Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Oliver Alexander Schmidt

Auslegung und Erprobung eines Lithiumbatterie-Prüfstands für typische Lade- und Lastprofile von Elektrobussen

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Oliver Alexander Schmidt

Auslegung und Erprobung eines Lithiumbatterie-Prüfstands für typische Lade- und Lastprofile von Elektrobussen

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
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. Jürgen Vollmer

Abgegeben am 15. Juni 2015

Oliver Alexander Schmidt

Thema der Bachelorthesis

Auslegung und Erprobung eines Lithiumbatterie-Prüfstands für typische Lade- und Lastprofile von Elektrobussen

Stichworte

Batterie-Prüfstand, Lithium-Ionen-Batterien, Lade- und Lastprofile von Elektrobussen, Elektrobuszyklus, Elektrischer Stadtbus, Innovationslinie 109 der Hamburger Hochbahn, EDDA-Elektrobus des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme IVI in Dresden

Kurzzusammenfassung

Im Rahmen der Forschungsprojekte BATSEN und BEEDeL wird das Verhalten der Batterien von Elektrobussen im Stadtverkehr analysiert. Aus der Messung der Fahrdynamik der Busse soll der Batteriestrom und der Ladezustand abgeleitet werden. Das Modell wird mit Daten des Kooperationspartners Fraunhofer IVI verifiziert. Die entsprechenden Profile dienen zum Test des dafür angepassten Prüfstandes für Lithium-Ionen-Batterien.

Oliver Alexander Schmidt

Title of the paper

Dimensioning and Test of a Lithium-Ion Battery Test Stand for typical Charge and Load Profiles of Electrical Busses

Keywords

Battery test stand, lithium-ion batteries, load and charge profiles of electric busses, Plug-in-electrical-city-bus, Hamburger Hochbahn innovation-line 109, electrical Bus EDDA of the Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme IVI in Dresden

Abstract

The behaviour of the batteries in electrical city busses is investigated within the research projects BATSEN and BEEDeL. Goal is the derivation of a relation between the measured vehicle dynamics and the battery current and thereby the state of charge. The model is verified by data from the project partner Fraunhofer IVI. A test stand for lithium-ion batteries is specially adapted for the profiles and thereby tested.

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Bachelorarbeit beigetragen haben.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Karl-Ragmar Riemschneider und Herrn Prof. Dipl.-Ing Jürgen Vollmer für die Möglichkeit der Mitarbeit in ihrer Forschungsgruppe, die vielen interessanten Diskussionen und die engagierte Betreuung dieser Arbeit.

Weiterhin danke ich dem Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme für die Bereitstellung der Elektrobusdaten, die dieser Thesis einen großen Schub gegeben haben.

Ebenso bedanke ich mich bei Herrn Dipl.-Phys. Valentin Roscher und B.Eng. Sergej Pereguda für ihre fachliche Unterstützung. Der Austausch mit ihnen hat mir sehr geholfen.

Weiterer Dank gilt Herrn Dipl.-Ing. Günter Müller für sein großes Engagement hinsichtlich der formellen Hinweise und Anmerkungen.

Abschließend danke ich meiner Familie für jegliche Unterstützung während des gesamten Studiums.

Inhaltsverzeichnis

1. Einführung **9**

2. Analyse der Rahmenbedingungen **12**
2.1. Lithium-Ionen-Batterien . . . 12
2.1.1. Aufbau einer Lithium-Ionen-Batterie . . . 14
2.1.2. Kenngrößen einer Lithium-Ionen-Batterie . . . 15
2.1.3. Kennlinien einer Lithium-Ionen-Batterie . . . 18
2.1.4. Die Testbatterien . . . 20
2.2. Beschreibung der im Feldtest laufenden Elektrobusse . . . 21
2.2.1. Elektrobusse der Hochbahn Innovationslinie 109 . . . 21
2.2.1.1. Volvo 7900 Electric Hybrid . . . 21
2.2.1.2. Volvo 7900 Articulated Hybrid . . . 23
2.2.2. EDDA-Bus . . . 24
2.3. Fahrdynamik eines Elektrobusses . . . 25
2.3.1. Entwicklung der Beschleunigungsgleichung . . . 25
2.3.2. Die aus der Beschleunigungsgleichung resultierende Leistungsgleichung . . . 29

3. Konzeption des Elektrobuszyklus **31**
3.1. Messmethoden zur Aufnahme des Fahrverhaltens . . . 31
3.1.1. Das Smartphone zur Aufnahme von Bus-Beschleunigungsdaten . . . 32
3.1.1.1. Beschreibung des Messsystems . . . 33
3.1.1.2. Probebetrieb des Smartphones als Datenlogger . . . 34
3.1.1.3. Fazit . . . 37
3.1.2. Das Messsystem mit 3-Achsen Beschleunigungssensor zur Aufnahme der Bus-Beschleunigungsdaten . . . 37
3.1.2.1. Der 3-Achsen Beschleunigungsaufnehmer . . . 38
3.1.2.2. Probebetrieb des Beschleunigungs-Sensors und des Brüel&Kjær Mess-Systems . . . 40
3.1.2.3. Fazit . . . 41
3.1.3. EDDA-Messdaten des Fraunhofer Instituts . . . 42
3.1.3.1. Aufbereiten der Messdaten . . . 42
3.1.3.2. Fazit . . . 44

