG3, 0x2E);
190             WriteSingleReg(MDMCFG2, 0x33);
                WriteSingleReg(MDMCFG1, 0x22);
192             WriteSingleReg(MDMCFG0, 0x7A);
            }break;
194

            /*****************************************************************************************
196         ** Übertragungstest
            ** 79.9408 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
198         ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
            *****************************************************************************************/
200         case 80: {
                WriteSingleReg(MDMCFG4, 0x8B);
202             WriteSingleReg(MDMCFG3, 0x93);
                WriteSingleReg(MDMCFG2, 0x33);
204             WriteSingleReg(MDMCFG1, 0x22);
                WriteSingleReg(MDMCFG0, 0x7A);
206         }break;

208         /*****************************************************************************************
            ** Übertragungstest
210         ** 99.9756 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
            ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
212         *****************************************************************************************/
            case 100: {
214             WriteSingleReg(MDMCFG4, 0x8B);
                WriteSingleReg(MDMCFG3, 0xF8);
216             WriteSingleReg(MDMCFG2, 0x33);
                WriteSingleReg(MDMCFG1, 0x22);
218             WriteSingleReg(MDMCFG0, 0x7A);
            }break;
220

            /*****************************************************************************************
222         ** Übertragungstest
            ** 119.812 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 406.250000 kHz
224         ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
            *****************************************************************************************/
226         case 120: {
                WriteSingleReg(MDMCFG4, 0x8C);
228             WriteSingleReg(MDMCFG3, 0x2E);
                WriteSingleReg(MDMCFG2, 0x33);
230             WriteSingleReg(MDMCFG1, 0x22);
                WriteSingleReg(MDMCFG0, 0x7A);
232         }break;

234         /*****************************************************************************************
            ** Übertragungstest
236         ** 140.045 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 406.250000 kHz
            ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
238         *****************************************************************************************/
            case 140: {
240             WriteSingleReg(MDMCFG4, 0x8C);
                WriteSingleReg(MDMCFG3, 0x61);
242             WriteSingleReg(MDMCFG2, 0x33);
                WriteSingleReg(MDMCFG1, 0x22);
244             WriteSingleReg(MDMCFG0, 0x7A);
            }break;
246

            /*****************************************************************************************
248         ** Übertragungstest
            ** 159.882 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 406.250000 kHz
250         ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
            *****************************************************************************************/
```

```
252             case 160: {
                    WriteSingleReg(MDMCFG4, 0x8C);
254                 WriteSingleReg(MDMCFG3, 0x93);
                    WriteSingleReg(MDMCFG2, 0x33);
256                 WriteSingleReg(MDMCFG1, 0x22);
                    WriteSingleReg(MDMCFG0, 0x7A);
258             }break;

260             /*****************************************************************************************
                ** Übertragungstest
262             ** 180.115 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 406.250000 kHz
                ** No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
264             *****************************************************************************************/
                case 180: {
266                 WriteSingleReg(MDMCFG4, 0x8C);
                    WriteSingleReg(MDMCFG3, 0xC6);
268                 WriteSingleReg(MDMCFG2, 0x33);
                    WriteSingleReg(MDMCFG1, 0x22);
270                 WriteSingleReg(MDMCFG0, 0x7A);
                }break;
272
            }
274     WriteSingleReg(DEVIATN,0x15); //Modem Deviation Setting
        WriteSingleReg(MCSM0,0x10);   // Calibrate when going from IDLE to RX or TX, Crystal off when in SLEEP state
276     WriteSingleReg(FOCCFG,0x16);  // FCL gain: 3000, Saturation point for the frequency offset compensation algorithm -> SmartRF Studio

278     WriteBurstPATable(cc430_patable, PATABLE_LENGTH);

280     WriteSingleReg(FREND0,0x11);  //Front End TX Configuration
        WriteSingleReg(FSCAL3,0xEA);  //Frequency Synthesizer Calibration
282     WriteSingleReg(FSCAL2,0x2A);  //Frequency Synthesizer Calibration
        WriteSingleReg(FSCAL1,0x00);  //Frequency Synthesizer Calibration
284     WriteSingleReg(FSCAL0,0x1F);  //Frequency Synthesizer Calibration
}
286
void cc430_fill_tx_fifo(uint8_t * buffer, uint8_t length)
288 {
    // Clear TX FIFO
290 cc430_clear_tx_fifo();

292 // Transfer the bytes to the TX fifo of the transceiver
    WriteBurstReg(RF_TXFIFOWR, buffer, length);
294 }

296 void cc430_read_rx_fifo(uint8_t * buffer, uint8_t length)
298 {
  // Disable the receiver
      cc430_idle();
300 // Transfer the bytes via SPI from the RX fifo
      ReadBurstReg(RF_RXFIFORD, buffer, length);
302 }
303 void cc430_enable_crc(void)
304 {
  uint8_t current_state;
306 current_state = ReadSingleReg(PKTCTRL1);
  WriteSingleReg(PKTCTRL1,  current_state | CC430_CRC_EN );
308 }

310 void cc430_disable_crc(void)
312 {
  uint8_t current_state;
  current_state = ReadSingleReg(PKTCTRL1);
314 WriteSingleReg(PKTCTRL1,  current_state & ~CC430_CRC_EN );
315 }
316 void cc430_clear_tx_fifo(void)
317 {
318   Strobe( RF_SIDLE );      // Needs to be in Idle state to Flush Fifo
    Strobe( RF_SFTX );           // Strobe SFTX -> Flush TX-Fifo
320 }

322 void cc430_clear_rx_fifo(void)
323 {
324   Strobe( RF_SIDLE );      // Needs to be in Idle state to Flush Fifo
    Strobe( RF_SFRX );           // Strobe SFTX -> Flush RX-Fifo
326 }

328 uint8_t  cc430_get_rxbytes(void)
329 {
330   return (ReadSingleReg(RXBYTES));
331 }
332
333 void cc430_sleep(void)
334 {
    Strobe( RF_SPWD );
336 }
```

```
338 void cc430_idle(void)
{
340   Strobe( RF_SIDLE );
}
342
void cc430_tx_carrier(void)
344 {
   Strobe( RF_SIDLE );
346   Strobe( RF_STX );
}
348 void cc430_tx(uint8_t packages_to_tx)
{
350      // DATA packet length
       WriteSingleReg(PKTLEN, packages_to_tx);
352
       // Enable CRC calculation when transmitting data
354      cc430_enable_crc();

356    // TX state
       Strobe( RF_STX);
358
       // Wait 10 ms for TX finished
360    //delay_ms(8);
}
362
void cc430_rx(uint8_t packages_to_rx)
364 {
    // DATA packet length
366   WriteSingleReg(PKTLEN, packages_to_rx);

368    // Enable CRC check when receiving data
    cc430_enable_crc();
370
    // Clear RX FIFO
372    cc430_clear_rx_fifo();

374    // RX state
    Strobe( RF_SRX);
376 }

378 void cc430_burst_rx(void)
{
380   Strobe( RF_SIDLE );
     Strobe( RF_SRX);
382 }

384 uint8_t cc430_get_partnum(void)
{
386   return ReadSingleReg(PARTNUM);
}
388
uint8_t cc430_get_version(void)
390 {
   return ReadSingleReg(VERSION);
392 }

394 uint8_t cc430_get_marcstate(void)
{
396   return ReadSingleReg(MARCSTATE);
}
```

code/ZSK3V04/cc430.c

E.14 cc430.h

```
/*******************************************************************************
**     Description:     cc430.h
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            24.09.2014
**     Last Update:     24.09.2014
**     Author:          Eike Mense
*******************************************************************************/


#ifndef cc430_H_
#define cc430_H_

#include "main.h"


#define RFIFG0  0x0001
#define RFIFG1  0x0002
#define RFIFG2  0x0004
#define RFIFG3  0x0008
#define RFIFG4  0x0010
#define RFIFG5  0x0020
#define RFIFG6  0x0040
#define RFIFG7  0x0080
#define RFIFG8  0x0100
#define RFIFG9  0x0200
#define RFIFG10 0x0400
#define RFIFG11 0x0800
#define RFIFG12 0x1000
#define RFIFG13 0x2000
#define RFIFG14 0x4000
#define RFIFG15 0x8000

#define RFIE0   0x0001// enables the associated RFIFGx to cause an interrupt.
#define RFIE1   0x0002
#define RFIE2   0x0004
#define RFIE3   0x0008
#define RFIE4   0x0010
#define RFIE5   0x0020
#define RFIE6   0x0040
#define RFIE7   0x0080
#define RFIE8   0x0100
#define RFIE9   0x0200
#define RFIE10  0x0400
#define RFIE11  0x0800
#define RFIE12  0x1000
#define RFIE13  0x2000
#define RFIE14  0x4000
#define RFIE15  0x8000

#define RFIES0  0x0001
#define RFIES1  0x0002
#define RFIES2  0x0004
#define RFIES3  0x0008
#define RFIES4  0x0010
#define RFIES5  0x0020
#define RFIES6  0x0040
#define RFIES7  0x0080
#define RFIES8  0x0100
#define RFIES9  0x0200  // Interrupt edge select bit RFIESx allows to trigger an interrupt on the positive (RFIES = 0) or on the
      negative (RFIES = 1) edge of the associated signal
#define RFIES10 0x0400  // Interrupt edge select bit RFIESx allows to trigger an interrupt on the positive (RFIES = 0) or on the
      negative (RFIES = 1) edge of the associated signal
#define RFIES11 0x0800
#define RFIES12 0x1000
#define RFIES13 0x2000
#define RFIES14 0x4000
#define RFIES15 0x8000

#define RFIN0   0x0001  //The input bit RFINx allows to query the actual status of a signal and RFIEx
#define RFIN1   0x0002
#define RFIN2   0x0004
#define RFIN3   0x0008
#define RFIN4   0x0010
#define RFIN5   0x0020
#define RFIN6   0x0040
#define RFIN7   0x0080
#define RFIN8   0x0100
#define RFIN9   0x0200
#define RFIN10  0x0400
#define RFIN11  0x0800
#define RFIN12  0x1000
#define RFIN13  0x2000
```

```
#define RFIN14   0x4000
#define RFIN15   0x8000

#define CC430_GDO2_PxDIR          P2DIR
#define CC430_GDO2_PxIN           P2IN
#define CC430_GDO2_PxIES          P2IES
#define CC430_GDO2_PxIE         P2IE
#define CC430_GDO2_PxIFG        P2IFG
#define CC430_GDO2_PIN          BIT6
#define CC430_GDO2_POS            6

#define CC430_GDO2_OUT_PxDIR     P2DIR
#define CC430_GDO2_OUT_PxSEL   P2SEL
#define CC430_GDO2_OUT_PIN      BIT7
#define CC430_GDO2_OUT_PMAP    P2MAP7

/*-------------------------------------------------------------------------
  Macros
-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------
  Macros for cc430
-------------------------------------------------------------------------*/
// Sync word detected
#define CC430_SYNC_DETECT_IRQ_EDGE       (RF1AIES &= ~RFIES9)
#define CC430_SYNC_DETECT_IRQ_ENABLE     (RF1AIE |= RFIE9)
#define CC430_SYNC_DETECT_IRQ_DISABLE      (RF1AIE &= ~RFIE9)
#define CC430_SYNC_DETECT_IRQ_PENDING      ((RF1AIFG & RFIFG9) == RFIFG9)
#define CC430_SYNC_DETECT_CLEAR_IRQ      (RF1AIFG &= ~(RFIFG9))

// End of Packet
#define CC430_END_OF_PKT_IRQ_EDGE      (RF1AIES |= RFIES9)
#define CC430_END_OF_PKT_IRQ_ENABLE      (RF1AIE |= RFIE9)
#define CC430_END_OF_PKT_IRQ_DISABLE     (RF1AIE &= ~RFIE9)
#define CC430_END_OF_PKT_IRQ_PENDING     ((RF1AIFG & RFIFG9) == RFIFG9)
#define CC430_END_OF_PKT_CLEAR_IRQ       (RF1AIFG &= ~(RFIFG9))
//#define CC1101_GDO2_IRQ_FALLING_EDGE     (RF1AIES |= RFIES2)

//
// CRC ok //

#define CC430_CRC_OK_IRQ_EDGE         (RF1AIES &= ~RFIES10)
#define CC430_CRC_OK_IRQ_ENABLE             (RF1AIE |= RFIE10)
#define CC430_CRC_OK_IRQ_DISABLE            (RF1AIE &= ~RFIE10)
#define CC430_CRC_OK_IRQ_PENDING            ((RF1AIFG & RFIFG10) == RFIFG10)
#define CC430_CRC_OK_CLEAR_IRQ              (RF1AIFG &= ~(RFIFG10))

// Burst- Syncronisation+

#define CC43_BURST_SIG_DIR_IN               (CC430_GDO2_PxDIR &= ~CC430_GDO2_PIN)
#define CC430_BURST_SIG_IN                    ((CC430_GDO2_PxIN & CC430_GDO2_PIN) >> CC430_GDO2_POS)
#define CC430_BURST_SIG_IRQ_RISING_EDGE     (CC430_GDO2_PxIES &= ~CC430_GDO2_PIN)
#define CC430_BURST_SIG_IRQ_FALLING_EDGE    (CC430_GDO2_PxIES |= CC430_GDO2_PIN)
#define CC430_BURST_SIG_IRQ_ENABLE          (CC430_GDO2_PxIE |= CC430_GDO2_PIN)
#define CC430_BURST_SIG_IRQ_DISABLE         (CC430_GDO2_PxIE &= ~CC430_GDO2_PIN)
#define CC430_BURST_SIG_IRQ_PENDING         ((CC430_GDO2_PxIFG & CC430_GDO2_PIN) == CC430_GDO2_PIN)
#define CC430_BURST_SIG_CLEAR_IRQ           (CC430_GDO2_PxIFG &= ~(CC430_GDO2_PIN))

//#define CC430_BURST_SIG_IRQ_EDGE         (RF1AIES &= ~RFIES2)
//#define CC430_BURST_SIG_IRQ_ENABLE             (RF1AIE |= RFIE2)
//#define CC430_BURST_SIG_IRQ_DISABLE            (RF1AIE &= ~RFIE2)
//#define CC430_BURST_SIG_IRQ_PENDING            ((RF1AIFG & RFIFG2) == RFIFG2)
//#define CC430_BURST_SIG_CLEAR_IRQ              (RF1AIFG &= ~(RFIFG2))

//#define CC1101_GDO0_DIR_IN
//#define CC1101_GDO0_IN                   ((RF1AIN & RFIN10) >> RFIN2)
//
//#define CC1101_GDO0_IRQ_RISING_EDGE      (RF1AIES &= ~RFIES10)
//#define CC1101_GDO0_IRQ_FALLING_EDGE     (RF1AIES |= RFIES10)
//#define CC1101_GDO0_IRQ_ENABLE           (RF1AIE |= RFIE10)
//#define CC1101_GDO0_IRQ_DISABLE          (RF1AIE &= ~RFIE10)
//#define CC1101_GDO0_IRQ_PENDING          ((RF1AIFG & RFIFG10) == RFIFG10)
//#define CC1101_GDO0_CLEAR_IRQ            (RF1AIFG &= ~(RFIFG10))

/*-------------------------------------------------------------------------
  Defines
-------------------------------------------------------------------------*/
/* Additional Defines for cc430  */

#define GDx_TriState 0x2E

#define CC430_CRC_EN 0x04

/* Command Strobes (Table 42 in datasheet) */
```

```
#define cc430_CS_SRES               0x30
#define cc430_CS_SFSTXON            0x31
#define cc430_CS_SXOFF              0x32
#define cc430_CS_SCAL               0x33
#define cc430_CS_SRX                0x34
#define cc430_CS_STX                0x35
#define cc430_CS_SIDLE              0x36
#define cc430_CS_SAFC               0x37
#define cc430_CS_SWOR               0x38
#define cc430_CS_SPWD               0x39
#define cc430_CS_SFRX               0x3A
#define cc430_CS_SFTX               0x3B
#define cc430_CS_SWORRST            0x3C
#define cc430_CS_SNOP               0x3D

/* Configuration Registers */

#define cc430_CR_IOCFG2             0x00    /* GDO2 output pin configuration */
#define cc430_CR_IOCFG1             0x01    /* GDO1 output pin configuration */
#define cc430_CR_IOCFG0             0x02    /* GDO0 output pin configuration */
#define cc430_CR_FIFOTHR            0x03    /* RX FIFO and TX FIFO thresholds */
#define cc430_CR_SYNC1              0x04    /* Sync word, high byte */
#define cc430_CR_SYNC0              0x05    /* Sync word, low byte */
#define cc430_CR_PKTLEN             0x06    /* Packet length */
#define cc430_CR_PKTCTRL1           0x07    /* Packet automation control */
#define cc430_CR_PKTCTRL0           0x08    /* Packet automation control */
#define cc430_CR_ADDR               0x09    /* Device address */
#define cc430_CR_CHANNR             0x0A    /* Channel number */
#define cc430_CR_FSCTRL1            0x0B    /* Frequency synthesizer control */
#define cc430_CR_FSCTRL0            0x0C    /* Frequency synthesizer control */
#define cc430_CR_FREQ2              0x0D    /* Frequency control word, high byte */
#define cc430_CR_FREQ1              0x0E    /* Frequency control word, middle byte */
#define cc430_CR_FREQ0              0x0F    /* Frequency control word, low byte */
#define cc430_CR_MDMCFG4            0x10    /* Modem configuration */
#define cc430_CR_MDMCFG3            0x11    /* Modem configuration */
#define cc430_CR_MDMCFG2            0x12    /* Modem configuration */
#define cc430_CR_MDMCFG1            0x13    /* Modem configuration */
#define cc430_CR_MDMCFG0            0x14    /* Modem configuration */
#define cc430_CR_DEVIATN            0x15    /* Modem deviation setting */
#define cc430_CR_MCSM2              0x16    /* Main Radio Cntrl State Machine config */
#define cc430_CR_MCSM1              0x17    /* Main Radio Cntrl State Machine config */
#define cc430_CR_MCSM0              0x18    /* Main Radio Cntrl State Machine config */
#define cc430_CR_FOCCFG             0x19    /* Frequency Offset Compensation config */
#define cc430_CR_BSCFG              0x1A    /* Bit Synchronization configuration */
#define cc430_CR_AGCCTRL2           0x1B    /* AGC control */
#define cc430_CR_AGCCTRL1           0x1C    /* AGC control */
#define cc430_CR_AGCCTRL0           0x1D    /* AGC control */
#define cc430_CR_WOREVT1            0x1E    /* High byte Event 0 timeout */
#define cc430_CR_WOREVT0            0x1F    /* Low byte Event 0 timeout */
#define cc430_CR_WORCTRL            0x20    /* Wake On Radio control */
#define cc430_CR_FREND1             0x21    /* Front end RX configuration */
#define cc430_CR_FREND0             0x22    /* Front end TX configuration */
#define cc430_CR_FSCAL3             0x23    /* Frequency synthesizer calibration */
#define cc430_CR_FSCAL2             0x24    /* Frequency synthesizer calibration */
#define cc430_CR_FSCAL1             0x25    /* Frequency synthesizer calibration */
#define cc430_CR_FSCAL0             0x26    /* Frequency synthesizer calibration */
#define cc430_CR_RCCTRL1            0x27    /* RC oscillator configuration */
#define cc430_CR_RCCTRL0            0x28    /* RC oscillator configuration */
#define cc430_CR_FSTEST             0x29    /* Frequency synthesizer cal control */
#define cc430_CR_PTEST              0x2A    /* Production test */
#define cc430_CR_AGCTEST            0x2B    /* AGC test */
#define cc430_CR_TEST2              0x2C    /* Various test settings */
#define cc430_CR_TEST1              0x2D    /* Various test settings */
#define cc430_CR_TEST0              0x2E    /* Various test settings */

/* Status Registers */
#define cc430_SR_PARTNUM            0x30    /* Part number */
#define cc430_SR_VERSION            0x31    /* Current version number */
#define cc430_SR_FREQEST            0x32    /* Frequency offset estimate */
#define cc430_SR_LQI                0x33    /* Demodulator estimate for link quality */
#define cc430_SR_RSSI               0x34    /* Received signal strength indication */
#define cc430_SR_MARCSTATE          0x35    /* Control state machine state */
#define cc430_SR_WORTIME1           0x36    /* High byte of WOR timer */
#define cc430_SR_WORTIME0           0x37    /* Low byte of WOR timer */
#define cc430_SR_PKTSTATUS          0x38    /* Current GDOx status and packet status */
#define cc430_SR_VCO_VC_DAC         0x39    /* Current setting from PLL cal module */
#define cc430_SR_TXBYTES            0x3A    /* Underflow and # of bytes in TXFIFO */
#define cc430_SR_RXBYTES            0x3B    /* Overflow and # of bytes in RXFIFO */
#define cc430_SR_RCCTRL1_STATUS     0x3C    /* Last RC oscilator calibration results */
#define cc430_SR_RCCTRL0_STATUS     0x3D    /* Last RC oscilator calibration results */

/* Single / Burst access */
#define cc430_WRITE_BURST           0x40
#define cc430_READ_SINGLE           0x80
```

```
251 #define cc430_READ_BURST            0xC0

253 /* Memory locations */
    #define cc430_ML_PATABLE            0x3E
255 #define cc430_ML_TXFIFO             0x3F
    #define cc430_ML_RXFIFO             0x3F
257

259 /*---------------------------------------------------------------------
      Public functions
261 ---------------------------------------------------------------------*/
    void cc430_set_tx(uint8_t);
263 void cc430_set_rx(uint8_t);
    void cc430_init_rx(void);
265 void cc430_init_tx(void);
    void cc430_init_burst(void);
267 void cc430_init(void);
    void cc430_power_up_reset(void);
269 void cc430_reset(void);
    void cc430_config_no_packet(void);
271 void cc430_config_no_packet_rx(void);
    void cc430_config_packet(uint8_t brate);
273 void cc430_fill_tx_fifo(uint8_t * buffer, uint8_t length);
    void cc430_read_rx_fifo(uint8_t * buffer, uint8_t length);
275 void cc430_enable_crc(void);
    void cc430_disable_crc(void);
277 void cc430_clear_tx_fifo(void);
    void cc430_clear_rx_fifo(void);
279 uint8_t cc430_get_rxbytes(void);
    void cc430_sleep(void);
281 void cc430_idle(void);
    void cc430_tx_carrier(void);
283 void cc430_tx(uint8_t);
    void cc430_rx(uint8_t);
285 void cc430_burst_rx(void);
    uint8_t cc430_get_partnum(void);
287 uint8_t cc430_get_version(void);
    uint8_t cc430_get_marcstate(void);
289

    #endif /* cc430_H_ */
```

code/ZSK3V04/cc430.h

E.15 clk.c

```
/*******************************************************************************
**    Description:    clk.h
**    Hardware:       BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:           24/07/2014
**    Last Update:    06/08/2014 - EM
**    Author:         Eike Mense
**    Note:         Funktionsnamen und case-Struktur entnommen aus clk.c
**           nach Sassano für  BATSEN ZS Klasse 3 v0.2
*******************************************************************************/

#include "main.h"

extern uint8_t clk_set;

void clk_init_startup(void) {   // 06.05.13 NS

    clk_set = 1;    //CLK ist auf 1MHz gesetzt


    //DCOCTL = (DCO0 | DCO1 | DCO2 | MOD3 | MOD2 | MOD1 | MOD0);


    // Set external XT off
    UCSCTL6  |= XT2OFF;

    __bis_SR_register(SCG0);                  // Disable the FLL control loop
    UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_3;                      // Select DCO range 16MHz operation
    UCSCTL2 = FLLD_1 + DCO_MP_1MHZ;                     // Set DCO Multiplier for 8MHz
                                              // (N + 1) * FLLRef = Fdco
                                              // (249 + 1) * 32768 = 8MHz
                                              // Set FLL Div = fDCOCLK/2
    __bic_SR_register(SCG0);                  // Enable the FLL control loop
    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // UG for optimization.
    // 32 x 32 x 1 MHz / 32768 Hz = 31250 = MCLK cycles for DCO to settle
    __delay_cycles(31250);

}


void clk_init_MHz(uint8_t mhz) {

    switch(mhz) {
        case 1: {
            clk_set = 1;    //CLK ist auf 1MHz gesetzt
            __bis_SR_register(SCG0);                  // Disable the FLL control loop
            UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
            UCSCTL1 = DCORSEL_3;                      // Select DCO range 16MHz operation
            UCSCTL2 = FLLD_1 + DCO_MP_1MHZ;                     // Set DCO Multiplier for 8MHz
                                                      // (N + 1) * FLLRef = Fdco
                                                      // (249 + 1) * 32768 = 8MHz
                                                      // Set FLL Div = fDCOCLK/2
            __bic_SR_register(SCG0);                  // Enable the FLL control loop
            // Worst-case settling time for the DCO when the DCO range bits have been
            // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
            // UG for optimization.
            // 32 x 32 x 1 MHz / 32768 Hz = 31250 = MCLK cycles for DCO to settle
            __delay_cycles(31250);

        }break;

        case 8: {

            clk_set = 8;    //CLK ist auf 8MHz gesetzt

            __bis_SR_register(SCG0);                  // Disable the FLL control loop
            UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
            UCSCTL1 = DCORSEL_5;                      // Select DCO range 16MHz operation
            UCSCTL2 = FLLD_1 + DCO_MP_8MHZ;                     // Set DCO Multiplier for 8MHz
                                                      // (N + 1) * FLLRef = Fdco
                                                      // (249 + 1) * 32768 = 8MHz
                                                      // Set FLL Div = fDCOCLK/2
            __bic_SR_register(SCG0);                  // Enable the FLL control loop

            // Worst-case settling time for the DCO when the DCO range bits have been
            // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
            // UG for optimization.
            // 32 x 32 x 8 MHz / 32,768 Hz = 250000 = MCLK cycles for DCO to settle
            __delay_cycles(250000);
        }break;
```

```
84          case 12: {
86              clk_set = 12;     //CLK ist auf 12MHz gesetzt
88              __bis_SR_register(SCG0);                  // Disable the FLL control loop
                UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
90              UCSCTL1 = DCORSEL_6;                      // Select DCO range 24MHz operation
                UCSCTL2 = FLLD_1 + DCO_MP_12MHZ;                  // Set DCO Multiplier for 12MHz
92                                                        // (N + 1) * FLLRef = Fdco
                                                          // (374 + 1) * 32768 = 12MHz
94                                                        // Set FLL Div = fDCOCLK/2
                __bic_SR_register(SCG0);                  // Enable the FLL control loop
96
                // Worst-case settling time for the DCO when the DCO range bits have been
98              // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
                // UG for optimization.
100             // 32 x 32 x 12 MHz / 32,768 Hz = 375000 = MCLK cycles for DCO to settle
                __delay_cycles(375000);
102         }break;
104         case 16: {
106             clk_set = 16;     //CLK ist auf 16MHz gesetzt
108
                __bis_SR_register(SCG0);                  // Disable the FLL control loop
110             UCSCTL0 = 0x0000;                         // Set lowest possible DCOx, MODx
                UCSCTL1 = DCORSEL_7;                      // Select DCO range 24MHz operation
112             UCSCTL2 = FLLD_1 + DCO_MP_16MHZ;                  // Set DCO Multiplier for 12MHz
                                                          // (N + 1) * FLLRef = Fdco
114                                                       // (487 + 1) * 32768 = 16MHz
                                                          // Set FLL Div = fDCOCLK/2
116             __bic_SR_register(SCG0);                  // Enable the FLL control loop
118             // Worst-case settling time for the DCO when the DCO range bits have been
                // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
120             // UG for optimization.
                // 32 x 32 x 12 MHz / 32,768 Hz = 375000 = MCLK cycles for DCO to settle
122             __delay_cycles(500000);
            }break;
124     }
}
```

code/ZSK3V04/clk.c

E.16 clk.h

```
1  /*****************************************************************************
   **    Description:    clk.h
3  **    Hardware:       BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
   **    Date:           24/04/2013
5  **    Last Update:    06/08/2014 - EM
   **    Author:         Nico Sassano
7  *****************************************************************************/
   #include "main.h"
9
   #ifndef CLK_H_
11 #define CLK_H_
13
15 #define DCO_MP_1MHZ 29    // added 06/08/2014 - EM
   #define DCO_MP_8MHZ 242   // added 06/08/2014 - EM
17 #define DCO_MP_12MHZ 365  // added 06/08/2014 - EM
   #define DCO_MP_16MHZ 487  // added 06/08/2014 - EM
19
   void clk_init_startup(void);
21 void clk_init_MHz(uint8_t mhz);

23 #endif /* CLK_H_ */
```

code/ZSK3V04/clk.h

E.17 delay.c

```
/*******************************************************************************
**     Description:     delay.h
**     Hardware:        BBATSEN ZS Klasse 3 v0.4 – 09/2014 – EM
**     Date:            09/04/2013
**     Last Update:     18/07/2013
**     Author:          Nico Sassano
*******************************************************************************/

#include "main.h"

extern uint8_t clk_set;

/*-----------------------------------------------------------------------------
  Prototypes of the private functions
-----------------------------------------------------------------------------*/

void delay(u16_t i);

/*-----------------------------------------------------------------------------
  Public functions
-----------------------------------------------------------------------------*/

void delay_ms(u16_t delay_ms)
{
  while (delay_ms > 0) {
      delay(DELAY_CYCLES_PER_MS);
      delay_ms--;
  }
}


void delay_100us_1MHz(u16_t delay_100us_1MHz)
{
    while (delay_100us_1MHz > 0) {
        delay(DELAY_CYCLES_PER_100US_1MHZ);
        delay_100us_1MHz--;
    }
}

void delay_20us_1MHz(u16_t delay_10us_1MHz)    //changed 01/10/2014 – EM
{
    while (delay_10us_1MHz > 0) {
        _NOP();
        delay_10us_1MHz--;
    }
}

void delay_10us_16MHz(u16_t delay_10us_16MHz)
{
    while (delay_10us_16MHz > 0) {
          delay(DELAY_CYCLES_PER_10US_16MHZ);
          delay_10us_16MHz--;
    }
}


void new_delay_ms(uint16_t time_ms) {
    uint16_t index = 0;

    switch(clk_set) {
        case 1: {
            while (time_ms > 0) {
                index = 70;
                while (index > 0) {
                    _NOP();
                    index--;
                }
                time_ms--;
            }

        }break;

        case 8: {
            while (time_ms > 0) {
                index = 560;
                while (index > 0) {
                    _NOP();
                    index--;
                }
                time_ms--;
            }
        }break;
```

```
83
        case 12: {
85          while (time_ms > 0) {
                index = 840;
87              while (index > 0) {
                    _NOP();
89                  index--;
                }
91              time_ms--;
            }
93      }break;

95      case 16: {
            while (time_ms > 0) {
97              index = 1120;
                while (index > 0) {
99                  _NOP();
                    index--;
101             }
                time_ms--;
103         }
        }break;
105 }
}
107
/*-------------------------------------------------------------------------
109   Private functions
-------------------------------------------------------------------------*/
111
void delay(u16_t i)
113 {
  while (i > 0) {
115     _NOP();
      i--;
117 }
}
```

code/ZSK3V04/delay.c

E.18 delay.h

```
/*******************************************************************************
2 **    Description:    delay.h
**    Hardware:       BBATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
4 **    Date:           09/04/2013
**    Last Update:    01/10/2014 - EM
6 **    Author:         Nico Sassano
*******************************************************************************/
8
#ifndef DELAY_H_
10 #define DELAY_H_

12 #include "main.h"

14 /*-------------------------------------------------------------------------
  Defines
16 -------------------------------------------------------------------------*/

18 #define DELAY_CYCLES_PER_MS             259   //changed 01/10/2014 - EM
#define DELAY_CYCLES_PER_100US_1MHZ     22  //changed 01/10/2014 - EM
20 #define DELAY_CYCLES_PER_10US_16MHZ     36  //changed 01/10/2014 - EM

22 /*-------------------------------------------------------------------------
  Public functions
24 -------------------------------------------------------------------------*/

26 void delay_ms(u16_t delay_ms);
void delay_100us_1MHz(u16_t delay_100us);
28 void delay_20us_1MHz(uint16_t);        //changed 01/10/2014 - EM
void delay_10us_16MHz(uint16_t);
30 void new_delay_ms(uint16_t);

32 #endif /* DELAY_H_ */
```

code/ZSK3V04/delay.h

E.19 hal_pmm.c

```
//*****************************************************************************
// Function Library for setting the PMM
//    File: hal_pmm.c
//
//    Texas Instruments
//
//    Version 1.2
//    11/24/09
//
//    V1.0  Initial Version
//    V1.1  Adjustment to UG
//    V1.2  Added return values
//
      ****************************************************************************//==========================================================


/* --COPYRIGHT--,BSD
* THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
* REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
* INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
* FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
* COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
* TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
* POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
* INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
* YOUR USE OF THE PROGRAM.
*
* IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
* CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
* THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
* OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
* EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
* REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
* OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
* USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
* AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
* YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
* (U.S.$500).
*
* Unless otherwise stated, the Program written and copyrighted
* by Texas Instruments is distributed as "freeware".  You may,
* only under TI's copyright in the Program, use and modify the
* Program without any charge or restriction.  You may
* distribute to third parties, provided that you transfer a
* copy of this license to the third party and the third party
* agrees to these terms by its first use of the Program. You
* must reproduce the copyright notice and any other legend of
* ownership on each copy or partial copy, of the Program.
*
* You acknowledge and agree that the Program contains
* copyrighted material, trade secrets and other TI proprietary
* information and is protected by copyright laws,
* international copyright treaties, and trade secret laws, as
* well as other intellectual property laws.  To protect TI's
* rights in the Program, you agree not to decompile, reverse
* engineer, disassemble or otherwise translate any object code
* versions of the Program to a human-readable form.  You agree
* that in no event will you alter, remove or destroy any
* copyright notice included in the Program.  TI reserves all
* rights not specifically granted under this license. Except
* as specifically provided herein, nothing in this agreement
* shall be construed as conferring by implication, estoppel,
* or otherwise, upon you, any license or other right under any
* TI patents, copyrights or trade secrets.
*
* You may not use the Program in non-TI devices.
* --/COPYRIGHT-- */

#include "cc430f5137.h"
#include "hal_pmm.h"

#define _HAL_PMM_DISABLE_SVML_
#define _HAL_PMM_DISABLE_SVSL_
#define _HAL_PMM_DISABLE_FULL_PERFORMANCE_


//*****************************************************************************
#ifdef _HAL_PMM_DISABLE_SVML_
#define _HAL_PMM_SVMLE SVMLE
#else
```

```
#define _HAL_PMM_SVMLE 0
#endif
#ifdef _HAL_PMM_DISABLE_SVSL_
#define _HAL_PMM_SVSLE SVSLE
#else
#define _HAL_PMM_SVSLE 0
#endif
#ifdef _HAL_PMM_DISABLE_FULL_PERFORMANCE_
#define _HAL_PMM_SVSFP SVSLFP
#else
#define _HAL_PMM_SVSFP 0
#endif
//*****************************************************************************//
// Set VCore
//*****************************************************************************//
unsigned int SetVCore (unsigned char level)
{
  unsigned int actlevel;
  unsigned int status = 0;
  level &= PMMCOREV_3;                      // Set Mask for Max. level
  actlevel = (PMMCTL0 & PMMCOREV_3);        // Get actual VCore

  while (((level != actlevel) && (status == 0)) || (level < actlevel))    // step by step increase or decrease
  {
    if (level > actlevel)
      status = SetVCoreUp(++actlevel);
    else
      status = SetVCoreDown(--actlevel);
  }
  return status;
}

//*****************************************************************************//
// Set VCore Up
//*****************************************************************************//
unsigned int SetVCoreUp (unsigned char level)
{
  unsigned int PMMRIE_backup, SVSMHCTL_backup;

  // Open PMM registers for write access
  PMMCTL0_H = 0xA5;

  // Disable dedicated Interrupts to prevent that needed flags will be cleared
  PMMRIE_backup = PMMRIE;
  PMMRIE &= ~(SVSMHDLYIE | SVSMLDLYIE | SVMLVLRIE | SVMHVLRIE | SVMHVLRPE);
  // Set SVM highside to new level and check if a VCore increase is possible
  SVSMHCTL_backup = SVSMHCTL;
  PMMIFG &= ~(SVMHIFG | SVSMHDLYIFG);
  SVSMHCTL = SVMHE | SVMHFP | (SVSMHRRL0 * level);
  // Wait until SVM highside is settled
  while ((PMMIFG & SVSMHDLYIFG) == 0);
  // Disable full-performance mode to save energy
  SVSMHCTL &= ~_HAL_PMM_SVSFP ;
  // Check if a VCore increase is possible
  if ((PMMIFG & SVMHIFG) == SVMHIFG){     //-> Vcc is to low for a Vcore increase
    // recover the previous settings
    PMMIFG &= ~SVSMHDLYIFG;
    SVSMHCTL = SVSMHCTL_backup;
    // Wait until SVM highside is settled
    while ((PMMIFG & SVSMHDLYIFG) == 0);
    // Clear all Flags
    PMMIFG &= ~(SVMHVLRIFG | SVMHIFG | SVSMHDLYIFG | SVMLVLRIFG | SVMLIFG | SVSMLDLYIFG);
    // backup PMM-Interrupt-Register
    PMMRIE = PMMRIE_backup;

    // Lock PMM registers for write access
    PMMCTL0_H = 0x00;
    return PMM_STATUS_ERROR;                        // return: voltage not set
  }
  // Set also SVS highside to new level     //-> Vcc is high enough for a Vcore increase
  SVSMHCTL |= SVSHE | (SVSHRVL0 * level);
  // Set SVM low side to new level
  SVSMLCTL = SVMLE | SVMLFP | (SVSMLRRL0 * level);
  // Wait until SVM low side is settled
  while ((PMMIFG & SVSMLDLYIFG) == 0);
  // Clear already set flags
  PMMIFG &= ~(SVMLVLRIFG | SVMLIFG);
  // Set VCore to new level
  PMMCTL0_L = PMMCOREV0 * level;
  // Wait until new level reached
  if (PMMIFG & SVMLIFG)
  while ((PMMIFG & SVMLVLRIFG) == 0);
  // Set also SVS/SVM low side to new level
  PMMIFG &= ~SVSMLDLYIFG;
  SVSMLCTL |= SVSLE | (SVSLRVL0 * level);
```

```
166     // wait for lowside delay flags
        while ((PMMIFG & SVSMLDLYIFG) == 0);
168
    // Disable SVS/SVM Low
170 // Disable full-performance mode to save energy
        SVSMLCTL &= ~(_HAL_PMM_DISABLE_SVSL_+_HAL_PMM_DISABLE_SVML_+_HAL_PMM_SVSFP );
172
        // Clear all Flags
174     PMMIFG &= ~(SVMHVLRIFG | SVMHIFG | SVSMHDLYIFG | SVMLVLRIFG | SVMLIFG | SVSMLDLYIFG);
        // backup PMM-Interrupt-Register
176     PMMRIE = PMMRIE_backup;

178     // Lock PMM registers for write access
        PMMCTL0_H = 0x00;
180     return PMM_STATUS_OK;                                   // return: OK
    }
182
    //------------------------------------------------------------------------------//
184 // Set VCore down (Independent from the enabled Interrupts in PMMRIE)
    //------------------------------------------------------------------------------//
186 unsigned int SetVCoreDown (unsigned char level)
    {
188   unsigned int PMMRIE_backup;

190   // Open PMM registers for write access
      PMMCTL0_H = 0xA5;
192
      // Disable dedicated Interrupts to prevent that needed flags will be cleared
194   PMMRIE_backup = PMMRIE;
      PMMRIE &= ~(SVSMHDLYIE | SVSMLDLYIE | SVMLVLRIE | SVMHVLRIE | SVMHVLRPE);
196
      // Set SVM high side and SVM low side to new level
198   PMMIFG &= ~(SVMHIFG | SVSMHDLYIFG | SVMLIFG | SVSMLDLYIFG);
      SVSMHCTL = SVMHE | SVMHFP | (SVSMHRRL0 * level);
200   SVSMLCTL = SVMLE | SVMLFP | (SVSMLRRL0 * level);
      // Wait until SVM high side and SVM low side is settled
202   while ((PMMIFG & SVSMHDLYIFG) == 0 || (PMMIFG & SVSMLDLYIFG) == 0);

204   // Set VCore to new level
      PMMCTL0_L = PMMCOREV0 * level;
206
      // Set also SVS highside and SVS low side to new level
208   PMMIFG &= ~(SVSHIFG | SVSMHDLYIFG | SVSLIFG | SVSMLDLYIFG);
      SVSMHCTL |= SVSHE | SVSHFP | (SVSHRVL0 * level);
210   SVSMLCTL |= SVSLE | SVSLFP | (SVSLRVL0 * level);
      // Wait until SVS high side and SVS low side is settled
212   while ((PMMIFG & SVSMHDLYIFG) == 0 || (PMMIFG & SVSMLDLYIFG) == 0);
      // Disable full-performance mode to save energy
214   SVSMHCTL &= ~_HAL_PMM_SVSFP;
    // Disable SVS/SVM Low
216 // Disable full-performance mode to save energy
      SVSMLCTL &= ~(_HAL_PMM_DISABLE_SVSL_+_HAL_PMM_DISABLE_SVML_+_HAL_PMM_SVSFP );
218
      // Clear all Flags
220   PMMIFG &= ~(SVMHVLRIFG | SVMHIFG | SVSMHDLYIFG | SVMLVLRIFG | SVMLIFG | SVSMLDLYIFG);
      // backup PMM-Interrupt-Register
222   PMMRIE = PMMRIE_backup;
      // Lock PMM registers for write access
224   PMMCTL0_H = 0x00;

226   if ((PMMIFG & SVMHIFG) == SVMHIFG)
        return PMM_STATUS_ERROR;            // Highside is still to low for the adjusted VCore Level
228   else return PMM_STATUS_OK;            // Return: OK
    }
```

code/ZSK3V04/hal_pmm.c

E.20 hal_pmm.h

```
//*****************************************************************************
// Function Library for setting the PMM
//    File: hal_pmm.h
//
//    Texas Instruments
//
//    Version 1.2
//    10/17/09
//
//    V1.0  Initial Version
//    V1.1  Adjustment to UG
//    V1.2  Added return values
//
      ***************************************************************************////===========================================================================



#ifndef __PMM
#define __PMM

#define PMM_STATUS_OK     0
#define PMM_STATUS_ERROR  1

//------------------------------------------------------------------------
/**
  * Set the VCore to a new level if it is possible and return a
  * error - value.
  *
  * \param      level       PMM level ID
  * \return int     1: error / 0: done
  */
unsigned int SetVCore (unsigned char level);

//------------------------------------------------------------------------
/**
  * Set the VCore to a higher level, if it is possible.
  * Return a 1 if voltage at highside (Vcc) is to low
  * for the selected Level (level).
  *
  * \param      level       PMM level ID
  * \return int     1: error / 0: done
  */
unsigned int SetVCoreUp (unsigned char level);

//------------------------------------------------------------------------
/**
  * Set the VCore to a lower level.
  * Return a 1 if voltage at highside (Vcc) is still to low
  * for the selected Level (level).
  *
  * \param      level       PMM level ID
  * \return int     1: done with error / 0: done without error
  */
unsigned int SetVCoreDown (unsigned char level);

#endif /* __PMM */
```

code/ZSK3V04/hal_pmm.h

E.21 i2c.c

```
/*******************************************************************************
**    Description:    I2C Interface
**    Hardware:       BBATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:           05/03/2013
**    Last Update:    31/07/14 - EM
**    Author:         Nico Sassano
*******************************************************************************/

#include "main.h"


#define UCB0BR_BIT_CLK 100.0     // UCB0 Bit Clock [kHz]
                                 // UCB0 Baud Rate Control 0 Setting
/*-------------------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------------------*/

void i2c_init(void)
{
  I2C_SDA_PxSEL |= I2C_SDA_PIN; //changed 31/07/14 - EM
  I2C_SCL_PxSEL |= I2C_SCL_PIN; //Set I2C Pins to I2C CommunicationM


    UCB0CTL1 |= UCSWRST;         // Reset the USCI logic


    /** control register 0 *****************************************************
    * Set UCB0CTL  -> Master mode | I2C Mode | Synchronous mode
    *              -> Own address is a 7-bit address, Address slave
    *                 with 7-bit address
    ******************************************************************************/
    UCB0CTL0 = (UCMST | UCMODE1 | UCMODE0 | UCSYNC);

    /** control register 1 *****************************************************
    * Set UCB0CTL1 -> clock source SMCLK | Software reset enable
    *              -> Receiver | Acknowledge normally | No STOP generated |
    *                 Do not generate START condition
    ******************************************************************************/
    UCB0CTL1 = (UCSSEL1 | UCSWRST);


    /** bit rate control register 0/1 ******************************************
    * -> Datasheet S. 476 <-
    *
    * Baud rate = 100kbps
    *
    * -> f_brclk = SMCLK = 0.5MHz
    * -> UCBRX = 5
    * f_bitclk = f_brclk / UCBRX = 100kHz,
    ******************************************************************************/
    //UCB0BR0 = ((uint16_t) SMCLK / UCB0BR_BIT_CLK); // low byte, UCB0RX = (UCB0R0 + UCB0R1 x 128)
    UCB0BR0 = 0x0A;
    UCB0BR1 = 0x00; // high byte

    /** Own Address Register ***************************************************
    * I2C own 7-bit address = 0x7B (123)
    ******************************************************************************/
    UCB0I2COA |= 0x007B;

    /** Interrupt Enable Register **********************************************
    * Interrupt disabled for Not-acknowledge, Start/Stop condition and
    * Arbitration lost
    ******************************************************************************/
    UCB0I2CIE = 0x00;

}

void i2c_write(uint8_t slave_address, uint8_t data_length, uint8_t *data)
{
    uint8_t i;

    UCB0CTL1 |= UCTR;                                   // set transmitter-mode
    UCB0I2CSA = slave_address;                          // set slave address
    UCB0CTL1 &= ~UCSWRST;                               // unset SWRESET
    UCB0CTL1 |= UCTXSTT;                                // send START-CON
    UCB0TXBUF = data[0];                                // then write data

    for(i = 1; i < data_length; i++) {
        while(1) { // wait until ...
            /*if((UCB0STAT & UCNACKIFG) == UCNACKIFG)      // NACK from slave
            {
                UCB0CTL1 |= UCTXSTP;                    // then send STOP-CON
```

```
                UCB0STAT &= ~UCNACKIFG;                 // reset flag
84              UCB0CTL1 |= UCSWRST;                    // set SWRESET
                break;
86          }*/

88          if((IFG2 & UCB0TXIFG) == UCB0TXIFG)         // data / start-con was send
            {
90              UCB0TXBUF = data[i];                    // then write data
                break;
92          }
        }
94  }

96  while((IFG2 & UCB0TXIFG) != UCB0TXIFG);             // wait until data/start-con was send

98  UCB0CTL1 |= UCTXSTP;                                // send STOP-COND
    while((UCB0CTL1 & UCTXSTP) == UCTXSTP);             // wait until STOP-con was send
100
    UCB0CTL1 |= UCSWRST;                                // set SWRESET
102 }

104 void i2c_read(uint8_t slave_address, uint8_t data_length, uint8_t *data)
    {
106     uint8_t i;

108     UCB0CTL1 &= ~UCTR;          // reset transmitter-mode
        UCB0I2CSA = slave_address;  // set slave address of sensor
110     UCB0CTL1 &= ~UCSWRST;       // unset SWRESET
        UCB0CTL1 |= UCTXSTT;        // send START-CON
112
        if(data_length > 1)
114     {
            for(i = 0; i < data_length; i++)
116         {
                while(1)
118             {
                    /*****************************************************************
120                 * IFG2 = Interrupt Flag Register 2
                    *****************************************************************/
122                 if((IFG2 & UCB0RXIFG) == UCB0RXIFG) // data / start-con was send
                    {
124                     data[i] = UCB0RXBUF;            // readout data
                        if(i == data_length-2)          // if next byte will be the last one
126                         UCB0CTL1 |= UCTXSTP;        // send STOP-CON after next receive

128                     break;
                    }
130             }
            }
132     }
        else
134     {
            while((UCB0CTL1 & UCTXSTT) == UCTXSTT); // wait for acknowledge of slave,
136         UCB0CTL1 |= UCTXSTP; // then send STOP-COND immediately

138         data[0] = UCB0RXBUF; // readout data
        }
140
        while((UCB0CTL1 & UCTXSTP) == UCTXSTP); // wait until STOP-con was send
142     UCB0CTL1 |= UCSWRST; // set SWRESET
    }
```

code/ZSK3V04/i2c.c

E.22 i2c.h

```
/*******************************************************************************
**    Description:      I2C Interface
**    Hardware:         BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:             05/03/2013
**    Last Update:      31/07/2014
**    Author:           Nico Sassano
*******************************************************************************/
#ifndef I2C_H_
#define I2C_H_

#include "main.h"

/*******************************************************************************
*  Defines
*******************************************************************************/
 /* Allready defined by LED.h  // commented out 31/07/2014 - EM

#define LED_1_PxDIR      (P5DIR)
#define LED_1_PxOUT      (P5OUT)
#define LED_1_PIN        (BIT4)

#define LED_2_PxDIR      (P5DIR)
#define LED_2_PxOUT      (P5OUT)
#define LED_2_PIN        (BIT3)

#define LED_3_PxDIR      (P5DIR)
#define LED_3_PxOUT      (P5OUT)
#define LED_3_PIN        (BIT2)*/

/* Bending Defines for CC430 */

#define I2C_SDA_PxSEL    (P1SEL) // added 31/07/2014 - EM
#define I2C_SDA_PIN    (BIT3)   // added 31/07/2014 - EM
#define I2C_SCL_PxSEL    (P1SEL) // added 31/07/2014 - EM
#define I2C_SCL_PIN    (BIT2)   // added 31/07/2014 - EM

#define UCB0TXIFG    UCTXIFG    // added 31/07/2014 - EM
#define UCB0RXIFG    UCRXIFG    // added 31/07/2014 - EM
#define UCB0I2CIE    UCB0IE     // added 31/07/2014 - EM
#define IFG2     UCB0IFG    // added 31/07/2014 - EM

/*******************************************************************************
*  Prototyping declaration
*******************************************************************************/
unsigned char *PTxData;                     // Pointer to TX data
unsigned char TXByteCtr;           // added 31/07/2014 - EM

void i2c_init(void);
void i2c_write(uint8_t , uint8_t , uint8_t *);
void i2c_read(uint8_t , uint8_t , uint8_t *);

#endif /* I2C_H_ */
```

code/ZSK3V04/i2c.h

E.23 init.c

```
/*******************************************************************************
**    Description:    init.c
**    Hardware:       BBATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:           09/04/2013
**    Last Update:    26/09/2014  - EM
**    Author:         Nico Sassano
*******************************************************************************/

#include "main.h"

extern volatile uint8_t brate_start;
extern volatile uint8_t brate;

extern volatile uint16_t upper_alarm_temp;
extern volatile uint16_t lower_alarm_temp;

extern volatile uint8_t irq_alert;        // TMP102 Alarm


void init(void) {

    TPS61201_PxDIR |= TPS61201_PIN;
    TPS61201_ENABLE;

    // LEDs initialisieren
    led_init();
    i2c_init();
    // Test Port als Ausgang konfigurieren
    TEST_PORT_PxDIR |= TEST_PORT_PIN;

    // Alles LED leuchten 3 Sekunden beim start
    led_on(LED_ALL);
    delay_ms(300);
    led_off(LED_ALL);

    brate = brate_start;

    // RF-switch
    adg918_init();
    adg918_wakeup();    // Connect the loop antenna with the wakeup circuit

    // Balancing unit
    balancing_init();
    balancing_off();    // Balancing unit off

    /***********************************
    ** Temperatursensor konfigurieren
    ***********************************/
    temp_sensor_init();
    temp_sensor_config_reg();
    temp_sensor_set_alert(lower_alarm_temp, upper_alarm_temp);
    IRQ_ALERT_SET_LOW;       // normale Temperaur setzen
    temp_sensor_get_contr_reg();

    /***********************************
    ** Timer konfigurieren
    ***********************************/
    timer_a_init();
    timer_b_init();

    // CC1101 transceiver
   //  tx_carrier();
    cc430_init();          // changed 26/09/2014  - EM
    cc430_power_up_reset();   // changed 26/09/2014 - EM

A

    cc430_sleep();              // Power down state  // changed 26/09/2014 - EM

    // Configure packet received interrupt
    CC430_END_OF_PKT_CLEAR_IRQ;    // changed 26/09/2014 - EM
    CC430_END_OF_PKT_IRQ_DISABLE; // changed 26/09/2014 - EM

    adc24_volt_init(ADC24_500_SPS); // added 26/09/2014 - EM

    // LF wakeup receiver
    as3930_init();
    as3930_preset_default();     // Reset
    as3930_config_no_pattern(); // Wakeup upon LF carrier detection
    as3930_clear_wakeup();       // Clear wakeup

    _EINT();
```

```
84      /*** Soll einschlafen ***/
        // Configure Wake interrupt
86      AS3930_WAKE_CLEAR_IRQ;
        AS3930_WAKE_IRQ_ENABLE;
88
90      // Global interrupt enable

92
        // Enter Low Power Mode 4 (LPM4) with all clocks disabled
94      // (wait for wakeup)

96      AS3930_WAKE_IRQ_ENABLE;
        AS3930_WAKE_CLEAR_IRQ;
98      ENTER_LPM4;
        /*******************************/
100

102

}
104
void init_for_sleep(void) {
106
        // RF-switch
108     adg918_init();
        adg918_wakeup();     // Connect the loop antenna with the wakeup circuit
110
        balancing_off();     // Balancing unit off
112

114     // CC1101 transceiver
        cc430_init();       // changed 26/09/2014 - EM
116     cc430_power_up_reset(); // changed 26/09/2014 - EM
        cc430_sleep();              // Power down state  // changed 26/09/2014 - EM
118
        // Configure packet received interrupt
120     CC430_END_OF_PKT_CLEAR_IRQ;    // changed 26/09/2014 - EM
        CC430_END_OF_PKT_IRQ_DISABLE; // changed 26/09/2014 - EM
122
        // LF wakeup receiver
124     as3930_init();
        as3930_preset_default();     // Reset
126     as3930_config_no_pattern(); // Wakeup upon LF carrier detection
        as3930_clear_wakeup();       // Clear wakeup
128
        // Configure Wake interrupt
130     AS3930_WAKE_CLEAR_IRQ;
        AS3930_WAKE_IRQ_ENABLE;
132
        // Global interrupt enable
134
        _EINT();
136     // Enter Low Power Mode 4 (LPM4) with all clocks disabled
        // (wait for wakeup)
138
        //     AS3930_WAKE_IRQ_DISABLE;
140     //     AS3930_WAKE_CLEAR_IRQ;
        //     tx_carrier();
142     //ENTER_LPM4;

144 }
```

code/ZSK3V04/init.c

E.24 init.h

```
/*****************************************************************************
**     Description:     init.h
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            09/04/2013
**     Last Update:     18/07/2013
**     Author:          Nico Sassano
*****************************************************************************/

#ifndef INIT_H_
#define INIT_H_

void init(void);
void init_for_sleep(void);

#endif /* INIT_H_ */
```

code/ZSK3V04/init.h

E.25 isr.c

```
/*******************************************************************************
**     Description:     isr.c
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            08/10/2014
**     Last Update:     08/10/2014
**     Author:          Eike Mense
*******************************************************************************/

#include "main.h"


extern volatile uint8_t wakeup_state;

//extern volatile uint8_t state;
extern volatile state_t state;

/***********************************************
** CLK
***********************************************/
extern volatile uint8_t clk_set;

/***********************************************
** RX
***********************************************/
extern volatile uint8_t rx_command;
extern volatile uint8_t rx_data_length;

/***********************************************
** Balancing
***********************************************/
extern volatile uint16_t balancing_volt;  // Zielwert der Balancierung
extern volatile uint16_t balancing_time;
extern volatile uint8_t balanc_state;
extern volatile uint8_t temp_state;
extern volatile uint16_t upper_balanc_temp;
extern volatile uint16_t lower_balanc_temp;
extern volatile uint16_t upper_alarm_temp;
extern volatile uint16_t lower_alarm_temp;

/***********************************************
** Data buffer
***********************************************/
uint16_t config_value;
extern volatile uint16_t sample_burst_buf[BURST_VALUES];
extern volatile uint16_t sample_buf_volt;                    // The latest cell voltage sample
//extern volatile uint16_t sample_buf_temp;

/***********************************************
** Kalibrierung
***********************************************/
extern volatile uint8_t cali_pos_offset_tmp102;
extern volatile uint8_t cali_neg_offset_tmp102;
extern volatile uint8_t cali_pos_offset_msp430;
extern volatile uint8_t cali_neg_offset_msp430;
extern volatile uint8_t cali_pos_offset_adc;
extern volatile uint8_t cali_neg_offset_adc;

/***********************************************
** Burst Mode
***********************************************/
extern volatile uint8_t burst_freq;             // Burst Frequenz
extern volatile uint16_t burst_values;           // Anzahl der erwarteten  Burst Werte
extern volatile uint16_t burst_counter;         // Counter der Burst-Werte
extern volatile uint8_t frame_number;
extern volatile uint8_t burst_frame_lenght;    // 50 -> 25 Werten
extern volatile uint8_t burst_error;
extern volatile uint8_t burst_error_flag;

extern volatile uint8_t adc_select;

/***********************************************
** Interruptflags
***********************************************/
extern volatile uint8_t irq_mode;               // PORT1 IRQ Status
extern volatile uint8_t irq_send_done_flag;     // CC1101 senden
extern volatile uint8_t irq_alert;              // TMP102 Alarm
extern volatile uint8_t irq_timera;             // Status TimerA
extern volatile uint8_t irq_timerb;             // Status TimerB


/***********************************************
** Communication states
```

```
83  ********************************************/
    extern volatile uint8_t command_recived;
85  extern volatile uint8_t wakeup_state;
    extern volatile uint8_t brate_start;
87  extern volatile uint8_t brate;

89  volatile uint16_t test = 0;

91

93  /**************************************************************************************************
    **
95  **
    **
97  **************************************************************************************************/
    #pragma vector=CC1101_VECTOR
99  __interrupt void CC1101_ISR(void)
    {
101   switch(__even_in_range(RF1AIV,32))        // Prioritizing Radio Core Interrupt
      {
103     case  0: break;                         // No RF core interrupt pending
        case  2: break;                         // RFIFG0
105     case  4: break;                         // RFIFG1
        case  6: break;              // RFIFG2
107     case  8: break;                         // RFIFG3
        case 10: break;                         // RFIFG4
109     case 12: break;                         // RFIFG5
        case 14: break;                         // RFIFG6
111     case 16: break;                         // RFIFG7
        case 18: break;                         // RFIFG8
113     case 20: // End of Packet IRQ Recieved

115       if(IRQ_IS_RX) {
            /*****************************************************
117         ** IRQ Packet empfangen
            *****************************************************/
119           //TIMER_A_0_CM_IRQ_DISABLE;
              _DINT();                  // Globale Interrups ausschalten
121
              volatile uint8_t bytes_in_fifo = 0;
123           CC430_END_OF_PKT_IRQ_DISABLE;
             // CC430_END_OF_PKT_CLEAR_IRQ;     // Seems to Auto-Clear on Read from RX-Fifo TODO : verifizieren
125
              rx_command = COMMAND_DOWNLINK_UNKWOWN;
127           command_recived = 1;
              bytes_in_fifo = cc430_get_rxbytes();
129           if(bytes_in_fifo >= 2)
                bytes_in_fifo-=2;    //Non-Zero Values from get_rxbytes are to great by 2 TODO Ursache ermitteln
131
              uint8_t rxbuf[64];
133           // Check whether received data is valid
              if ((bytes_in_fifo == HEADER_LENGTH + rx_data_length)) {
135


137
                  cc430_read_rx_fifo(rxbuf, HEADER_LENGTH + rx_data_length);
139               rx_data_length = 0; // Datenlänge wieder zurücksetzen

141               if (rxbuf[0] == BROADCAST || rxbuf[0] == ADDRESS_THIS_SENSOR) { // TODO Adresse müsste autmotisch geprüft werden.

143                   led_on(LED_RX);           // Empfangs LED an

145                   rx_command = rxbuf[1];   // Empfangenes Kommando

147                   switch(rx_command) {
                          /*********************************************
149                       ** Dekodierung des WAKEUP Kommandos
                          ** Paketzusammenstellung:
151                       ** ( Adr. ZS | Kommando | 0x00 | 0x00 )
                          *********************************************/
153                       case COMMAND_WAKEUP: {
                              state = S_WAKEUP_RX;
155                           wakeup_state = 1;
                          }break;
157
                          /*********************************************
159                       ** Dekodierung des WAKEUP_DONE Kommandos
                          ** Paketzusammenstellung:
161                       ** ( Adr. ZS | Kommando | 0x00 | 0x00 )
                          *********************************************/
163                       case COMMAND_WAKEUP_DONE: {
                              state = S_WAKEUP_DONE;
165                           if(!wakeup_state)
                                  rx_command = COMMAND_DOWNLINK_SLEEP;
167
```

```
                    }break;
169
                    /***********************************************
171                 ** Dekodierung des CONFIG_SET Kommandos
                    ** Paketzusammenstellung:
173                 ** ( Adr. ZS | Kommando | Anzahl | 0x00 )
                    ***********************************************/
175                 case COMMAND_DOWNLINK_CONFIG_SET: {
                        rx_data_length = rxbuf[2];  // Länge der nächsten Datensendung
177
                    }break;
179
                    /***********************************************
181                 ** Dekodierung des CONFIG Kommandos
                    ** Paketzusammenstellung:
183                 **    0    1     2    3
                    ** ( Adr. ZS | Kommando | 0x00 | 0x00 )
185                 **     0     1    2     3    4     5    6     7    8     9       10
                    ** ( Brate | oberer Bal. Wert | unterer Bal. Wert | oberer Alarmwert | unterer Alarmwert | Burst Framelänge |
        ADC Auswahl)
187                 ***********************************************/
                    case COMMAND_DOWNLINK_CONFIG: {
189                     rx_data_length  = 0; // Datenlänge zurücksetzen
                        uint16_t msb    = 0;
191                     uint16_t lsb    = 0;

193                     // Übertragungsrate
                        brate               = rxbuf[HEADER_LENGTH + 0];
195
                        // oberer Balancierungswert
197                     msb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 1] & 0x0F)) << 8) & 0xFF00;
                        lsb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 2] & 0xFF)) << 0) & 0x00FF;
199                     upper_balanc_temp   = msb | lsb;
                        msb = 0;
201                     lsb = 0;

203                     // unterer Balancierungswert
                        msb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 3] & 0x0F)) << 8) & 0xFF00;
205                     lsb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 4] & 0xFF)) << 0) & 0x00FF;
                        lower_balanc_temp   = msb | lsb;
207                     msb = 0;
                        lsb = 0;
209
                        // oberer Alarmwert
211                     msb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 5] & 0x0F)) << 8) & 0xFF00;
                        lsb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 6] & 0xFF)) << 0) & 0x00FF;
213                     upper_alarm_temp    = msb | lsb;
                        msb = 0;
215                     lsb = 0;

217                     // unterer Alarmwert
                        msb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 7] & 0x0F)) << 8) & 0xFF00;
219                     lsb                 = (((uint16_t)(rxbuf[HEADER_LENGTH + 8] & 0xFF)) << 0) & 0x00FF;
                        lower_alarm_temp    = msb | lsb;
221
                        // Burst Framelänge
223                     burst_frame_lenght  = rxbuf[HEADER_LENGTH + 9];

225                     adc_select      = rxbuf[HEADER_LENGTH + 10];

227
                    }break;
229
                    /***********************************************
231                 ** Dekodierung des Balancierungsheader
                    ** Paketzusammenstellung:
233                 ** ( Adr. ZS | Kommando | VOLT_MSB | VOLT_LSB )
                    ***********************************************/
235                 case COMMAND_DOWNLINK_BALANCING_ON: {
                        uint16_t volt_msb = 0;
237                     uint16_t volt_lsb = 0;

239                     volt_msb = (((uint16_t)(rxbuf[3] & 0x0F)) << 8) & 0xFF00;
                        volt_lsb = (((uint16_t)(rxbuf[4] & 0xFF)) << 0) & 0x00FF;
241
                        balancing_volt = volt_msb | volt_lsb;
243                     balancing_volt = 2620;  //TEST

245                 }break;

247                 /***********************************************
                    ** Dekodierung des Burstheader
249                 ** Paketzusammenstellung:
                    ** ( Adr. ZS | Kommando | Burst Freq | Anzahl Werte )
251                 ***********************************************/
```

```
                case COMMAND_DOWNLINK_BURST_MODE: {
253                 burst_freq          = rxbuf[2];
                    uint8_t burst_value = rxbuf[3];
255
                   switch(burst_value) {
257                    case BURST_VALUES_50:
                           burst_values = 50;
259                    break;

261                    case BURST_VALUES_100:
                           burst_values = 100;
263                    break;

265                    case BURST_VALUES_150:
                           burst_values = 150;
267                    break;

269                    case BURST_VALUES_200:
                           burst_values = 200;
271                    break;

273                    case BURST_VALUES_250:
                           burst_values = 250;
275                    break;

277                    case BURST_VALUES_300:
                           burst_values = 300;
279                    break;

281                    case BURST_VALUES_350:
                           burst_values = 350;
283                    break;

285                    case BURST_VALUES_400:
                           burst_values = 400;
287                    break;

289                    case BURST_VALUES_450:
                           burst_values = 450;
291                    break;

293                    case BURST_VALUES_500:
                           burst_values = 500;
295                    break;

297                    case BURST_VALUES_550:
                           burst_values = 550;
299                    break;

301                    case BURST_VALUES_600:
                           burst_values = 600;
303                    break;

305                    case BURST_VALUES_650:
                           burst_values = 650;
307                    break;

309                    case BURST_VALUES_700:
                           burst_values = 700;
311                    break;

313                    case BURST_VALUES_750:
                           burst_values = 750;
315                    break;

317                    case BURST_VALUES_800:
                           burst_values = 800;
319                    break;

321                    case BURST_VALUES_850:
                           burst_values = 850;
323                    break;

325                    case BURST_VALUES_900:
                           burst_values = 900;
327                    break;

329                    case BURST_VALUES_1000:
                         burst_values = 1000;
331                    break;

333                    case BURST_VALUES_1500:
                         burst_values = 1500;
335                    break;
```

```
337                             case BURST_VALUES_1900:
                                  burst_values = 1900;
339                               break;

341                         }

343                         /* TODO KENNZEICHEN
                             * NEU von EIKE für ADC24
345                          */
                            if(adc_select == ADC24_EXT_SELECT) {
347                 switch(burst_freq) {

349                   case BURST_FREQ_10000HZ: {
                            // Unable for direct meassurment
351                   }break;

353                   case BURST_FREQ_8000HZ: {
                        adc24_volt_cont_init(ADC24_8000_SPS);   // Unable for direct meassurment
355                   }break;

357                   case BURST_FREQ_6000HZ: {
                        adc24_volt_cont_init(ADC24_8000_SPS);   // Unable for direct meassurment
359                   }break;

361                   case BURST_FREQ_4000HZ: {
                        adc24_volt_cont_init(ADC24_4000_SPS);   // Unable for direct meassurment
363                   }break;

365                   case BURST_FREQ_2000HZ: {
                        adc24_volt_cont_init(ADC24_2000_SPS); // Unable for direct meassurment
367
                     }break;
369
                     case BURST_FREQ_1000HZ: {
371                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
373
                     case BURST_FREQ_950HZ: {
375                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
377
                     case BURST_FREQ_900HZ: {
379                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
381
                     case BURST_FREQ_850HZ: {
383                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
385
                     case BURST_FREQ_800HZ: {
387                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
389
                     case BURST_FREQ_750HZ: {
391                    adc24_volt_init(ADC24_8000_SPS);
                     }break;
393
                     case BURST_FREQ_700HZ: {
395                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
397
                     case BURST_FREQ_650HZ: {
399                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
401
                     case BURST_FREQ_600HZ: {
403                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
405
                     case BURST_FREQ_550HZ: {
407                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
409
                     case BURST_FREQ_500HZ: {
411                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
413
                     case BURST_FREQ_450HZ: {
415                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
417
                     case BURST_FREQ_400HZ: {
419                    adc24_volt_init(ADC24_4000_SPS);
                     }break;
421
```

```
                case BURST_FREQ_350HZ: {
423               adc24_volt_init(ADC24_2000_SPS);
                }break;
425
                case BURST_FREQ_300HZ: {
427               adc24_volt_init(ADC24_2000_SPS);
                }break;
429
                case BURST_FREQ_250HZ: {
431               adc24_volt_init(ADC24_2000_SPS);
                }break;
433
                case BURST_FREQ_200HZ: {
435               adc24_volt_init(ADC24_1000_SPS);
                }break;
437
                case BURST_FREQ_150HZ: {
439               adc24_volt_init(ADC24_1000_SPS);
                }break;
441
                case BURST_FREQ_100HZ: {
443               adc24_volt_init(ADC24_1000_SPS);
                }break;
445
                case BURST_FREQ_50HZ: {
447               adc24_volt_init(ADC24_250_SPS);
                }break;
449                         }
                        }
451
                    }
453
                    /************************************************
                    ** Dekodierung des Burstheader
                    ** Paketzusammenstellung:
457                 ** ( Adr. ZS | Kommando | Frame Nummer | 0x00 )
                    ************************************************/
459                 case COMMAND_DOWNLINK_BURST_DATA_RX: {
                        frame_number       = rxbuf[2];
461
                    }break;
463
                    /************************************************
465                 ** Dekodierung des Kalibrierungsheader: TMP102
                    ** Paketzusammenstellung:
467                 ** ( Adr. ZS | Kommando | Pos. Offset | Neg. Offset )
                    ************************************************/
469                 case COMMAND_DOWNLINK_CALIBRATION_TMP102: {
                        cali_pos_offset_tmp102 = rxbuf[2];
471                     cali_neg_offset_tmp102 = rxbuf[3];
                    }break;
473
                    /************************************************
475                 ** Dekodierung des Kalibrierungsheader: MSP430
                    ** Paketzusammenstellung:
477                 ** ( Adr. ZS | Kommando | Pos. Offset | Neg. Offset )
                    ************************************************/
479                 case COMMAND_DOWNLINK_CALIBRATION_MSP430: {
                        cali_pos_offset_msp430 = rxbuf[2];
481                     cali_neg_offset_msp430 = rxbuf[3];
                    }break;
483
                    /************************************************
485                 ** Dekodierung des Kalibrierungsheader: ADC
                    ** Paketzusammenstellung:
487                 ** ( Adr. ZS | Kommando | Pos. Offset | Neg. Offset )
                    ************************************************/
489                 case COMMAND_DOWNLINK_CALIBRATION_ADC: {
                        cali_pos_offset_adc = rxbuf[2];
491                     cali_neg_offset_adc = rxbuf[3];
                    }break;
493
495                 default: break;
                }
497
            }
499         led_off(LED_RX);
        } else {
501       cc430_read_rx_fifo(rxbuf, bytes_in_fifo); //TODO CHeck. Dummy read to reset Rx Flag
        }
503

505     //TIMER_A_0_CM_IRQ_ENABLE;
        _EINT();                // Globale Interrups einschalten
```

```
507
        } else if(IRQ_IS_TX) {
509       CC430_END_OF_PKT_IRQ_DISABLE;
            //CC430_END_OF_PKT_CLEAR_IRQ; Seems to Auto-Clear  TODO : Herausfinden, bei was Auto-Clear
511         irq_send_done_flag = 1;
        }
513      break;                          // RFIFG9
    case 22: break;                          // RFIFG10
515 case 24: break;                          // RFIFG11
    case 26: break;                          // RFIFG12
517 case 28: break;                          // RFIFG13
    case 30: break;                          // RFIFG14
519 case 32: break;                          // RFIFG15
  }
521 }
#pragma vector=PORT1_VECTOR
523 __interrupt void PORT1_ISR(void)
{
525   switch(__even_in_range(P1IV,16))
  {
527   case 0: break;
    case 2: break;                           // Vector P1IFG.0
529 case 4: // Temp IRQ Empfangeln

531      if(IRQ_ALERT_IS_LOW) {          // Anfrage Alarm auslösen

533          TMP102_ALERT_IRQ_DISABLE;   // Interrupt stoppen
             TMP102_ALERT_CLEAR_IRQ;
535
             //led_off(LED_AWAKE);
537          IRQ_ALERT_SET_HIGH;         // Alarm setzen

539          TMP102_SET_IRQ_RISING_EDGE; // IRQ auf steigende Flnke setzen
             TMP102_ALERT_CLEAR_IRQ;
541          TMP102_ALERT_IRQ_ENABLE;

543      }else if(IRQ_ALERT_IS_HIGH) {   // Anfrage Ararm zurücksetzen

545          TMP102_ALERT_IRQ_DISABLE;   // Interrupt stoppen
             TMP102_ALERT_CLEAR_IRQ;
547
             //led_on(LED_AWAKE);
549          IRQ_ALERT_SET_LOW;          // Alarm löschen
             TMP102_SET_IRQ_FALLING_EDGE;// IRQ auf fallende Flanke setzen
551
             TMP102_ALERT_CLEAR_IRQ;
553          TMP102_ALERT_IRQ_ENABLE;
      }
555     break;                            // Vector P1IFG.1
    case 6: break;                           // Vector P1IFG.2
557 case 8: break;                           // Vector P1IFG.3
    case 10: break;                          // Vector P1IFG.4
559 case 12: break;                          // Vector P1IFG.5
    case 14: break;                          // Vector P1IFG.6
561 case 16: break;                          // Vector P1IFG.7


563
  }
565 }

567 /*************************************************************************
** IRQ Temperatursensor TMP102
569 ** Alarmausgang
*************************************************************************/
571
#pragma vector=PORT2_VECTOR
573 __interrupt void PORT2_ISR(void)
{
575     switch(__even_in_range(P2IV,16))
     {
577      case 0: break; // No Interrupt
         case 2: break; // P2.0
579      case 4: break; // P2.1
         case 6: break; // P2.2
581      case 8: break; // P2.3
         case 10: break; // P2.4
583      case 12:   // Wake interrupt occured
           /*****************************************************
585        ** IRQ Wakeup empfangen
           *****************************************************/
587            // Exit Low Power Mode 4
               EXIT_LPM4;
589
               // Disable and clear the wake interrupt as the sensor is awake now
591            AS3930_WAKE_IRQ_DISABLE;
```

```
                AS3930_WAKE_CLEAR_IRQ;
593
                //TMP102_SET_IRQ_FALLING_EDGE;
595             //TMP102_ALERT_IRQ_ENABLE;
                //TMP102_ALERT_CLEAR_IRQ;
597
                // Turn on the red LED
599             led_on(LED_AWAKE);
                //TIMER_B_START;
601
                // Change to RX state
603             adg918_transceiver();           // Connect the loop antenne with the transceiver
                adc12_volt_init(clk_set);       // Initialize ADC
605
                state = S_WAKEUP;
607
                rx_data_length = 0;             // Standart Datenlänge empangen
609             cc430_set_rx(brate_start);
                rx(HEADER_LENGTH + rx_data_length);
611

613             break; // P2.5
            case 14:   // Recieved Burst Signal
615           /*****************************************************
               ** IRQ Burstmode
617            *****************************************************/
              // Interrupt disablen
619         //  P3OUT &= ~BIT4;// TODO ENTFERNEN
              CC430_BURST_SIG_IRQ_DISABLE;
621           CC430_BURST_SIG_CLEAR_IRQ;

623           TIMER_B_STOP;    // Timer stoppen

625   //        P3OUT &= ~BIT4;// TODO ENTFERNEN
            switch(adc_select) {        // EM 01.10.2014
627             case ADC12_INT_SELECT:
                  sample_burst_buf[burst_counter] = adc12_get_volt_sample(clk_set);
629               break;
                case ADC24_EXT_SELECT :
631               sample_burst_buf[burst_counter] = adc24_get_volt_sample(clk_set);
                  break;
633           }

635           burst_counter++;     // Burstwert hochzählen

637

            if(burst_freq == BURST_FREQ_2000HZ || burst_freq == BURST_FREQ_4000HZ || burst_freq == BURST_FREQ_6000HZ || burst_freq ==
      BURST_FREQ_8000HZ || burst_freq == BURST_FREQ_10000HZ)
639         {
              CC430_BURST_SIG_CLEAR_IRQ;
641           if(burst_counter < burst_values)
                CC430_BURST_SIG_IRQ_ENABLE;
643           //Flag zurücksetzen
              TIMER_B_FLAG_RESET;
645         }

647


649         // Timer einstellen
            timer_b_init_burst(burst_freq);
651
            // Timer starten
653         TIMER_B_START_UP_MODE;
  //          P3OUT |= BIT4;// TODO ENTFERNEN
655             break; // P2.6
            case 16: break; // P2.7
657     }

659 }

661 /*****************************************************************************
    ** ISR TIMERA0: Balancierung 500ms
663 **
    *****************************************************************************/
665 #pragma vector=TIMER0_A0_VECTOR
    __interrupt void TIMER0_A0_ISR(void)
667 {
  if(IRQ_TIMERA_IS_DELAY) {
669   __bic_SR_register_on_exit(LPM4_bits);
  }
671
    if (IRQ_TIMERA_IS_BALANCING) {
673
        // Keine nested Interrupt
675     CC430_END_OF_PKT_IRQ_DISABLE;
```

```
        CC430_BURST_SIG_IRQ_DISABLE;

        balancing_time++;

        // 10min Balancieren -> 10min * 60sek *2 = 1200
        if(balancing_time < 1200) {

            uint16_t actual_volt = 0x0000;
            uint16_t actual_temp = 0x0000;

            // Aktuelle Werte holen
            actual_volt = adc12_get_volt_sample(clk_set);
            actual_temp = temp_sensor_get_temp();

            sample_buf_volt = actual_volt;

            // Temperaturkontrolle
            if (TEMP_IS_NORMAL) {               // Temperatur ist Normal
                if (actual_temp > upper_balanc_temp)   // Ist aktuelle Temp. zu hoch?
                    TEMP_SET_HIGH;
            } else if(TEMP_IS_HIGH) {
                if (actual_temp < lower_balanc_temp)   // Ist aktuelle Temp. ok?
                    TEMP_SET_NORMAL;
            }

            // Spannungskontrolle
            if (actual_volt > balancing_volt) {
                if (TEMP_IS_NORMAL) {
                    balancing_on();
                } else {
                    balancing_off();
                }
            } else { // Stop Balancing
                TIMER_A_STOP;
                balancing_off();
                balanc_state = OFF;
                IRQ_TIMERA_UNSET_BALANCING;
                TIMER_A_0_CM_IRQ_DISABLE;
                TIMER_A_1_CM_IRQ_DISABLE;
            }
        } else {
            TIMER_A_STOP;
            balancing_off();
            balanc_state = OFF;
            IRQ_TIMERA_UNSET_BALANCING;
            TIMER_A_0_CM_IRQ_DISABLE;
            TIMER_A_1_CM_IRQ_DISABLE;
        }

        if(balanc_state == OFF) {
            TIMER_A_0_CM_IRQ_DISABLE;
            TIMER_A_1_CM_IRQ_DISABLE;
        }

        CC430_END_OF_PKT_IRQ_ENABLE;
        CC430_BURST_SIG_IRQ_ENABLE;
    }
}


//#pragma vector=TIMER0_A1_VECTOR
//__interrupt void TIMER0_A1_ISR(void)
//{
//  switch(__even_in_range(TA0IV,14))
//  {
//    case 0: break;  //  No iterrupt pending
//    case 2: break;  //  TA0CCR1 CCIFG
//    case 4: break;  //  TA0CCR2 CCIFG
//    case 6: break;  //  TA0CCR3 CCIFG
//    case 8: break;  //  TA0CCR4 CCIFG
//    case 10: break; //  TA0CCR5 CCIFG
//    case 12: break; //  TA0CCR6 CCIFG
//    case 14: break; //  Timer Overflow
//  }
//}

/*************************************************************************
** ISR TIMERB
** Vector B1: öffnet das Zeitfenster
** Vector B0: schließt das Zeitfenster
*************************************************************************/



#pragma vector=TIMER1_A0_VECTOR
```

```
761 __interrupt void TIMER1_A0_ISR(void)
    {       //TA1CCR0 CCIFG
763   if(IRQ_TIMERB_IS_BURST) {
        //  P3OUT |= BIT4;// TODO ENTFERNEN
765           if(burst_counter >= burst_values) { //Sind alles Werte durchgekommen

767             CC430_BURST_SIG_IRQ_DISABLE;
                CC430_BURST_SIG_CLEAR_IRQ;
769
                TIMER_B_STOP;
771             TIMER_B_0_CM_IRQ_DISABLE;
                TIMER_B_1_CM_IRQ_DISABLE;
773
                burst_error = 0;    // Burst error zurücksetzen
775
                //Flag zurücksetzen
777             TIMER_B_FLAG_RESET;

779             IRQ_TIMERB_UNSET_BURST;
                rx_command = COMMAND_BACK_FROM_BURST;
781             command_recived = 1;

783           } else { //Fehler wird dedektiert
            //   P3OUT |= BIT2;      P3OUT &= ~BIT2; // TODO entfernen
785             BURST_ERROR_FLAG_SET;        // Fehlerflag setzen

787             // bei hohen Frequenzen gibt es keine Fensterung mehr, Interrupt bleibt freigeschalten
                if(burst_freq == BURST_FREQ_2000HZ || burst_freq == BURST_FREQ_4000HZ ||burst_freq == BURST_FREQ_6000HZ
789                     || burst_freq == BURST_FREQ_8000HZ || burst_freq == BURST_FREQ_10000HZ) {
                  CC430_BURST_SIG_CLEAR_IRQ;
791               CC430_BURST_SIG_IRQ_ENABLE;

793                 //Flag zurücksetzen
                    TIMER_B_FLAG_RESET;
795             }else{
                    // Interrupt disablen
797               CC430_BURST_SIG_IRQ_DISABLE;
                    CC430_BURST_SIG_CLEAR_IRQ;
799             }

801             // Timer einstellen
                timer_b_init_burst(burst_freq);
803
                // Timer starten
805             TIMER_B_START_UP_MODE;

807
                sample_burst_buf[burst_counter] = 0xFFFF;   // Fehlerwert
809             burst_counter++;                            // Burstcounter hochzählen
                burst_error++;                              // Burstfehler hochzählen
811         //    P3OUT &= ~BIT2; // TODO entfernen
            }
813
      }
815 }

817

819 #pragma vector=TIMER1_A1_VECTOR
    __interrupt void TIMER1_A1_ISR(void)
821 {
    switch(__even_in_range(TA1IV,14))
823 {
      case 0: break;  //  No iterrupt pending
825   case 2:
        // 528 Zeit für ISR abarbeitung bis Timer startet
827       // test = TBR + 528;
          if(IRQ_TIMERB_IS_BURST)
829         {
          //  P3OUT |=  BIT4;// TODO ENTFERNEN
831       // Interrupt enabeln
            CC430_BURST_SIG_CLEAR_IRQ;
833       if(burst_counter < burst_values)
          CC430_BURST_SIG_IRQ_ENABLE;
835
          //Flag zurücksetzen
837       TIMER_B_FLAG_RESET;
          }
839     break;  //  TA1CCR1 CCIFG
      case 4: break;  //  TA1CCR2 CCIFG
841   case 6: break;  //  TA1CCR3 CCIFG
      case 8: break;  //  TA1CCR4 CCIFG
843   case 10: break; //  TA1CCR5 CCIFG
      case 12: break; //  TA1CCR6 CCIFG
845   case 14: break; //  Timer Overflow
```

```
    }
}
```

code/ZSK3V04/isr.c

E.26 led.c

```
/*-------------------------------------------------------------------
    Description:    LED driver.
    Date:           11/22/2012
    Last Update:    11/22/2012
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#include "main.h"

/*-------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------*/

void led_init(void)
{
    LED_1_PxDIR |= LED_1_PIN;
    LED_2_PxDIR |= LED_2_PIN;
    LED_3_PxDIR |= LED_3_PIN;

    led_off(LED_ALL);
}

void led_on(LED_t led)
{
#ifdef ENABLE_LEDS
    switch(led) {
        case LED_AWAKE: LED_1_PxOUT |= LED_1_PIN; break;
        case LED_TX:    LED_2_PxOUT |= LED_2_PIN; break;
        case LED_RX:    LED_3_PxOUT |= LED_3_PIN; break;
        case LED_ALL:   led_on(LED_AWAKE);
                        led_on(LED_TX);
                        led_on(LED_RX);
                        break;
        default: break;
    }
#endif /* ENABLE_LEDS */
}

void led_off(LED_t led)
{
    switch(led) {
        case LED_AWAKE: LED_1_PxOUT &= ~LED_1_PIN; break;
        case LED_TX:    LED_2_PxOUT &= ~LED_2_PIN; break;
        case LED_RX:    LED_3_PxOUT &= ~LED_3_PIN; break;
        case LED_ALL:   led_off(LED_AWAKE);
                        led_off(LED_TX);
                        led_off(LED_RX);
                        break;
        default: break;
    }
}

void led_toggle(LED_t led)
{
#ifdef ENABLE_LEDS
    switch(led) {
        case LED_AWAKE: LED_1_PxOUT ^= LED_1_PIN; break;
        case LED_TX:    LED_2_PxOUT ^= LED_2_PIN; break;
        case LED_RX:    LED_3_PxOUT ^= LED_3_PIN; break;
        case LED_ALL:   led_toggle(LED_AWAKE);
                        led_toggle(LED_TX);
                        led_toggle(LED_RX);
                        break;
        default: break;
    }
#endif /* ENABLE_LEDS */
}
```

code/ZSK3V04/led.c

E.27 led.h

```
/*-------------------------------------------------------------------
    Description:    LED driver.
    Date:           11/22/2012
    Last Update:    24/07/2014 - EM
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#ifndef LED_H_
#define LED_H_

#include "main.h"

/*-------------------------------------------------------------------
  Defines
-------------------------------------------------------------------*/

#define LED_1_PxDIR  (P3DIR)  // changed 24/07/2014 - EM
#define LED_1_PxOUT  (P3OUT)  // changed 24/07/2014 - EM
#define LED_1_PIN    (BIT4)   // changed 24/07/2014 - EM

#define LED_2_PxDIR  (P3DIR)  // changed 24/07/2014 - EM
#define LED_2_PxOUT  (P3OUT)  // changed 24/07/2014 - EM
#define LED_2_PIN    (BIT3)   // changed 24/07/2014 - EM

#define LED_3_PxDIR  (P3DIR)  // changed 24/07/2014 - EM
#define LED_3_PxOUT  (P3OUT)  // changed 24/07/2014 - EM
#define LED_3_PIN    (BIT2)   // changed 24/07/2014 - EM

/*-------------------------------------------------------------------
  Types
-------------------------------------------------------------------*/

typedef enum
{
  LED_AWAKE = 0,
  LED_TX,
  LED_RX,
  LED_ALL
} LED_t;

/*-------------------------------------------------------------------
  Public functions
-------------------------------------------------------------------*/

void led_init(void);
void led_on(LED_t led);
void led_off(LED_t led);
void led_toggle(LED_t led);

#endif /* LED_H_ */
```

code/ZSK3V04/led.h

E.28 RF1A.c

```
#include "RF1A.h"
#include "cc430x513x.h"

// *************************************************************************
// @fn          Strobe
// @brief       Send a command strobe to the radio. Includes workaround for RF1A7
// @param       unsigned char strobe        The strobe command to be sent
// @return      unsigned char statusByte    The status byte that follows the strobe
// *************************************************************************
unsigned char Strobe(unsigned char strobe)
{
  unsigned char statusByte = 0;
  unsigned int  gdo_state;

  // Check for valid strobe command
  if((strobe == 0xBD) || ((strobe >= RF_SRES) && (strobe <= RF_SNOP)))
  {
    // Clear the Status read flag
    RF1AIFCTL1 &= ~(RFSTATIFG);

    // Wait for radio to be ready for next instruction
    while( !(RF1AIFCTL1 & RFINSTRIFG));

    // Write the strobe instruction
    if ((strobe > RF_SRES) && (strobe < RF_SNOP))
    {
      gdo_state = ReadSingleReg(IOCFG2);    // buffer IOCFG2 state
      WriteSingleReg(IOCFG2, 0x29);         // chip-ready to GDO2

      RF1AINSTRB = strobe;
      if ( (RF1AIN&0x04)== 0x04 )           // chip at sleep mode
      {
        if ( (strobe == RF_SXOFF) || (strobe == RF_SPWD) || (strobe == RF_SWOR) ) { }
        else
        {
          while ((RF1AIN&0x04)== 0x04);     // chip-ready ?
          // Delay for ~810usec at 1.05MHz CPU clock, see erratum RF1A7
          __delay_cycles(850);
        }
      }
      WriteSingleReg(IOCFG2, gdo_state);    // restore IOCFG2 setting

      while( !(RF1AIFCTL1 & RFSTATIFG) );
    }
    else                        // chip active mode (SRES)
    {
      RF1AINSTRB = strobe;
    }
    statusByte = RF1ASTATB;
  }
  return statusByte;
}

// *************************************************************************
// @fn          ReadSingleReg
// @brief       Read a single byte from the radio register
// @param       unsigned char addr      Target radio register address
// @return      unsigned char data_out  Value of byte that was read
// *************************************************************************
unsigned char ReadSingleReg(unsigned char addr)
{
  unsigned char data_out;

  // Check for valid configuration register address, 0x3E refers to PATABLE
  if ((addr <= 0x2E) || (addr == 0x3E))
    // Send address + Instruction + 1 dummy byte (auto-read)
    RF1AINSTR1B = (addr | RF_SNGLREGRD);
  else
    // Send address + Instruction + 1 dummy byte (auto-read)
    RF1AINSTR1B = (addr | RF_STATREGRD);

  while (!(RF1AIFCTL1 & RFDOUTIFG) );
  data_out = RF1ADOUTB;                     // Read data and clears the RFDOUTIFG

  return data_out;
}

// *************************************************************************
// @fn          WriteSingleReg
// @brief       Write a single byte to a radio register
// @param       unsigned char addr      Target radio register address
// @param       unsigned char value     Value to be written
```

```
// @return      none
// ****************************************************************************
void WriteSingleReg(unsigned char addr, unsigned char value)
{
  while (!(RF1AIFCTL1 & RFINSTRIFG));       // Wait for the Radio to be ready for next instruction
  RF1AINSTRB = (addr | RF_SNGLREGWR);     // Send address + Instruction

  RF1ADINB = value;           // Write data in

  __no_operation();
}

// ****************************************************************************
// @fn          ReadBurstReg
// @brief       Read multiple bytes to the radio registers
// @param       unsigned char addr      Beginning address of burst read
// @param       unsigned char *buffer   Pointer to data table
// @param       unsigned char count     Number of bytes to be read
// @return      none
// ****************************************************************************
void ReadBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count)
{
  unsigned int i;
  if(count > 0)
  {
    while (!(RF1AIFCTL1 & RFINSTRIFG));       // Wait for INSTRIFG
    RF1AINSTR1B = (addr | RF_REGRD);          // Send addr of first conf. reg. to be read
                                              // ... and the burst-register read instruction
    for (i = 0; i < (count-1); i++)
    {
      while (!(RFDOUTIFG&RF1AIFCTL1));        // Wait for the Radio Core to update the RF1ADOUTB reg
      buffer[i] = RF1ADOUT1B;                 // Read DOUT from Radio Core + clears RFDOUTIFG
                                              // Also initiates auo-read for next DOUT byte
    }
    buffer[count-1] = RF1ADOUT0B;             // Store the last DOUT from Radio Core
  }
}

// ****************************************************************************
// @fn          WriteBurstReg
// @brief       Write multiple bytes to the radio registers
// @param       unsigned char addr      Beginning address of burst write
// @param       unsigned char *buffer   Pointer to data table
// @param       unsigned char count     Number of bytes to be written
// @return      none
// ****************************************************************************
void WriteBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count)
{
  unsigned char i;

  if(count > 0)
  {
    while (!(RF1AIFCTL1 & RFINSTRIFG));       // Wait for the Radio to be ready for next instruction
    RF1AINSTRW = ((addr | RF_REGWR)<<8 ) + buffer[0]; // Send address + Instruction

    for (i = 1; i < count; i++)
    {
      RF1ADINB = buffer[i];                   // Send data
      while (!(RFDINIFG & RF1AIFCTL1));       // Wait for TX to finish
    }
    i = RF1ADOUTB;                            // Reset RFDOUTIFG flag which contains status byte
  }
}

// ****************************************************************************
// @fn          ResetRadioCore
// @brief       Reset the radio core using RF_SRES command
// @param       none
// @return      none
// ****************************************************************************
void ResetRadioCore (void)
{
  Strobe(RF_SRES);                          // Reset the Radio Core
  Strobe(RF_SNOP);                          // Reset Radio Pointer
}

// ****************************************************************************
// @fn          WriteRfSettings
// @brief       Write the minimum set of RF configuration register settings
// @param       RF_SETTINGS *pRfSettings  Pointer to the structure that holds the rf settings
// @return      none
// ****************************************************************************
void cc1101_config_no_packet() {

  WriteSingleReg(IOCFG2,0x0B);  //GDO2 Output Configuration
```

```
168   WriteSingleReg(IOCFG0,0x2D);   //GDO0 Output Configuration
      WriteSingleReg(FIFOTHR,0x47); //RX FIFO and TX FIFO Thresholds
170   WriteSingleReg(PKTCTRL0,0x32);//Packet Automation Control
      WriteSingleReg(FSCTRL1,0x06); //Frequency Synthesizer Control
172   WriteSingleReg(FREQ2,0x10);    //Frequency Control Word, High Byte
      WriteSingleReg(FREQ1,0xB1);    //Frequency Control Word, Middle Byte
174   WriteSingleReg(FREQ0,0x3B);    //Frequency Control Word, Low Byte
      WriteSingleReg(MDMCFG4,0xCA); //Modem Configuration
176   WriteSingleReg(MDMCFG3,0x93); //Modem Configuration
      WriteSingleReg(MDMCFG2,0x30); //Modem Configuration
178   WriteSingleReg(DEVIATN,0x34); //Modem Deviation Setting
      WriteSingleReg(MCSM0,0x10);    //Main Radio Control State Machine Configuration
180   WriteSingleReg(FOCCFG,0x16);   //Frequency Offset Compensation Configuration
      WriteSingleReg(WORCTRL,0xFB); //Wake On Radio Control
182   WriteSingleReg(FREND0,0x11);   //Front End TX Configuration
      WriteSingleReg(FSCAL3,0xE9);   //Frequency Synthesizer Calibration
184   WriteSingleReg(FSCAL2,0x2A);   //Frequency Synthesizer Calibration
      WriteSingleReg(FSCAL1,0x00);   //Frequency Synthesizer Calibration
186   WriteSingleReg(FSCAL0,0x1F);   //Frequency Synthesizer Calibration
      WriteSingleReg(TEST2,0x81);    //Various Test Settings
188   WriteSingleReg(TEST1,0x35);    //Various Test Settings
      WriteSingleReg(TEST0,0x09);    //Various Test Settings
190   WriteSinglePATable(PATABLE_VAL);

192 }

194 // ******************************************************************************
    // @fn          WritePATable
196 // @brief       Write data to power table
    // @param       unsigned char value   Value to write
198 // @return      none
    // ******************************************************************************
200 void WriteSinglePATable(unsigned char value)
    {
202   while( !(RF1AIFCTL1 & RFINSTRIFG));
      RF1AINSTRW = 0x3E00 + value;              // PA Table single write
204
      while( !(RF1AIFCTL1 & RFINSTRIFG));
206   RF1AINSTRB = RF_SNOP;                     // reset PA_Table pointer
    }
208
    // ******************************************************************************
210 // @fn          WritePATable
    // @brief       Write to multiple locations in power table
212 // @param       unsigned char *buffer Pointer to the table of values to be written
    // @param       unsigned char count Number of values to be written
214 // @return      none
    // ******************************************************************************
216 void WriteBurstPATable(unsigned char *buffer, unsigned char count)
    {
218   volatile char i = 0;

220   while( !(RF1AIFCTL1 & RFINSTRIFG));
      RF1AINSTRW = 0x7E00 + buffer[i];          // PA Table burst write
222
      for (i = 1; i < count; i++)
224   {
        RF1ADINB = buffer[i];                   // Send data
226     while (!(RFDINIFG & RF1AIFCTL1));       // Wait for TX to finish
      }
228   i = RF1ADOUTB;                            // Reset RFDOUTIFG flag which contains status byte

230   while( !(RF1AIFCTL1 & RFINSTRIFG));
      RF1AINSTRB = RF_SNOP;                     // reset PA Table pointer
232 }
```

code/ZSK3V04/RF1A.c

E.29 RF1A.h

```
/* -------------------------------------------------- Defines
 * ------------------------------------------------------------
 */
#define  PATABLE_VAL        (0x51)          // 0 dBm output

void ResetRadioCore (void);
unsigned char Strobe(unsigned char strobe);

void cc1101_config_no_packet();

void WriteSingleReg(unsigned char addr, unsigned char value);
void WriteBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count);
unsigned char ReadSingleReg(unsigned char addr);
void ReadBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count);
void WriteSinglePATable(unsigned char value);
void WriteBurstPATable(unsigned char *buffer, unsigned char count);
```

code/ZSK3V04/RF1A.h

E.30 temp_sensor.c

```
/*******************************************************************************
**    Description:    Temperatur Sensor
**    Hardware:       BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:           07/03/2013
**    Last Update:    30/07/2014 - EM
**    Author:         Nico Sassano
*******************************************************************************/

#include "main.h"

extern uint16_t config_value;

void temp_sensor_init(void) {
  i2c_init();         // added 30/07/2014 - EM
    temp_sensor_get_contr_reg();
}

void temp_sensor_get_contr_reg() {
    uint8_t data[2];

    /************************************************************
    * Pointer wird auf das Config Register ausgerichtet
    ************************************************************/
    data[0] = TMP102_REG_CONF; // Value for Pointer-Register

    i2c_write(ADDR_TMP102, 1, data);      // Set Pointer-Register

    i2c_read(ADDR_TMP102, 2, data);       // Read Control-Register

    config_value = (uint16_t) data[0] << 4;
    config_value |= data[1] >> 4;

}

void temp_sensor_config_reg() {
    uint8_t data[3];

    /************************************************************
    * Hier wird der Temperatursensor konfiguriert
    ************************************************************/
    TMP102_EM_OFF;
    TMP102_CON_RATE_4;
    TMP102_SD_OFF;
    TMP102_COMPERATOR_MODE;
    TMP102_POL_INV;
    TMP102_FAULTS_6;

    data[0] = TMP102_REG_CONF;                      // Value for Pointer-Register
    data[1] = (uint8_t) (config_value >> 8);        // Byte 1 at first
    data[2] = (uint8_t) (config_value & 0x00FF);    // Byte 2 at last

    i2c_write(ADDR_TMP102, 3, data);                // Write Control-Register

}

/*****************************************************************
** Setzen der Alarmtemperatur
** 1 Digit -> 0,0625°C
** 0x1900  -> 25°C
*****************************************************************/
void temp_sensor_set_alert(uint16_t temp_low, uint16_t temp_high) {
    uint8_t data[3];

    /************************************************************
    * Low Daten setzen
    ************************************************************/
    data[0] = TMP102_REG_LOW;                       // Value for Pointer-Register
    data[1] = (uint8_t) (temp_low >> 8);            // Byte 1 at first
    data[2] = (uint8_t) (temp_low & 0x00FF);        // Byte 2 at last

    i2c_write(ADDR_TMP102, 3, data);                // Write Control-Register

    /************************************************************
    * High Daten setzen
    ************************************************************/
    data[0] = TMP102_REG_HIGH;                      // Value for Pointer-Register
    data[1] = (uint8_t) (temp_high >> 8);           // Byte 1 at first
    data[2] = (uint8_t) (temp_high & 0x00FF);       // Byte 2 at last

    i2c_write(ADDR_TMP102, 3, data);                // Write Control-Register
}

```

```
uint16_t temp_sensor_get_temp(void) {

    // pointer register of the temperatur sensor must be 0x00 for readout
    // the temperatur register.
    // When configurated as 12-bit:
    // One LSB equals 0.0625mC, negative numbers are represented in the binary
    // twos complement format.
    // 20mC --> 320 counts
    // 23mC --> 368 counts

    uint8_t data[2];
    uint16_t temp_value = 0x0000;

    data[0] = TMP102_REG_TEMP;                          // Value for Pointer-Register
    i2c_write(ADDR_TMP102, 1, data);                    // Set Pointer-Register
    temp_value = 0x0000;
    i2c_read(ADDR_TMP102, 2, data);

    temp_value = (uint16_t) data[0] << 4;
    temp_value |= data[1] >> 4;

    return temp_value;

}
```

code/ZSK3V04/temp_sensor.c

E.31 temp_sensor.h

```
/*******************************************************************************
**    Description:      Temperatur Sensor
**    Hardware:         BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:             07/03/2013
**    Last Update:      30/07/2014 - EM
**    Author:           Nico Sassano
*******************************************************************************/

#ifndef TEMP_SENSOR_H_
#define TEMP_SENSOR_H_

#include "main.h"

/**** Adress of the Temp. Sensor ************************************************/
#define ADDR_TMP102       0x48

#define TMP102_ALERT_PxREN            P1REN // changed 30/07/2014 - EM
#define TMP102_ALERT_PxOUT            P1OUT // changed 30/07/2014 - EM
#define TMP102_ALERT_PxDIR            P1DIR // changed 30/07/2014 - EM
#define TMP102_ALERT_PxIES            P1IES // changed 30/07/2014 - EM
#define TMP102_ALERT_PxIE             P1IE  // changed 30/07/2014 - EM
#define TMP102_ALERT_PxIFG            P1IFG // changed 30/07/2014 - EM
#define TMP102_ALERT_PIN              BIT1  // changed 30/07/2014 - EM

#define TMP102_ALERT_REN_EN                   (TMP102_ALERT_PxREN |= TMP102_ALERT_PIN)
#define TMP102_ALERT_SET_PULLUP               (TMP102_ALERT_PxOUT |= TMP102_ALERT_PIN)
#define TMP102_ALERT_SET_PULLDOWN             (TMP102_ALERT_PxOUT &= ~TMP102_ALERT_PIN)
#define TMP102_ALERT_DIR_IN                   (TMP102_ALERT_PxDIR &= ~TMP102_ALERT_PIN)
#define TMP102_ALERT_IRQ_RISING_EDGE          (TMP102_ALERT_PxIES &= ~TMP102_ALERT_PIN)
#define TMP102_ALERT_IRQ_FALLING_EDGE         (TMP102_ALERT_PxIES |= TMP102_ALERT_PIN)
#define TMP102_ALERT_IRQ_ENABLE               (TMP102_ALERT_PxIE  |= TMP102_ALERT_PIN)
#define TMP102_ALERT_IRQ_DISABLE              (TMP102_ALERT_PxIE  &= ~TMP102_ALERT_PIN)
#define TMP102_ALERT_IRQ_PENDING              ((TMP102_ALERT_PxIFG & TMP102_ALERT_PIN) == TMP102_ALERT_PIN)
#define TMP102_ALERT_CLEAR_IRQ                (TMP102_ALERT_PxIFG &= ~(TMP102_ALERT_PIN))

#define TMP102_SET_IRQ_RISING_EDGE            (TMP102_ALERT_DIR_IN); \
                                          (TMP102_ALERT_REN_EN); \
                                          (TMP102_ALERT_SET_PULLUP); \
                                          (TMP102_ALERT_IRQ_RISING_EDGE); \

#define TMP102_SET_IRQ_FALLING_EDGE           (TMP102_ALERT_DIR_IN); \
                                          (TMP102_ALERT_REN_EN); \
                                          (TMP102_ALERT_SET_PULLUP); \
                                          (TMP102_ALERT_IRQ_FALLING_EDGE); \


/**** Register adresses *********************************************************/
#define TMP102_REG_TEMP           0x00    // Temperatur
#define TMP102_REG_CONF           0x01    // Configuration
#define TMP102_REG_HIGH           0x02    // Temperatur Low
#define TMP102_REG_LOW            0x03    // Temperatur High

#define TMP102_REG_OS             0x8000
#define TMP102_REG_R1             0x4000
#define TMP102_REG_R0             0x2000
#define TMP102_REG_F1             0x1000
#define TMP102_REG_F0             0x0800
#define TMP102_REG_POL            0x0400
#define TMP102_REG_TM             0x0200
#define TMP102_REG_SD             0x0100

#define TMP102_REG_CR1            0x0080
#define TMP102_REG_CR0            0x0040
#define TMP102_REG_AL             0x0020
#define TMP102_REG_EM             0x0010

#define TMP102_EM_OFF             (config_value &=~TMP102_REG_EM)
#define TMP102_EM_ON              (config_value |= TMP102_REG_EM)
#define TMP102_SD_OFF             (config_value &=~TMP102_REG_SD)
#define TMP102_SD_ON              (config_value |= TMP102_REG_SD)
#define TMP102_COMPERATOR_MODE    (config_value &=~TMP102_REG_TM)
#define TMP102_INTERRUPT_MODE     (config_value |= TMP102_REG_TM)
#define TMP102_POL_INV            (config_value &=~TMP102_REG_POL)
#define TMP102_POL_NORM           (config_value |= TMP102_REG_POL)

#define TMP102_FAULTS_1           (config_value &= ~(TMP102_REG_F1)); \
                              (config_value &= ~(TMP102_REG_F0));
#define TMP102_FAULTS_2           (config_value &= ~(TMP102_REG_F1)); \
                                  (config_value |=  (TMP102_REG_F0));
#define TMP102_FAULTS_4           (config_value |=  (TMP102_REG_F1)); \
                                  (config_value &= ~(TMP102_REG_F0));
#define TMP102_FAULTS_6           (config_value |=  (TMP102_REG_F1)); \
```

```
                                (config_value |=  (TMP102_REG_F0));
84
#define TMP102_CON_RATE_1       (config_value &= ~(TMP102_REG_CR1)); \
86                              (config_value &= ~(TMP102_REG_CR0));
#define TMP102_CON_RATE_2       (config_value &= ~(TMP102_REG_CR1)); \
88                              (config_value |=  (TMP102_REG_CR0));
#define TMP102_CON_RATE_4       (config_value |=  (TMP102_REG_CR1)); \
90                              (config_value &= ~(TMP102_REG_CR0));
#define TMP102_CON_RATE_6       (config_value |=  (TMP102_REG_CR1)); \
92                              (config_value |=  (TMP102_REG_CR0));

94 /**** Prototyp declaration ***********************************************************/
void temp_sensor_init(void);
96 void temp_sensor_get_contr_reg(void);
uint16_t temp_sensor_get_temp(void);
98 void temp_sensor_config_reg(void);
void temp_sensor_set_alert(uint16_t, uint16_t);
100
#endif /* TEMP_SENSOR_H_ */
```

code/ZSK3V04/temp_sensor.h

E.32 timer.c

```
/*******************************************************************************
**     Description:     Timer
**     Hardware:        BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**     Date:            13/03/2013
**     Last Update:     05/08/2014  - EM
**     Author:          Nico Sassano
*******************************************************************************/

#include "main.h"

extern volatile uint8_t irq_timera;             // Status TimerA
extern volatile uint8_t irq_timerb;             // Status TimerB
extern volatile uint8_t burst_error_flag;
extern volatile uint8_t adc_select;         // added 29/09/2014  - EM
extern uint8_t clk_set;                 // added 29/09/2014  - EM

void timer_a_init(void) {
    /*************************************************************
    * Timer A initialisierung 13.03.13 NS
    *************************************************************/

    // timer clear
    TACTL = TACLR;
    // no grouping, counter length = 16bit, clock source = SMCLK, div = 8
    TACTL |= (TASSEL1 | ID1 | ID0);
    // Capture/Compare Reg 0 set
    TACCR0 = 62500;
    // Compare-mode, IRQ enable
    TACCTL0 |= CCIE;
    // Capture/Compare Reg 1 set
    //TBCCR1 = TIME_TO_SAMPLE;
    // Compare-mode, IRQ enable
    //TBCCTL1 |= CCIE;
    // Capture/Compare Reg 2 set
    //TBCCR2 = TIME_TO_TX;
    // Compare-mode, IRQ enable
    //TBCCTL2 |= CCIE;
}

/*************************************************************
** Timer A Initialisierung für die Balancierung
** 1MHz; DIV=8; TACCR0 = 62500 -> 500ms
*************************************************************/
void timer_a_init_balanc(void) {
    IRQ_TIMERA_SET_BALANCING;
    TIMER_A_STOP;
    TIMER_A_RESET;
    TIMER_A_SOURCE_SMCLK;
    TIMER_A_SOURCE_DIV_8;

    TACCR0 = 62500;
    TIMER_A_0_CM_IRQ_ENABLE;
}


/*************************************************************
** Timer A Initialisierung für die Balancierung
** 1MHz; DIV=8; TACCR0 = 62500 -> 500ms
*************************************************************/
void timer_a_init_test(void) {
    TIMER_A_STOP;
    TIMER_A_RESET;

    TIMER_A_SOURCE_SMCLK;
    TIMER_A_SOURCE_DIV_1;
    TACCR0 = 6000;
    TIMER_A_0_CM_IRQ_ENABLE;

    TACCR1 = 600;
    TIMER_A_1_CM_IRQ_ENABLE;
}

/*************************************************************
** Timer B Initialisierung für die Burstmessung
**
*************************************************************/
void timer_b_init_burst(uint8_t freq) {
    IRQ_TIMERB_SET_BURST;

    TIMER_B_STOP;
    TIMER_B_RESET;
    TIMER_B_SOURCE_SMCLK;
```

```
83
    switch(freq) {
85
        /*************************
        ** 16MHz; TimerDiv 1
        ** Periode  -> 100us
89      ** 1472 (ISR)-> 92us
        *************************/
91      case BURST_FREQ_10000HZ: {
            TIMER_B_SOURCE_DIV_1;
93
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
95              TBCCR0 = 2000 - 1472;   // 125us - 92us (ISR)
            } else {
97              TBCCR0 = 1600 - 184;    // 100 us - 11,5us (ISR)
                BURST_ERROR_FLAG_UNSET;
99          }

101         // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
103         TIMER_B_1_CM_IRQ_DISABLE;
        }break;
105
        /*************************
107     ** 16MHz; TimerDiv 1
        ** Periode      -> 125us
109     ** 1472 (ISR)   -> 92us
        *************************/
111     case BURST_FREQ_8000HZ: {
            TIMER_B_SOURCE_DIV_1;
113
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
115           switch(adc_select) {    // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
117             TBCCR0 = 2400 - 480;   // 150us - 32us (ISR)
                break;
119           case ADC12_INT_SELECT :
                TBCCR0 = 2400 - 1472;   // 150us - 92us (ISR)
121             break;
              }
123         } else {
                TBCCR0 = 2000 - 184;    // 125 us - 11,5us (ISR)
125             BURST_ERROR_FLAG_UNSET;
            }
127
            // Interrupts für TimerB freischalten
129         TIMER_B_0_CM_IRQ_ENABLE;
            TIMER_B_1_CM_IRQ_DISABLE;
131     }break;

133     /*************************
        ** 16MHz; TimerDiv 1
135     ** Periode  -> 166us
        ** 1472 (ISR)-> 92us
137     *************************/
        case BURST_FREQ_6000HZ: {
139         TIMER_B_SOURCE_DIV_1;

141         if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
              switch(adc_select) {    // added 29/09/2014  - EM
143           case ADC24_EXT_SELECT :
                TBCCR0 = 3200 - 480;  // 200 us - 32us (ISR)
145             break;
              case ADC12_INT_SELECT :
147             TBCCR0 = 3200 - 1472;    // 200 us - 92us (ISR)
                break;
149           }
            } else {
151             TBCCR0 = 2656 - 184;     // 166 us - 11,5us (ISR)
                BURST_ERROR_FLAG_UNSET;
153         }

155         // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
157         TIMER_B_1_CM_IRQ_DISABLE;
        }break;
159
        /*************************
161     ** 16MHz; TimerDiv 1
        ** Periode  -> 250us
163     ** 1472 (ISR)-> 92us
        *************************/
165     case BURST_FREQ_4000HZ: {
            TIMER_B_SOURCE_DIV_1;
167
```

```
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
169           switch(adc_select) {     // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
171             TBCCR0 = 4400 - 480;  // 275us - 32us (ISR)
                break;
173           case ADC12_INT_SELECT :
                TBCCR0 = 4400 - 1472;   // 275us - 92us (ISR)
175             break;
              }
177         } else {
                TBCCR0 = 4000 - 184;     // 250 us - 11,5us (ISR)
179             BURST_ERROR_FLAG_UNSET;
            }
181
            // Interrupts für TimerB freischalten
183         TIMER_B_0_CM_IRQ_ENABLE;
            TIMER_B_1_CM_IRQ_DISABLE;
185       }break;

187       /*************************
          ** 16MHz; TimerDiv 1
189       ** Periode  -> 500us
          ** 1472 (ISR)-> 92us
191       *************************/
        case BURST_FREQ_2000HZ: {
193         TIMER_B_SOURCE_DIV_1;

195         if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
              switch(adc_select) {     // added 29/09/2014  - EM
197           case ADC24_EXT_SELECT :
                TBCCR0 = 9600 - 480;   // 600us -32us (ISR)
199             break;
              case ADC12_INT_SELECT :
201             TBCCR0 = 9600 - 1472;   // 600us - 92us (ISR)
                break;
203           }
            } else {
205             TBCCR0 = 8000 - 184;     // 500 us - 11,5us (ISR)
                BURST_ERROR_FLAG_UNSET;
207         }

209         // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
211         TIMER_B_1_CM_IRQ_DISABLE;
        }break;
213
        /*************************
215     ** 16MHz; TimerDiv 1
        ** Periode  -> 1ms
217     ** 12800     -> 800us
        ** 19200     -> 1.2ms
219     ** 1472 (ISR)-> 92us
        *************************/
221     case BURST_FREQ_1000HZ: {
            TIMER_B_SOURCE_DIV_1;
223
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
225           switch(adc_select) {     // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
227             TBCCR1 = 12800 - ADC24_ISR_TICKS_8000_SPS;
                TBCCR0 = 19200 - ADC24_ISR_TICKS_8000_SPS;
229             break;
              case ADC12_INT_SELECT :
231             TBCCR1 = 12800 - 1472;   // 800us  - 92us (ISR)
                TBCCR0 = 19200 - 1472;   // 1200us - 92us (ISR)
233             break;
              }
235         } else {
                TBCCR1 = 8600 ;          // 600 us
237             TBCCR0 = 16000;          // 1000 us
                BURST_ERROR_FLAG_UNSET;
239         }

241         // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
243         TIMER_B_1_CM_IRQ_ENABLE;

245     }break;

247     /*************************
        ** 16MHz; TimerDiv 1
249     ** Periode  -> 1.052ms
        ** 13632     -> 0.852 ms
251     ** 20032     -> 1.252 ms
        ** 1472 (ISR)-> 92us
```

```
253         ************************/
            case BURST_FREQ_950HZ: {
255             TIMER_B_SOURCE_DIV_1;
                if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
257               switch(adc_select) {     // added 29/09/2014 - EM
                  case ADC24_EXT_SELECT :
259                 TBCCR1 = 13632 - ADC24_ISR_TICKS_8000_SPS;     //TODO GENERISCH
                    TBCCR0 = 20032 - ADC24_ISR_TICKS_8000_SPS;
261                 break;
                  case ADC12_INT_SELECT :
263                 TBCCR1 = 13632 - 1472;  // 0.852 ms - 33 us
                    TBCCR0 = 20032 - 1472;   // 1.252 ms - 33 us
265                 break;
                  }
267             } else {
                    TBCCR1 = 10432;          // 652 us
269                 TBCCR0 = 16832;          // 1052 us
                    BURST_ERROR_FLAG_UNSET;
271             }

273             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
275             TIMER_B_1_CM_IRQ_ENABLE;
            }break;
277
            /************************
279         ** 16MHz; TimerDiv 1
            ** Periode  -> 1.10 ms
281         ** 14400     -> 0.90 ms
            ** 20800     -> 1.30 ms
283         ** 1472 (ISR)-> 92us
            ************************/
285         case BURST_FREQ_900HZ: {
                TIMER_B_SOURCE_DIV_1;
287             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {     // added 29/09/2014 - EM
289               case ADC24_EXT_SELECT :
                    TBCCR1 = 14400 - ADC24_ISR_TICKS_8000_SPS;     //TODO GENERISCH
291                 TBCCR0 = 20800 - ADC24_ISR_TICKS_8000_SPS;
                    break;
293               case ADC12_INT_SELECT :
                    TBCCR1 = 14400 - 1472;
295                 TBCCR0 = 20800 - 1472;
                    break;
297               }
                } else {
299                 TBCCR1 = 11200;          // 700 us
                    TBCCR0 = 17600;          // 1.1 ms
301                 BURST_ERROR_FLAG_UNSET;
                }
303
                // Interrupts für TimerB freischalten
305             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
307         }break;

309         /************************
            ** 16MHz; TimerDiv 1
311         ** Periode  -> 1.176 ms
            ** 15616     -> 0.976 ms
313         ** 22016     -> 1.376 ms
            ** 1472 (ISR)-> 92us
315         ************************/
            case BURST_FREQ_850HZ: {
317           TIMER_B_SOURCE_DIV_1;
                if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
319               switch(adc_select) {     // added 29/09/2014 - EM
                  case ADC24_EXT_SELECT :
321                 TBCCR1 = 15616 - ADC24_ISR_TICKS_8000_SPS;     //TODO GENERISCH
                    TBCCR0 = 22016 - ADC24_ISR_TICKS_8000_SPS;
323                 break;
                  case ADC12_INT_SELECT :
325                 TBCCR1 = 15616 - 1472;
                    TBCCR0 = 22016 - 1472;
327                 break;
                  }
329             } else {
                    TBCCR1 = 12416;          // 0.776 ms
331                 TBCCR0 = 18816;          // 1.176 ms
                    BURST_ERROR_FLAG_UNSET;
333             }

335             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
337             TIMER_B_1_CM_IRQ_ENABLE;
```

```
        }break;

        /************************
        ** 16MHz; TimerDiv 1
        ** Periode  -> 1.25 ms
        ** 16800    -> 1.05 ms
        ** 23200    -> 1.45 ms
        ** 1472 (ISR)-> 92us
        ************************/
        case BURST_FREQ_800HZ: {
            TIMER_B_SOURCE_DIV_1;
          if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
              switch(adc_select) {    // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
                TBCCR1 = 16800 - ADC24_ISR_TICKS_8000_SPS;    //TODO GENERISCH
                TBCCR0 = 23200 - ADC24_ISR_TICKS_8000_SPS;
                break;
              case ADC12_INT_SELECT :
                  TBCCR1 = 16800 - 1472;
                  TBCCR0 = 23200 - 1472;
                  break;
              }
            } else {
                TBCCR1 = 13600;          // 850us
                TBCCR0 = 20000;          // 1.25 ms
                BURST_ERROR_FLAG_UNSET;
            }

            // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
            TIMER_B_1_CM_IRQ_ENABLE;
        }break;

        /************************
        ** 16MHz; TimerDiv 1
        ** Periode  -> 1.33 ms
        ** 18080    -> 1.13 ms
        ** 24480    -> 1.53 ms
        ** 1472 (ISR)-> 92us
        ************************/
        case BURST_FREQ_750HZ: {
            TIMER_B_SOURCE_DIV_1;
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
              switch(adc_select) {    // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
                TBCCR1 = 18080 - ADC24_ISR_TICKS_8000_SPS;    //TODO GENERISCH
                TBCCR0 = 24480 - ADC24_ISR_TICKS_8000_SPS;
                break;
              case ADC12_INT_SELECT :
                TBCCR1 = 18080 - 1472;
                TBCCR0 = 24480 - 1472;
                break;
              }
            } else {
                TBCCR1 = 14880;          // 0.930 ms
                TBCCR0 = 21280;          // 1.330 ms
                BURST_ERROR_FLAG_UNSET;
            }

            // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
            TIMER_B_1_CM_IRQ_ENABLE;
        }break;

        /************************
        ** 16MHz; TimerDiv 1
        ** Periode  -> 1.49 ms
        ** 20640    -> 1.29 ms
        ** 27040    -> 1.69 ms
        ** 1472 (ISR)-> 92us
        ************************/
        case BURST_FREQ_700HZ: {
          TIMER_B_SOURCE_DIV_1;
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
              switch(adc_select) {    // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
                TBCCR1 = 20640 - ADC24_ISR_TICKS_4000_SPS;    //TODO GENERISCH
                TBCCR0 = 27040 - ADC24_ISR_TICKS_4000_SPS;
                break;
              case ADC12_INT_SELECT :
                TBCCR1 = 20640 - 1472;
                TBCCR0 = 27040 - 1472;
                break;
              }
            } else {
```

```
423                 TBCCR1 = 17440;          // 1.09 ms
                    TBCCR0 = 23840;          // 1.49 ms
425                 BURST_ERROR_FLAG_UNSET;
                }
427
                // Interrupts für TimerB freischalten
429             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
431         }break;

433         /************************
            ** 16MHz; TimerDiv 1
435         ** Periode  -> 1.54 ms
            ** 21440    -> 1.34 ms
437         ** 27840    -> 1.74 ms
            ** 1472 (ISR)-> 92us
439         ************************/
            case BURST_FREQ_650HZ: {
441             TIMER_B_SOURCE_DIV_1;

443             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {     // added 29/09/2014  - EM
445               case ADC24_EXT_SELECT :
                    TBCCR1 = 21440 - ADC24_ISR_TICKS_4000_SPS;     //TODO GENERISCH
447                 TBCCR0 = 27840 - ADC24_ISR_TICKS_4000_SPS;
                    break;
449               case ADC12_INT_SELECT :
                    TBCCR1 = 21440 - 1472;
451                 TBCCR0 = 27840 - 1472;
                    break;
453               }
                } else {
455                 TBCCR1 = 18240;          // 1.14 ms
                    TBCCR0 = 24640;          // 1.54 ms
457                 BURST_ERROR_FLAG_UNSET;
                }
459
                // Interrupts für TimerB freischalten
461             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
463         }break;

465         /************************
            ** 16MHz; TimerDiv 1
467         ** Periode  -> 1.60 ms
            ** 22400    -> 1.40 ms
469         ** 28800    -> 1.80 ms
            ** 1472 (ISR)-> 92us
471         ************************/
            case BURST_FREQ_600HZ: {
473             TIMER_B_SOURCE_DIV_1;

475             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {     // added 29/09/2014  - EM
477               case ADC24_EXT_SELECT :
                    TBCCR1 = 22400 - ADC24_ISR_TICKS_4000_SPS;     //TODO GENERISCH
479                 TBCCR0 = 28800 - ADC24_ISR_TICKS_4000_SPS;
                    break;
481               case ADC12_INT_SELECT :
                    TBCCR1 = 22400 - 1472;
483                 TBCCR0 = 28800 - 1472;
                    break;
485               }
                } else {
487                 TBCCR1 = 19200;          // 1.2 ms
                    TBCCR0 = 25600;          // 1.6 ms
489                 BURST_ERROR_FLAG_UNSET;
                }
491
                // Interrupts für TimerB freischalten
493             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
495         }break;

497         /************************
            ** 16MHz; TimerDiv 1
499         ** Periode  -> 1.80 ms
            ** 25600    -> 1.60 ms
501         ** 32000    -> 2.00 ms
            ** 1472 (ISR)-> 92us
503         ************************/
            case BURST_FREQ_550HZ: {
505             TIMER_B_SOURCE_DIV_1;

507             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
```

```
            switch(adc_select) {     // added 29/09/2014 - EM
509         case ADC24_EXT_SELECT :
              TBCCR1 = 25600 - ADC24_ISR_TICKS_4000_SPS;     //TODO GENERISCH
511           TBCCR0 = 32000 - ADC24_ISR_TICKS_4000_SPS;
              break;
513         case ADC12_INT_SELECT :
              TBCCR1 = 25600 - 1472;
515           TBCCR0 = 32000 - 1472;
              break;
517         }
          } else {
519           TBCCR1 = 22400;          // 1.4 ms
              TBCCR0 = 28800;          // 1.8 ms
521           BURST_ERROR_FLAG_UNSET;
          }
523
          // Interrupts für TimerB freischalten
525       TIMER_B_0_CM_IRQ_ENABLE;
          TIMER_B_1_CM_IRQ_ENABLE;
527     }break;

529     /************************
        ** 16MHz; TimerDiv 1
531     ** Periode  -> 2.00 ms
        ** 28800    -> 1.80 ms
533     ** 35200    -> 2.20 ms
        ** 1472 (ISR)-> 92us
535     ************************/
        case BURST_FREQ_500HZ: {
537       TIMER_B_SOURCE_DIV_1;

539       if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
            switch(adc_select) {     // added 29/09/2014 - EM
541         case ADC24_EXT_SELECT :
              TBCCR1 = 28800 - ADC24_ISR_TICKS_4000_SPS;     //TODO GENERISCH
543           TBCCR0 = 35200 - ADC24_ISR_TICKS_4000_SPS;
              break;
545         case ADC12_INT_SELECT :
              TBCCR1 = 28800 - 1472;
547           TBCCR0 = 35200 - 1472;
              break;
549         }
          } else {
551           TBCCR1 = 25600;          // 1.6 ms
              TBCCR0 = 32000;          // 2.0 ms
553           BURST_ERROR_FLAG_UNSET;
          }
555
          // Interrupts für TimerB freischalten
557       TIMER_B_0_CM_IRQ_ENABLE;
          TIMER_B_1_CM_IRQ_ENABLE;
559     }break;

561     /************************
        ** 16MHz; TimerDiv 1
563     ** Periode  -> 2.22 ms
        ** 32320    -> 2.02 ms
565     ** 38720    -> 2.42 ms
        ** 1472 (ISR)-> 92us
567     ************************/
        case BURST_FREQ_450HZ: {
569       TIMER_B_SOURCE_DIV_1;

571       if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
            switch(adc_select) {     // added 29/09/2014 - EM
573         case ADC24_EXT_SELECT :
              TBCCR1 = 32320 - ADC24_ISR_TICKS_4000_SPS;     //TODO GENERISCH
575           TBCCR0 = 38720 - ADC24_ISR_TICKS_4000_SPS;
              break;
577         case ADC12_INT_SELECT :
              TBCCR1 = 32320 - 1472;
579           TBCCR0 = 38720 - 1472;
              break;
581         }
          } else {
583           TBCCR1 = 29120;          // 1.82 ms
              TBCCR0 = 35520;          // 2.22 ms
585           BURST_ERROR_FLAG_UNSET;
          }
587
          // Interrupts für TimerB freischalten
589       TIMER_B_0_CM_IRQ_ENABLE;
          TIMER_B_1_CM_IRQ_ENABLE;
591     }break;
```

```
593         /************************
            ** 16MHz; TimerDiv 1
595         ** Periode  -> 2.50 ms
            ** 36800    -> 2.30 ms
597         ** 43200    -> 2.00 ms
            ** 1472 (ISR)-> 92us
599         ************************/
            case BURST_FREQ_400HZ: {
601             TIMER_B_SOURCE_DIV_1;

603             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {    // added 29/09/2014 - EM
605               case ADC24_EXT_SELECT :
                    TBCCR1 = 36800 - ADC24_ISR_TICKS_4000_SPS;    //TODO GENERISCH
607                 TBCCR0 = 43200 - ADC24_ISR_TICKS_4000_SPS;
                    break;
609               case ADC12_INT_SELECT :
                    TBCCR1 = 36800 - 1472;
611                 TBCCR0 = 43200 - 1472;
                    break;
613               }
                } else {
615                 TBCCR1 = 33600;          // 2.1 ms
                    TBCCR0 = 40000;          // 2.5 ms
617                 BURST_ERROR_FLAG_UNSET;
                }
619
                // Interrupts für TimerB freischalten
621             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
623         }break;

625         /************************
            ** 16MHz; TimerDiv 1
627         ** Periode  -> 2.85 ms
            ** 42400    -> 2.65 ms
629         ** 48800    -> 3.05 ms
            ** 1472 (ISR)-> 92us
631         ************************/
            case BURST_FREQ_350HZ: {
633             TIMER_B_SOURCE_DIV_1;

635             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {    // added 29/09/2014 - EM
637               case ADC24_EXT_SELECT :
                    TBCCR1 = 42400 - ADC24_ISR_TICKS_2000_DIV1_SPS;   //TODO GENERISCH
639                 TBCCR0 = 48800 - ADC24_ISR_TICKS_2000_DIV1_SPS;
                    break;
641               case ADC12_INT_SELECT :
                    TBCCR1 = 42400 - 1472;
643                 TBCCR0 = 48800 - 1472;
                    break;
645               }
                } else {
647                 TBCCR1 = 39200;          // 2.45 ms
                    TBCCR0 = 45600;          // 2.85 ms
649                 BURST_ERROR_FLAG_UNSET;
                }
651
                // Interrupts für TimerB freischalten
653             TIMER_B_0_CM_IRQ_ENABLE;
                TIMER_B_1_CM_IRQ_ENABLE;
655         }break;

657         /************************
            ** 16MHz; TimerDiv 1
659         ** Periode  -> 3.33 ms
            ** 50080    -> 3.13 ms
661         ** 56480    -> 3.53 ms
            ** 1472 (ISR)-> 92us
663         ************************/
            case BURST_FREQ_300HZ: {
665             TIMER_B_SOURCE_DIV_1;

667             if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
                  switch(adc_select) {    // added 29/09/2014 - EM
669               case ADC24_EXT_SELECT :
                    TBCCR1 = 50080 - ADC24_ISR_TICKS_2000_DIV1_SPS;   //TODO GENERISCH
671                 TBCCR0 = 56480 - ADC24_ISR_TICKS_2000_DIV1_SPS;
                    break;
673               case ADC12_INT_SELECT :
                    TBCCR1 = 50080 - 1472;
675                 TBCCR0 = 56480 - 1472;
                    break;
677               }
```

```
679             } else {
                    TBCCR1 = 46880;          // 2.93 ms
681                 TBCCR0 = 53280;          // 3.33 ms
                    BURST_ERROR_FLAG_UNSET;
683             }

685             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
687             TIMER_B_1_CM_IRQ_ENABLE;
            }break;
689
            /************************
691         ** 16MHz; TimerDiv 2
            ** Periode  -> 4.00 ms
693         ** 30400    -> 3.80 ms
            ** 33600    -> 4.20 ms
695         ** 264 (ISR)-> 92us
            ************************/
697         case BURST_FREQ_250HZ: {
                TIMER_B_SOURCE_DIV_2;
699
                if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
701               switch(adc_select) {     // added 29/09/2014 - EM
                  case ADC24_EXT_SELECT :
703                 TBCCR1 = 30400 - ADC24_ISR_TICKS_2000_DIV2_SPS;    //TODO GENERISCH
                    TBCCR0 = 33600 - ADC24_ISR_TICKS_2000_DIV2_SPS;
705                 break;
                  case ADC12_INT_SELECT :
707                 TBCCR1 = 30400 - 264;
                    TBCCR0 = 33600 - 264;
709                 break;
                  }
711             } else {
                    TBCCR1 = 28800;          // 3.6 ms
713                 TBCCR0 = 32800;          // 4.1 ms
                    BURST_ERROR_FLAG_UNSET;
715             }

717             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
719             TIMER_B_1_CM_IRQ_ENABLE;
            }break;
721
            /************************
723         ** 16MHz; TimerDiv 2
            ** Periode  -> 5.00 ms
725         ** 38400    -> 4.80 ms
            ** 41600    -> 5.20 ms
727         ** 264 (ISR)-> 92us
            ************************/
729         case BURST_FREQ_200HZ: {
                TIMER_B_SOURCE_DIV_2;
731
                if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
733               switch(adc_select) {     // added 29/09/2014 - EM
                  case ADC24_EXT_SELECT :
735                 TBCCR1 = 38400 - ADC24_ISR_TICKS_1000_DIV2_SPS;    //TODO GENERISCH
                    TBCCR0 = 41600 - ADC24_ISR_TICKS_1000_DIV2_SPS;
737                 break;
                  case ADC12_INT_SELECT :
739                 TBCCR1 = 38400 - 264;
                    TBCCR0 = 41600 - 264;
741                 break;
                  }
743             } else {
                    TBCCR1 = 36800;          // 4.6 ms
745                 TBCCR0 = 40000;          // 5.0 ms
                    BURST_ERROR_FLAG_UNSET;
747             }

749             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
751             TIMER_B_1_CM_IRQ_ENABLE;
            }break;
753
            /************************
755         ** 16MHz; TimerDiv 2
            ** Periode  -> 6.66 ms
757         ** 51680    -> 6.46 ms
            ** 54880    -> 6.86 ms
759         ** 264 (ISR)-> 92us
            ************************/
761         case BURST_FREQ_150HZ: {
                TIMER_B_SOURCE_DIV_2;
```

```
763
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
765           switch(adc_select) {     // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
767             TBCCR1 = 51680 - ADC24_ISR_TICKS_1000_DIV2_SPS;    //TODO GENERISCH
                TBCCR0 = 54880 - ADC24_ISR_TICKS_1000_DIV2_SPS;
769             break;
              case ADC12_INT_SELECT :
771             TBCCR1 = 51680 - 264;
                TBCCR0 = 54880 - 264;
773             break;
              }
775
            } else {
777             TBCCR1 = 50080;          // 6.26 ms
                TBCCR0 = 53280;          // 6.66 ms
779             BURST_ERROR_FLAG_UNSET;
            }
781
            // Interrupts für TimerB freischalten
783         TIMER_B_0_CM_IRQ_ENABLE;
            TIMER_B_1_CM_IRQ_ENABLE;
785     }break;

787     /*************************
        ** 16MHz; TimerDiv 4
789     ** Periode  -> 10.0 ms
        ** 39200    -> 9.80 ms
791     ** 40800    -> 10.2 ms
        ** 132 (ISR)-> 92us        //TODO NACH MEINER RECHNUNG 368 fuer 92us wohin ist der rest?
793     ** 6534 (ISR AD24) -> 4.553ms
        *************************/
795     case BURST_FREQ_100HZ: {
            TIMER_B_SOURCE_DIV_4;
797
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
799           switch(adc_select) {     // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
801       TBCCR1 = 39200 - ADC24_ISR_TICKS_1000_DIV4_SPS;    //TODO GENERISCH
          TBCCR0 = 40800 - ADC24_ISR_TICKS_1000_DIV4_SPS;
803     break;
              case ADC12_INT_SELECT :
805               TBCCR1 = 39200 - 132;
                  TBCCR0 = 40800 - 132;
807               break;
              }
809         } else {

811           TBCCR1 = 38400;          //  9.6 ms
              TBCCR0 = 40025;          // 10.0 ms
813
               // TBCCR1 = 38400;          //  9.6 ms
815            // TBCCR0 = 40000;          // 10.0 ms
                BURST_ERROR_FLAG_UNSET;
817         }

819         // Interrupts für TimerB freischalten
            TIMER_B_0_CM_IRQ_ENABLE;
821         TIMER_B_1_CM_IRQ_ENABLE;
        }break;
823
        /*************************
825     ** 16MHz; TimerDiv 8
        ** Periode  -> 20.0 ms
827     ** 39600    -> 19.8 ms
        ** 40400    -> 20.2 ms
829     ** 66 (ISR) -> 92us
        *************************/
831     case BURST_FREQ_50HZ: {
            TIMER_B_SOURCE_DIV_8;
833
            if(BURST_ERROR_FLAG_IS_UNSET) { // Wurde ein Fehler dedektiert?
835           switch(adc_select) {     // added 29/09/2014  - EM
              case ADC24_EXT_SELECT :
837             TBCCR1 = 39600 - ADC24_ISR_TICKS_250_SPS;    //TODO GENERISCH
                TBCCR0 = 40400 - ADC24_ISR_TICKS_250_SPS;
839             break;
              case ADC12_INT_SELECT :
841             TBCCR1 = 39600 - 66;
                TBCCR0 = 40400 - 66;
843             break;
              }
845         } else {
                TBCCR1 = 39200;          // 19.6 ms
847             TBCCR0 = 40000;          // 20.0 ms
```

```
                BURST_ERROR_FLAG_UNSET;
849             }

851             // Interrupts für TimerB freischalten
                TIMER_B_0_CM_IRQ_ENABLE;
853             TIMER_B_1_CM_IRQ_ENABLE;
            }break;
855     }
}
857
void timer_b_init(void) {
859     /*************************************************************
        * Timer B initialisierung 13.03.13 NS
861     *************************************************************/

863     // timer clear
        TBCTL = TACLR;
865     // no grouping, counter length = 16bit, clock source = SMCLK, div = 8
        TBCTL |= (TASSEL1 | ID1 | ID0);
867     // Capture/Compare Reg 0 set
        TBCCR0 = 62500;
869     // Compare-mode, IRQ enable
        TBCCTL0 |= CCIE;
871     // Capture/Compare Reg 1 set
        //TBCCR1 = TIME_TO_SAMPLE;
873     // Compare-mode, IRQ enable
        //TBCCTL1 |= CCIE;
875     // Capture/Compare Reg 2 set
        //TBCCR2 = TIME_TO_TX;
877     // Compare-mode, IRQ enable
        //TBCCTL2 |= CCIE;
879 }
```

code/ZSK3V04/timer.c

E.33 timer.h

```
/*******************************************************************************
**    Description:      Timer
**    Hardware:         BATSEN ZS Klasse 3 v0.4 - 09/2014 - EM
**    Date:             13/03/2013
**    Last Update:      29/09/2013 - EM
**    Author:           Nico Sassano
*******************************************************************************/

#ifndef TIMER_H_
#define TIMER_H_

#include "main.h"

/************************************************************
**  Bending Defines for CC430 use EM 31.07.2014
************************************************************/
#define TACCR0 TA0CCR0        // added 29/09/2013 - EM
#define TACCR1 TA0CCR1        // added 29/09/2013 - EM
#define TACCTL0 TA0CCTL0      // added 29/09/2013 - EM
#define TACCTL1 TA0CCTL1      // added 29/09/2013 - EM
#define TACTL TA0CTL         // added 29/09/2013 - EM
#define TAR TA0R           // added 29/09/2013 - EM
#define TAEX0 TA0EX0         // added 29/09/2013 - EM

#define TBCCR0 TA1CCR0        // added 29/09/2013 - EM
#define TBCCR1 TA1CCR1        // added 29/09/2013 - EM
#define TBCCTL0 TA1CCTL0      // added 29/09/2013 - EM
#define TBCCTL1 TA1CCTL1      // added 29/09/2013 - EM
#define TBCLR TACLR          // added 29/09/2013 - EM
#define TBCTL TA1CTL         // added 29/09/2013 - EM
#define TBSSEL_2 TASSEL_2     // added 29/09/2013 - EM

/************************************************************
**  Defines Timer A
************************************************************/
#define TIMER_A_START_UP_MODE         (TACTL |= MC0)
#define TIMER_A_START_CM_MODE         (TACTL |= MC1)
#define TIMER_A_START_UD_MODE         (TACTL |=(MC1 | MC0))

#define TIMER_A_STOP                  (TACTL &= ~(MC0 | MC1))

#define TIMER_A_RESET                 (TACTL = TACLR)
#define TIMER_A_FLAG_RESET0           (TACCTL0 &= ~(CCIFG|COV))

#define TIMER_A_SOURCE_TACLK          (TACTL = TASSEL_0)
#define TIMER_A_SOURCE_ACLK           (TACTL = TASSEL_1)
#define TIMER_A_SOURCE_SMCLK          (TACTL = TASSEL_2)

#define TIMER_A_SOURCE_DIV_8          (TACTL |= (ID1 | ID0))
#define TIMER_A_SOURCE_DIV_4          (TACTL |= ID1); \
                                      (TACTL &= ~ID0);
#define TIMER_A_SOURCE_DIV_2          (TACTL &= ~ID1); \
                                      (TACTL |= ID0);
#define TIMER_A_SOURCE_DIV_1          (TACTL &= ~ID1); \
                                      (TACTL &= ~ID0);

#define TIMER_A_0_OUTMOD_RESET_SET    (TACCTL0 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
#define TIMER_A_1_OUTMOD_RESET_SET    (TACCTL1 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
#define TIMER_A_2_OUTMOD_RESET_SET    (TACCTL2 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))

#define TIMER_A_0_OUTMOD_TOGGLE_SET   (TACCTL0 |= (OUTMOD2 | OUTMOD1)); \
                                      (TACCTL0 &= ~OUTMOD0);
#define TIMER_A_1_OUTMOD_TOGGLE_SET   (TACCTL1 |= (OUTMOD2 | OUTMOD1)); \
                                      (TACCTL1 &= ~OUTMOD0);
#define TIMER_A_2_OUTMOD_TOGGLE_SET   (TACCTL2 |= (OUTMOD2 | OUTMOD1)); \
                                      (TACCTL2 &= ~OUTMOD0);

#define TIMER_A_0_OUTMOD_RESET        (TACCTL0 |= (OUTMOD2 | OUTMOD0)); \
                                      (TACCTL0 &= ~OUTMOD1);
#define TIMER_A_1_OUTMOD_RESET        (TACCTL1 |= (OUTMOD2 | OUTMOD0)); \
                                      (TACCTL1 &= ~OUTMOD1);
#define TIMER_A_2_OUTMOD_RESET        (TACCTL2 |= (OUTMOD2 | OUTMOD0)); \
                                      (TACCTL2 &= ~OUTMOD1);

#define TIMER_A_0_CM_IRQ_ENABLE       (TACCTL0 |= CCIE);
#define TIMER_A_1_CM_IRQ_ENABLE       (TACCTL1 |= CCIE);
#define TIMER_A_2_CM_IRQ_ENABLE       (TACCTL2 |= CCIE);

#define TIMER_A_0_CM_IRQ_DISABLE      (TACCTL0 &= ~CCIE);
#define TIMER_A_1_CM_IRQ_DISABLE      (TACCTL1 &= ~CCIE);
#define TIMER_A_2_CM_IRQ_DISABLE      (TACCTL2 &= ~CCIE);
```

```
83
   /************************************************************
85 **  Defines Timer B
   ************************************************************/
87 #define TIMER_B_START_UP_MODE        (TBCTL |= MC0)
   #define TIMER_B_START_CM_MODE        (TBCTL |= MC1)
89 #define TIMER_B_START_UD_MODE        (TBCTL |=(MC1 | MC0))

91 #define TIMER_B_STOP                 (TBCTL &= ~(MC0 | MC1))

93 #define TIMER_B_RESET                (TBCTL = TBCLR)
   #define TIMER_B_FLAG_RESET           (TBCCTL1 &= ~(CCIFG|COV))
95
   #define TIMER_B_SOURCE_TACLK         (TBCTL = TBSSEL_0)
97 #define TIMER_B_SOURCE_ACLK          (TBCTL = TBSSEL_1)
   #define TIMER_B_SOURCE_SMCLK         (TBCTL = TBSSEL_2)
99
   #define TIMER_B_SOURCE_DIV_8         (TBCTL |= (ID1 | ID0))
101#define TIMER_B_SOURCE_DIV_4         (TBCTL |= ID1); \
                                        (TBCTL &= ~ID0);
103#define TIMER_B_SOURCE_DIV_2         (TBCTL &= ~ID1); \
                                        (TBCTL |= ID0);
105#define TIMER_B_SOURCE_DIV_1         (TBCTL &= ~ID1); \
                                        (TBCTL &= ~ID0);
107
   #define TIMER_B_CM_RISING_EDGE       (TBCCTL0 |= CM1)
109
   #define TIMER_B_0_OUTMOD_RESET_SET   (TBCCTL0 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
111#define TIMER_B_1_OUTMOD_RESET_SET   (TBCCTL1 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
   #define TIMER_B_2_OUTMOD_RESET_SET   (TBCCTL2 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
113
   #define TIMER_B_0_OUTMOD_TOGGLE_SET (TBCCTL0 |= (OUTMOD2 | OUTMOD1)); \
115                                     (TBCCTL0 &= ~OUTMOD0);
   #define TIMER_B_1_OUTMOD_TOGGLE_SET (TBCCTL1 |= (OUTMOD2 | OUTMOD1)); \
117                                     (TBCCTL1 &= ~OUTMOD0);
   #define TIMER_B_2_OUTMOD_TOGGLE_SET (TBCCTL2 |= (OUTMOD2 | OUTMOD1)); \
119                                     (TBCCTL2 &= ~OUTMOD0);

121#define TIMER_B_0_OUTMOD_RESET       (TBCCTL0 |= (OUTMOD2 | OUTMOD0)); \
                                        (TBCCTL0 &= ~OUTMOD1);
123#define TIMER_B_1_OUTMOD_RESET       (TBCCTL1 |= (OUTMOD2 | OUTMOD0)); \
                                        (TBCCTL1 &= ~OUTMOD1);
125#define TIMER_B_2_OUTMOD_RESET       (TBCCTL2 |= (OUTMOD2 | OUTMOD0)); \
                                        (TBCCTL2 &= ~OUTMOD1);
127
   #define TIMER_B_0_CM_IRQ_ENABLE      (TBCCTL0 |= CCIE);
129#define TIMER_B_1_CM_IRQ_ENABLE      (TBCCTL1 |= CCIE);
   #define TIMER_B_2_CM_IRQ_ENABLE      (TBCCTL2 |= CCIE);
131
   #define TIMER_B_0_CM_IRQ_DISABLE     (TBCCTL0 &= ~CCIE);
133#define TIMER_B_1_CM_IRQ_DISABLE     (TBCCTL1 &= ~CCIE);
   #define TIMER_B_2_CM_IRQ_DISABLE     (TBCCTL2 &= ~CCIE);
135

137/**** Prototyp declaration ***********************************************/
   void timer_a_init(void);
139void timer_a_init_balanc(void);
   void timer_a_init_test(void);
141void timer_b_init_burst(uint8_t);
   void timer_b_init_burst_end(uint8_t);
143void timer_b_init(void);

145
   #endif
```

code/ZSK3V04/timer.h

E.34 types.h

```
/*-------------------------------------------------------------------
    Description:    MSP430 (mspgcc) data types.
    Date:           12/02/2012
    Last Update:    12/02/2012
    Author:         Phillip Durdaut
-------------------------------------------------------------------*/

#ifndef TYPES_H_
#define TYPES_H_

/*-------------------------------------------------------------------
  Types
-------------------------------------------------------------------*/

typedef unsigned char       uint8_t;
typedef signed char         sint8_t;
typedef unsigned int        uint16_t;
typedef signed int          sint16_t;
typedef unsigned long       uint32_t;
typedef signed long         sint32_t;
typedef unsigned long long  uint64_t;
typedef signed long long    sint64_t;

typedef unsigned char       u8_t;
typedef signed char         s8_t;
typedef unsigned int        u16_t;
typedef signed int          s16_t;
typedef unsigned long       u32_t;
typedef signed long         s32_t;
typedef unsigned long long  u64_t;
typedef signed long long    s64_t;
#endif /* TYPES_H_ */
```

code/ZSK3V04/types.h

F Spannungslupe

Die Spannungslupe hat die Aufgabe, zunächst eine Offsetspannung abzuziehen. Dabei soll nicht der gesamte Gleichanteil kompensiert werden, da mit dem ADC keine negativen Spannungen aufgenommen werden können. Stattdessen wird die Offsetkorrektur um 20 mV reduziert, sodass ein 40 mV breiter Korridor entsteht. Dieser soll dann mit einer Zweiten Schaltung so Verstärkt werden, dass der 40 mV auf die Möglichen EIngangsspannungen von 0 V bis 2.5 V abgebildet werden.

In Abbildung F.1 ist links die Schaltung zur Offsetkorrekur gezeigt, die einen Operationsverstärker und die drei Widerstände R_1 R_2 und R_3 nutzt. Der Zusammenhang zwischen Eingangspannung $U_[IN]$ und Zwischenspannung U_Z wird im folgenden entwickelt.

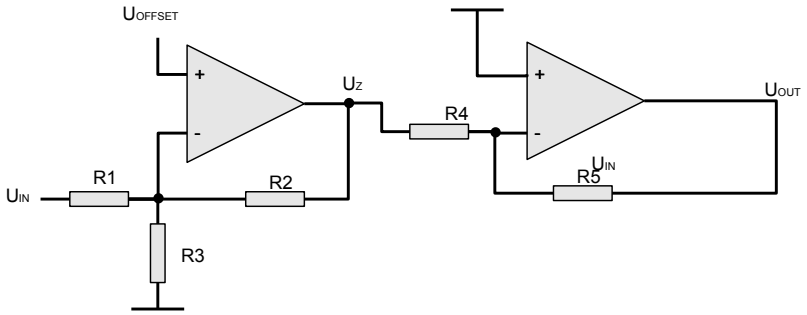


Abbildung F.1: Erster Entwurd der Spannungslupe. Der erste Operationsverstärker reduziert die Eingangsspannung um ein Offset und invertiert es. Der zweite Opamp ist ebenfalls invertierend, und Verstärkt das Signal um den Faktor 62.5.

$$
\begin{aligned}
U_+ &= U_{OFFSET} \\
U_- &= U_{IN}\frac{R_2||R_3}{R_1+R_2||R_3} + U_Z\frac{R_1||R_3}{R_1||R_3+R_2} \\
&= U_{IN}\frac{(R_2R_3)/(R2+R3)}{R_1+(R_2R_3)/(R2+R3)} + U_Z\frac{(R_1R_3)/(R_1+R_3)}{(R_1R_3)/(R_1+R_3)+R_2} \\
&= U_{IN}\frac{R_2R_3}{R_1(R_2+R_3)+R_2R_3} + U_Z\frac{R_1R_3}{R_1R_3+R_2(R_1+R_3)} \\
&= U_{IN}\frac{R_2R_3}{R_1R_2+R_1R_3+R_2R_3} + U_Z\frac{R_1R_3}{R_1R_3+R_1R_2+R_2R_3} \\
U_Z &= U_+ - U_- = 0 \\
0 &= U_{OFFSET} - U_{IN}\frac{R_2R_3}{R_1R_2+R_1R_3+R_2R_3} - U_Z\frac{R_1R_3}{R_1R_3+R_1R_2+R_2R_3} \\
0 &= U_{OFFSET}(R_1R_2+R_1R_3+R_2R_3) - U_{IN}R_2R_3 - U_ZR_1R_3 \\
U_ZR_1R_3 &= U_{OFFSET}(R_1R_2+R_1R_3+R_2R_3) - U_{IN}R_2R_3 \\
U_Z &= U_{OFFSET}\frac{R_1R_2+R_1R_3+R_2R_3}{R_1R_3} - U_{IN}\frac{R_2R_3}{R_1R_3} \\
U_Z &= U_{OFFSET}(\frac{R_2+R_3}{R_3}+\frac{R_2}{R_1}) - U_{IN}\frac{R_2}{R_1}
\end{aligned}
$$

In dieser Stufe soll der die Ruhespannung U_R kompensiert werden Daher soll der Verstärkungsfaktor dieser Stufe -1 betragen, woraus folgt $R_1 = R_2$. Wir legen fest $R_1 = R_2 = 10\,\mathrm{k\Omega}$. Für die Eingangsspannung gilt, dass sie sich aus einem Gleichspannunganteil und einem Wechselspannungsanteil zusammensetzt.

$$U_{IN} = U_R + U_{AC}$$

Für U_R gilt das Werte zwischen 2.6 V und 4 V in Frage kommen. Um eine Einstellbare Spannungsquelle mit einem einfachen Microcontroller zu erhalten, kann eine Pulsweitenmodulation (PWM) mit anschliessender Tiefpassfilterung verwendet werden. Diese Schaltung kann dann Spannungen zwischen 0V und der Logikspannung von 3.3 V erzeugen.

U_R	U_{OFFSET}
2.6 V	0 V
4 V	3.3 V

Tabelle F.1: Ein und Ausgangsspannungen der Offsetkompensation

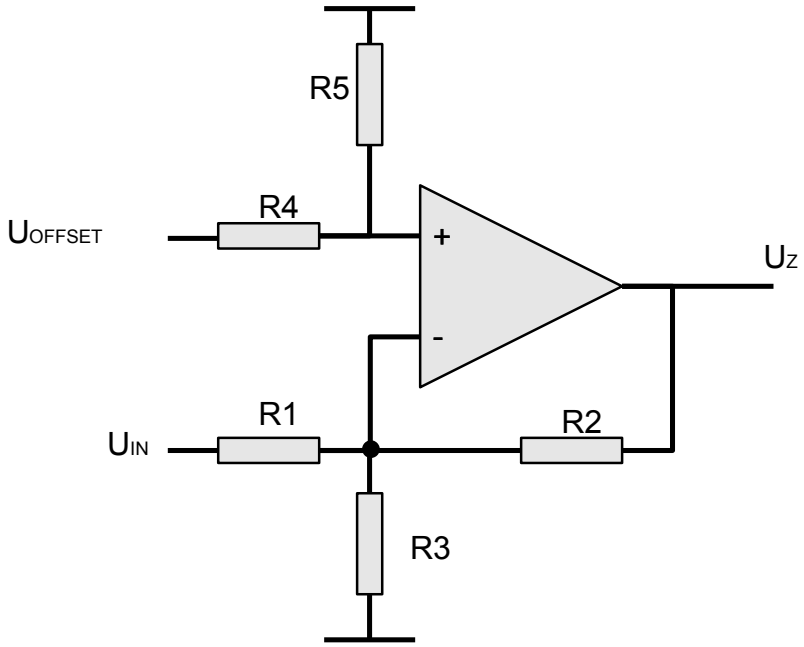


Abbildung F.2: Spannungslupe mit Halbierung des Offsets.

$$
\begin{aligned}
U_{OUT} &= U_{OFFSET}\left(\frac{R_2 + R_3}{R_3} + \frac{R_2}{R_1}\right) \\
U_{OUT} &= U_{OFFSET}\left(\frac{10\,\mathrm{k\Omega} + R_3}{R_3} + 1\right) \\
4\,\mathrm{V} &= 3.3\,\mathrm{V}\left(\frac{10\,\mathrm{k\Omega} + R_3}{R_3} + 1\right) \\
\frac{4\,\mathrm{V}}{3.3\,\mathrm{V}} - 1 &= \frac{10\,\mathrm{k\Omega} + \mathrm{R}_3}{R_3} \\
\overline{0.21}R_3 - R_3 &= 10\,\mathrm{k\Omega}
\end{aligned}
$$

Dieses Gleichungssystem hat keine Lösung die mit realen Widerständen zu realisieren ist, da U_R und U_{OFFSET} zu dicht beieinander liegen. Eine Möglichkeit wäre nun eine Verstärkungsfaktor kleiner als 0.21 zu wählen. Alternativ kann man festlegen, das nicht der gesamte Ausgangsbereich bis 3.3 V Verwendet werden soll. Um dennoch die Auflösung der PWM nutzen zu können, wird die Offsetspannung vor dem Operationsversärker halbiert. Es folgt das in Abbildung F.2 gezeigte Bild für die erste Hälfte des Spannungslupe.

Für den Zusammenhang zwischen Eingangspannung $U_[IN]$ und Zwischenspannung U_Z gilt dann:

$$
\begin{aligned}
U_+ &= \frac{R_5}{R_4 + R_5} U_{OFFSET} \\
U_+ &= \frac{1}{2} U_{OFFSET} \\
U_- &= U_{IN} \frac{R_2 R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3} + U_Z \frac{R_1 R_3}{R_1 R_3 + R_1 R_2 + R_2 R_3} \\
U_Z &= U_+ - U_- = 0 \\
0 &= \frac{1}{2} U_{OFFSET} - U_{IN} \frac{R_2 R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3} - U_Z \frac{R_1 R_3}{R_1 R_3 + R_1 R_2 + R_2 R_3} \\
U_Z &= \frac{1}{2} U_{OFFSET} (\frac{R_2 + R_3}{R_3} + \frac{R_2}{R_1}) - U_{IN} \frac{R_2}{R_1}
\end{aligned}
$$

$$
\begin{aligned}
U_{OUT} &= \frac{1}{2} U_{OFFSET} \left(\frac{R_2 + R_3}{R_3} + \frac{R_2}{R_1} \right) \\
U_{OUT} &= \frac{1}{2} U_{OFFSET} \left(\frac{10\,\mathrm{k\Omega} + R_3}{R_3} + 1 \right) \\
4\,\mathrm{V} &= 1.65\,\mathrm{V} \left(\frac{10\,\mathrm{k\Omega} + R_3}{R_3} + 1 \right) \\
\frac{4\,\mathrm{V}}{1.65\,\mathrm{V}} - 1 &= \frac{10\,\mathrm{k\Omega} + \mathrm{R}_3}{R_3} \\
1.\overline{42} R_3 - R_3 &= 10\,\mathrm{k\Omega} \\
0.\overline{42} R_3 &= 10\,\mathrm{k\Omega} \\
\frac{14}{33} R_3 &= 10\,\mathrm{k\Omega} \\
\rightarrow R_3 &\approx \frac{330}{14}\,\mathrm{k\Omega}
\end{aligned}
$$

Daraus folgt dann:

$$
\begin{aligned}
U_{OUT} &= U_{OFFSET} \left(\frac{R_2 + R_3}{R_2} + \frac{R_3}{R_1} \right) - U_Z \frac{R_3}{R_1} \\
U_{OUT} &= U_{OFFSET} \left(\frac{\frac{330}{14}\,\mathrm{k\Omega} + 10\,\mathrm{k\Omega}}{\frac{330}{14}\,\mathrm{k\Omega}} + \frac{10\,\mathrm{k\Omega}}{10\,\mathrm{k\Omega}} \right) - U_Z \frac{10\,\mathrm{k\Omega}}{10\,\mathrm{k\Omega}} \\
U_{OUT} &= U_{OFFSET} \frac{80}{33} - U_Z;
\end{aligned}
$$

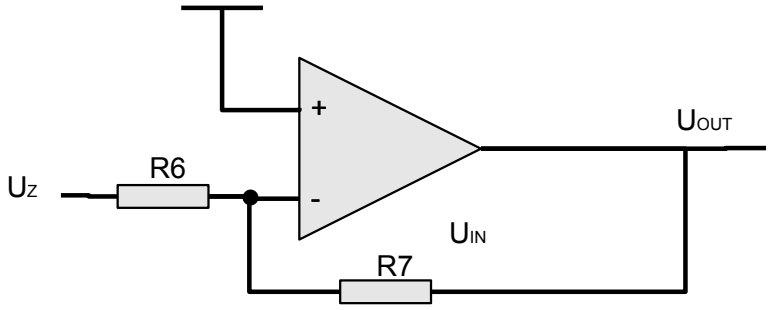


Abbildung F.3: Zweite Stufe der Spannungslupe.

Die Zweite Stufe ist in

Für die Zweite Stufe gilt:

$$
\begin{aligned}
U_+ &= 0 \\
U_- &= U_Z \frac{R_5}{R_4 + R_5} + U_{OUT} \frac{R_4}{R_4 + R_5} \\
U_D &= 0 \\
&= U_+ - U_- \\
0 &= -U_Z \frac{R_5}{R_4 + R_5} - U_{OUT} \frac{R_4}{R_4 + R_5} \\
U_Z \frac{R_5}{R_4 + R_5} &= -U_{OUT} \frac{R_4}{R_4 + R_5} \\
U_Z &= -U_{OUT} \frac{R_4 (R_4 + R_5)}{R_5 (R_4 + R_5)} \\
U_Z &= -U_{OUT} \frac{R_4}{R_5}
\end{aligned}
$$

Der Spannungsbereich, auf den die Eingangsspannung abgebildet werden soll ist immer 0 V bis 2.5 V, die Eingangsspannung liegt aufgrund der Invertierung durch die Stufe der Offsetkompensation -40 mV.

U_{AC}	U_{OUT}
0 mV	0 V
-40 mV	2.5 V

Tabelle F.2: Ein und Ausgangsspannungen der Spannungslupe

Woraus folgt:

$$-40\,\mathrm{mV} = -2.5\,\mathrm{V}\frac{R_6}{R_7}$$
$$\frac{-40\,\mathrm{mV}}{2.5\,\mathrm{V}} = -\frac{R_6}{R_7}$$
$$-0.016 = -\frac{R_6}{R_7}$$

Durch Festlegung von $R_5 = 100\,\mathrm{k\Omega}$ folgt $R_4 = 1.6\,\mathrm{k\Omega}$

Der entwickelte Schaltplan wurde in LT-Spice realisiert und simuliert (vgl. Abbildung F.4 auf der nächsten Seite). Die simulation zeigt, das für alle Ruhespannungen innerhalb des Intervalls das Wechselspannungsignal ideal auf den gewünschten Spannungsbereich abgebildet wird. Allerdings ist zu beachten, das die Zwischenspannung negativ ist. Sie liegt bei bis zu -40 mV. Aus diesem Grund muss eine negative Spannungsversorgung dieser Größenordnung bereitgestellt und ein Rail-to-Rail-Operationsverstärker eingesetzt werden.

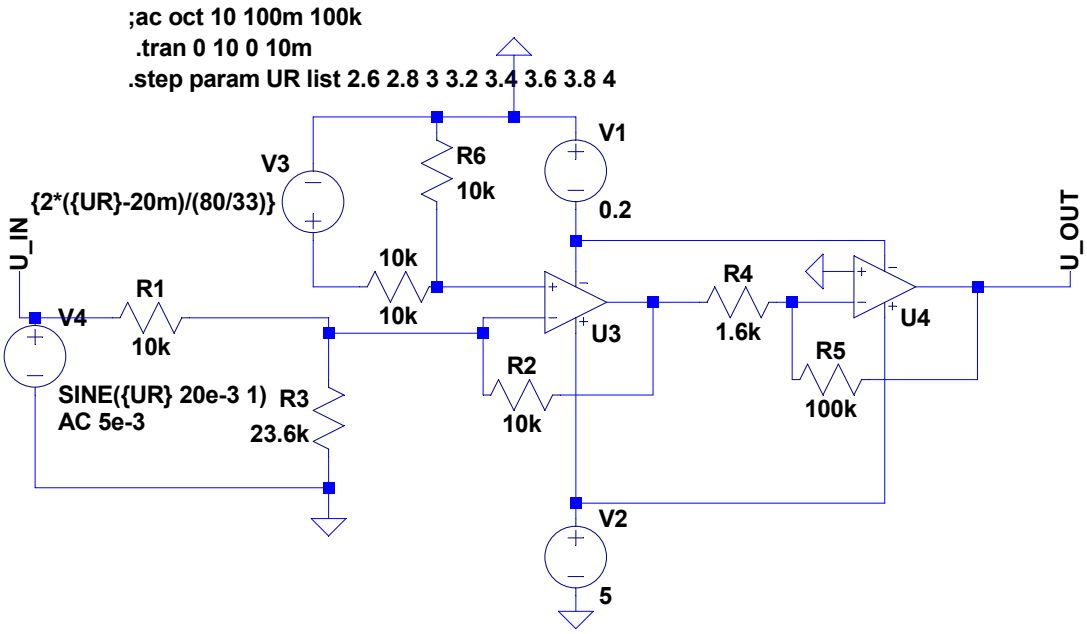


Abbildung F.4: Spice-Modell der Spannungslupe. Die Offsetspannung zur Spannungskorrektur wird hier ohne die bei der PWM entstehenden Quantisierung eingespeisst.

G Batteriemodel für Beispielhafte EIS-Messung

Um für die Darstellung der EIS im Grundlagenkapitel Abschnitt 2.2 auf Seite 44 Daten nutzen zu können wurde auf Grundlage der in Abschnitt 2.1.4 auf Seite 34 erstellten Ersatzschaltbilder ein Model erstellt, das im Relevanten Frequenzbereich zwischen 10 mHz und 10 kHz ein stark inhomogenes Verhalten aufweist, dessen Form an die einer realen Batterie angelehnt ist. Das Modell ist in Abbildung G.1 auf der nächsten Seite inklusiver der verwendeten Parameter abgebildet. LT-Spice bietet keine Möglichkeit die Ordinate zu invertieren (vgl. Abbildung G.2 auf der nächsten Seite), weshalb alle Abbildungen in den entsprechenden Kapiteln mit den aus LT-Spice Exportierten Daten in GNU Octave gemacht wurden.

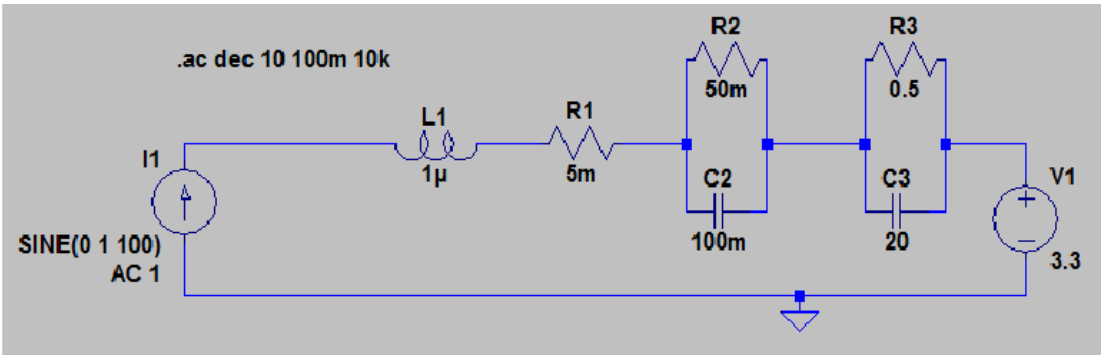


Abbildung G.1: Beispielmodell eines Akkumulators in LT-Spice. Die Spule wirkt durch ihre niedrige Induktivität erst bei Frequenzen mehrer Kiloherz deutlich. Die Abszisse wird beim Wert von Rs geschnitten, und die Kapazitäten der Beiden Kondensatoren unterscheidet sich ausreichend Stark, sodass zwei Bögen entstehen.

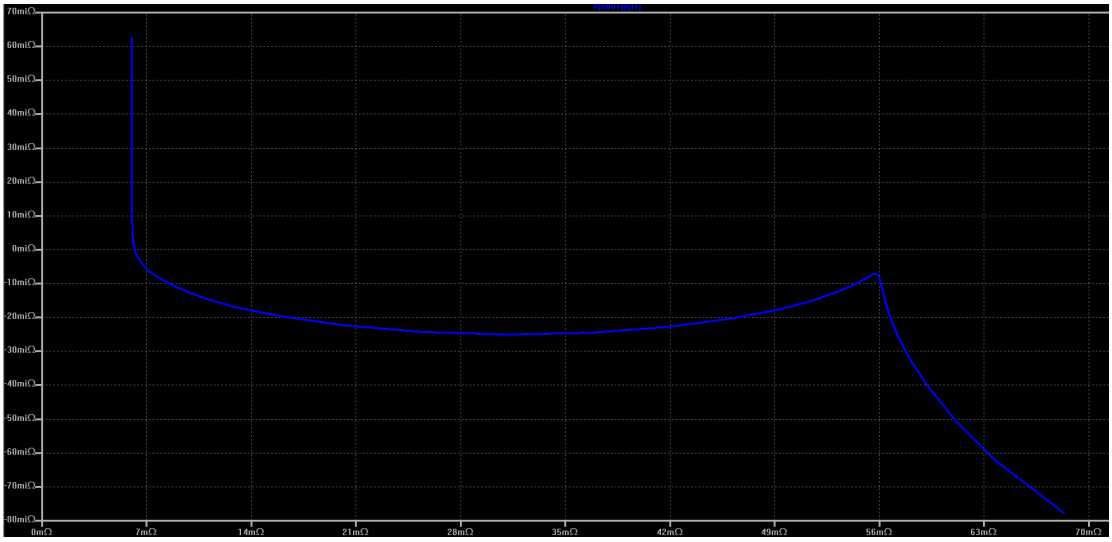


Abbildung G.2: Nyquist-Diagram des LT-Spice Models. Hier ist die Ordinate anders als bei der EIS üblich nicht invertiert.

H Erweiterungsmodul für den Zellsensor Klasse 3 Version 4

In diesem Abschnitt sollen kurz die Möglichkeiten aufgezeigt werden, die mit der Erweiterungsmodul-Schnittstelle auf dem neuen Zellesensor genutzt werden können.

In der vorliegenden Arbeit wird über den Pfostenstecker mit dem ADS1291 über SPI kommuniziert. Dafür wurden die Leitungen des 'universal serial communication interface (USCI)' mit vier Pins des Erweiterungsmodul-Steckers verbunden. Die Pingbelegung wurde bereits in Abbildung 3.13 auf Seite 95 gezeigt, und der Quellcode für die Konfiguration kann der Datei ADC24.c im Anhang E entnommen werden.

Da das USCI auch andere Arten synchroner und asynchroner Kommunikation unterstützt und außerdem über die Port-Map-Matrix eine Reihe weiterer Funktionen mit dem Erweiterungsmodul-Stecker verbunden werden kann, soll im folgenden kurz vorgestellt werden, welche Anwendungsmöglichkeiten sich bieten.

H.1 Asynchrone Kommunikation

Das USCI unterstützt asynchrone Kommunikation, wie sie beispielsweise das RS232-Protokol verwendet. Dabei ist darauf zu achten, das diese Kommunikation mit der Logikspannung des CC430 durchgeführt wird, sodass eventuell Level-Shifter notwengig sind, um eine externe Peripherie anzuschließen. Abbildung H.1 auf der nächsten Seite zeigt die Pinbelegung des Erweiterungsmoduls für Asynchrone Kommunikation. Da nur Sende- und Empfangsleitung und kein Taktsignal benötigt werden, verbleiben zwei frei Konfigurierbare Pins auf dem Erweiterungsmodul-Stecker, die über die Pins P1.7 und P5.0 angesprochen werden können. Da die SPI-Schnittstelle des WakeUp-Receivers immernoch mit den Leitungen Verbunden ist, die jetzt als UART-Schnittstelle genutzt werden soll, muss sichergestellt werden, das dieser Chip während der Kommunikation die Signale auf den Leitungen nicht beachtet. Dazu wird die Kommunikation über das Chip-Select Signal des WakeUp-Receivers abgeschaltet. Das Folgende Listing zeigt die Konfiguration der Schnittstelle für eine Baudrate von 115200 $^{Bit}/_{s}$ und die Funktionen für eine Kommunikation über UART.

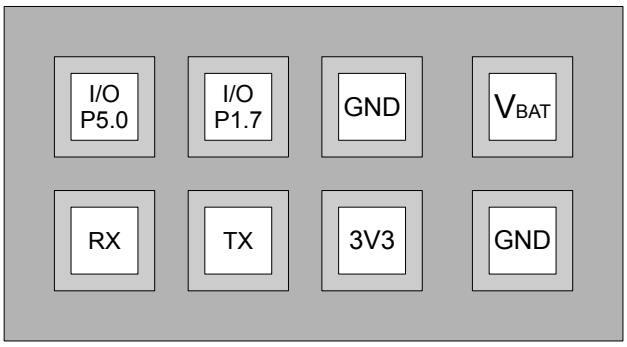


Abbildung H.1: Pinbelegung des Pfostensteckers für das Erweiterungsmodul für eine Kommunikation über UART in der Draufsicht.

```
void uart_huckepack_init(void) {
    P5DIR |= BIT1;                  // AS3930 CS is output
    P5OUT &= ~BIT1;                 // AS3930 Chip disable

    P1DIR |= BIT6;                  // Set P1.6 as TX output
    P1SEL |= BIT5 + BIT6;           // Select P1.5 & P1.6 to UART function

    UCA0CTL1 |= UCSWRST;            // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;           // SMCLK @ 1MHz
    UCA0BR0 = 0x09;                 // 1MHz/115200 = 8.6
    UCA0BR1 = 0x00;                 // CLK prescaler = (UCA0BR0 + UCA0BR1 * 265) = 3
    UCA0MCTL = 0x06;                // Modulation
    UCA0CTL1 &= ~UCSWRST;           // **Initialize USCI state machine**
    UCA0IE &= ~(UCTXIE + UCRXIE);    // Disable USCI_A0 TX/RX interrupt
}

void uart_huckepack_write(char txData) {
    while (!(UCA0IFG&UCTXIFG)); // wait for TXBUF to be empty
        UCA0TXBUF = txData;     // Write Data to TXBUF
}

char uart_huckepack_read(void) {
    UCA0IE |=  UCRXIE;          // Enable USCI_A0 RX interrupt
    while (!(UCA0IFG&UCRXIFG)); // wait for complete character in rxBuf
        return UCA0RXBUF;       // Read character automaticly resets IFG
}
```

Listing H.1: UART Schnittstelle für das Erweiterungsmodul

H.2 I²C

Die I²C Schnittstelle belegt wie UART lediglich zwei Pins, sodass die selben Pins P1.7 und P5.0 zur freien Verfügung bleiben.

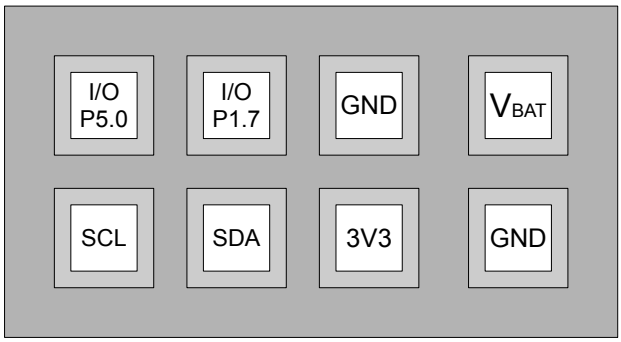


Abbildung H.2: Pinbelegung des Pfostensteckers für das Erweiterungsmodul für eine Kommunikation über I2C in der Draufsicht.

```
1   void i2c_huckepack_init(void)   {
        P1DIR |= BIT5 + BIT6;            // Set P1.5 & P1.6 as output
3       P1SEL |= BIT5 + BIT6;            // Select P1.5 & P1.6 to I2C function
        UCA0CTL1 |= UCSWRST;             // Reset the USCI logic
5       UCA0CTL0 = (UCMST | UCMODE1 | UCMODE0 | UCSYNC);// Master-Mode | I2C-Mode
     | Sync-Mode | 7-Bit Adresses
        UCA0CTL1 = (UCSSEL1 | UCSWRST); //I2CCLK = SMCLK @ 1MHz
7       UCA0BR0 = 0x0A;                  // CLK prescaler = (UCA0BR0 + UCA0BR1 *
    265) = 10
        UCA0BR1 = 0x00;                  // 1MHZ / 10 = 100kHz
9       UCA0I2COA |= 0x0077;             // Own 7-Bit Adress
        UCA0I2CIE = 0x00;                // Disable interrupts
11  }

13  void i2c_write(uint8_t slave_address, uint8_t data_length, uint8_t *data)   {
        uint8_t i;
15
        UCA0CTL1 |= UCTR;                // set transmitter-mode
17      UCA0I2CSA = slave_address;       // set slave address
        UCA0CTL1 &= ~UCSWRST;            // unset SWRESET
19      UCA0CTL1 |= UCTXSTT;             // send START-CON
        UCA0TXBUF = data[0];             // then write data
21
        for(i = 1; i < data_length; i++) {
23          while(1) { // wait until ...
                if((IFG2 & UCA0TXIFG) == UCA0TXIFG) //start-con was send
25              {
```

```
                    UCA0TXBUF = data[i];              //then write data
27                  break;
                }
29          }
        }
31
        while((IFG2 & UCA0TXIFG) != UCA0TXIFG); // wait until data was send
33      UCA0CTL1 |= UCTXSTP;                    // send STOP-COND
        while((UCA0CTL1 & UCTXSTP) == UCTXSTP); // wait until STOP-con was send
35      UCA0CTL1 |= UCSWRST;                    // set SWRESET
    }
37
    void i2c_read(uint8_t slave_address, uint8_t data_length, uint8_t *data)    {
39      uint8_t i;

41      UCA0CTL1 &= ~UCTR;                      // reset transmitter-mode
        UCA0I2CSA = slave_address;              // set slave address of sensor
43      UCA0CTL1 &= ~UCSWRST;                   // unset SWRESET
        UCA0CTL1 |= UCTXSTT;                    // send START-CON
45
        if(data_length > 1) {
47          for(i = 0; i < data_length; i++)    {
                while(1)    {
49                  if((IFG2 & UCA0RXIFG) == UCA0RXIFG) // start-con was send
                    {
51                      data[i] = UCA0RXBUF;    // readout data
                        if(i == data_length-2)  // if next byte will be the last
    one
53                      UCA0CTL1 |= UCTXSTP;    // send STOP-CON after next
    receive
                        break;
55                  }
                }
57          }
        }
59      else
        {
61          while((UCA0CTL1 & UCTXSTT) == UCTXSTT); // wait for acknowledge of
    slave,
            UCA0CTL1 |= UCTXSTP;                // then send STOP-COND
    immediately
63          data[0] = UCA0RXBUF;                // readout data
        }
65      while((UCA0CTL1 & UCTXSTP) == UCTXSTP); // wait until STOP-con was send
        UCA0CTL1 |= UCSWRST;// set SWRESET
67  }
```

Listing H.2: I2C Schnittstelle für das Erweiterungsmodul

H.3 PWM zur Offsetkompensation

In dieser Arbeit wurde beschlossen die Messung von kleinen Wechselspannungen bei großem Gleichspannungsoffset durch Verwendung eines hochauflösenden ADCs zu erreichen. Sollte zukünftig doch eine Aufbereitung des Signals wie Bspw. mit der Spannungslupe realisiert werden, so könnte die Erweiterungsmodul-Schnittstelle weiterhin genutzt werden. Um eine Variable Offset-Korrektur zu realisieren, ist es notwendig den durch die Spannungslupe zu kompensierenden Spannungswert an das Erweiterungsmodul weiterzugeben. Da der CC430 über keinen Digital-Analog-Konverter verfügt, müsste dies über eine PWM geschehen, die auf dem Erweiterungsmodul zu filtern ist, um einen stabilen Spannungswert zu erhalten. Da die Eingänge des ADC nicht über die Port-Map verbunden werden können, ist mit der aktuellen Version des Zellensensors ein eigener niedrig auflösender ADC auf das Erweiterungsmodul notwendig. Sollte in Zukunft ein Redesign des Zellensensors anstehen, so sollte überlegt werden, ob das ChipSelect Signal des das Erweiterungsmodul mit einem Pin des Port 2 Verbunden werden kann, da diese mit den Eingängen des internen 12 Bit ADC verbunden werden können. Aktuell müsste eine Möglichkeit gewählt werden, die Analog-Digital-Umsetzung auf das Erweiterungsmodul durchzuführen, und die Werte bspw. per I2C zu Übertragen. Das Folgenden Listing zeigt die Ausgabe eines 1 kHz PWM-Singals mit einer Einschaltdauer, die sich in 1024 Schritten einstellen lässt. Bei idealer Filterung des Signals entspricht dies einem 10 Bit DAC.

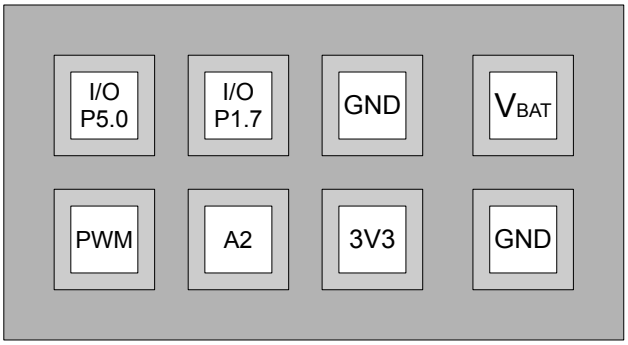


Abbildung H.3: Pinbelegung des Pfostensteckers für das Erweiterungsmodul für eine Offsetkompensation mittels PWM .

```
1
2  void huckepack_init(void)   {
3      P5DIR |= BIT1;                          // AS3930 CS is output
4      P5OUT &= ~BIT1;                         // AS3930 Chip disable
5
6      PMAPPWD = 0x02D52;                      // Get write-access to port mapping
       regs
7      P1MAP7 = PM_TA1CCR1A;                   // Map TA1CCR1 output to P1.5
8      PMAPPWD = 0;                            // Lock port mapping registers
9
10     P1DIR |= BIT7;                          // P1.7 output
11     P1SEL |= BIT7;                          // P1.7 options select
12
13     TA1CCR0 = 1024;                         // PWM Period: 1MHz/1024 ~ 1kHz
14     TA1CCTL1 = OUTMOD_7;                    // CCR1 reset/set
15     TA1CCR1 = 0;                            // CCR1 PWM duty cycle
16     TA1CTL = TASSEL_2 + MC_1 + TACLR;       // SMCLK, up mode, clear TAR
17 }
18
19 void huckepack_write_offset(uint16_t Offset) {
20     TA1CCR1 = Offset>>6;                    // PWM On-Time. 1024 possible steps
21 }
```

Listing H.3: PWM für das Erweiterungsmodul

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit 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.

Hamburg, 27. November 2014

Ort, Datum

Unterschrift