3.2. Entwurf und Erstellung von Elektrobuszyklen für den Prüfstand 44
3.2.1. Entwurf und Erstellung eines Zyklus auf Basis der statistischen Verteilung der gemessenen Beschleunigungen (Zyklus-I) 45
3.2.1.1. Analyse und statistische Verteilung der gemessenen Beschleunigungen 46
3.2.1.2. Anpassung der statistischen Daten an die fünf Laststufen des Prüfstandes 47
3.2.1.3. Festlegungen der Ströme für die einzelnen Lastpfade/Relaisstellungen 49
3.2.1.4. Berechnung der Geschwindigkeiten aus den gemessenen Beschleunigungen 50
3.2.2. Entwurf und Erstellung eines Zyklus aus den EDDA-Daten (Zyklus-E) 50
3.2.2.1. Ermittlung der Leistungsprofile 50
3.2.2.2. Ermittlung des Leistungsprofils aus den gemessenen Strömen und Spannungen 54
3.2.2.3. Erstellung des EDDA-Zyklus aus den gemessenen Strömen 56
3.2.2.4. Festlegungen der Ströme für die einzelnen Lastpfade/Relaisstellungen 59

4. Konzeption der Prüfstandsanpassungen **60**
4.1. Das Zykliergerät 60
4.2. Weitere Komponenten des Prüfstandes 61
4.2.1. Der Temperaturschrank 61
4.2.2. Lastwiderstände und aktive Lasten 62
4.2.3. Die Spannungs- bzw. Stromquellen 63
4.3. Vorhergehende Arbeiten über den Aufbau des Zykliergerätes 63
4.4. Inbetriebnahme der Zykliergeräte 65
4.4.1. Untersuchung des Zeitverhaltens der Lastrelais 66
4.4.2. Untersuchung des Zeitverhaltens der Strom- und Spannungsmessungen im Zykliergerät 69
4.4.3. Bestimmung der Lastpfad-Innenwiderstände 70
4.5. Anforderungen an das Zykliergerät 71
4.5.1. Spannungskalibrierung 71
4.5.2. Neuauswahl der Temperatursensoren 72
4.5.3. Vergrößerung der Konfigurations-Datei 72
4.5.4. Anpassung des Zeitverhaltens der Lastrelais-Steuerung und der Messsteuerung 73

5. Prüfstandsanpassung **74**
5.1. Softwareanpassungen für das Zykliergerät 74

5.1.1. Ergänzungen und Erweiterungen von Kalibrierfunktionen im Zykliergerät . . . 74
5.1.2. Implementierung der 1-Wire Temperaturmessung . . . 79
5.1.3. Erhöhung der Taktrate für die Lastrelais-Umschaltung sowie der Strom- und Spannungsmessung . . . 79
5.1.4. Implementierung von Loops und Erweiterung der Konfigurationsdatei 80
5.1.5. Die Anpassungen für den Externen ADC . . . 81
5.2. Hardwareanpassungen im Zykliergerät . . . 81
5.2.1. Neuaufbau des Zykliergerätes . . . 81
5.2.2. Ergänzung einer Schaltung für die 1-Wire-Temperaturmessung . . . 82
5.3. Mechanischer Aufbau des Prüfstandes . . . 84
5.3.1. Beschreibung des Zyklierprüfstandes . . . 84
5.3.2. Bau und Anpassung der Batterieboxen . . . 85

6. Probebetrieb und Durchführung **87**
6.1. Kalibrierung des Zykliergerätes . . . 88
6.2. Untersuchung der Batteriezellen . . . 91
6.2.1. Untersuchung der LiNANO-Batteriezelle . . . 92
6.2.1.1. Ermittlung der Ruhespannung . . . 92
6.2.1.2. Ruhekennlinie der LiNANO-Batteriezelle . . . 92
6.2.1.3. Konstantstromkennlinien der LiNANO-Batteriezelle . . . 93
6.2.2. Untersuchung der A123-Batteriezelle . . . 95
6.2.2.1. Ruhekennlinie der A123-Batteriezelle . . . 95
6.2.2.2. Hochstromtest an der A123-Batteriezelle . . . 95
6.2.3. Untersuchung der ECC-Batteriezelle . . . 98
6.2.3.1. Konstantstromkennlinien der ECC-Batteriezelle . . . 98
6.2.3.2. Zyklierung der ECC-Batteriezelle im linearen Bereich . . . 99
6.3. Probebetrieb des Zyklus-I und Zyklus-E auf dem Prüfstand mit den $LiFePO_4$ Batterien . . . 101
6.3.1. Probebetrieb des Zyklus-I . . . 101
6.3.1.1. Einmaliger Durchlauf des Zyklus-I mit der LiNANO-Zelle . 101
6.3.1.2. Mehrfache Durchläufe des Zyklus-I mit der LiNANO-Zelle . 104
6.3.1.3. Mehrfache Durchläufe des Zyklus-I mit jeweiligen Zwischenladungen der LiNANO-Zelle . . . 106
6.3.1.4. Einmaliger Durchlauf des Zyklus-I mit der A123-Zelle . . . 108
6.3.1.5. Mehrfache Durchläufe des Zyklus-I mit jeweiligen Zwischenladungen der A123-Zelle . . . 110
6.3.2. Probebetrieb des Zyklus-E . . . 112

7. Fazit **115**
7.1. Zusammenfassung . . . 115

7.2. Ausblick 117

Tabellenverzeichnis **120**

Abbildungsverzeichnis **121**

Literaturverzeichnis **126**

A. Aufgabenstellung **131**

B. Ergänzungen **134**
B.1. Zellsimulator 134
B.2. Messaufbau zur Bestimmung des Frequenzganges 135
B.3. Prinzipschaltbild EDDA 137
B.4. Flussdiagramm des Aufrufs der Kalibrierfunktion 138
B.5. Nassi-Shneiderman-Diagramm der Kalibrierfunktion 139
B.6. Fehlersuche am Zykliergerät 141

C. Abbildungen **144**

D. MATLAB-Skripte **161**

E. Quellcode Zykliergerät **168**

Abkürzungsverzeichnis **293**

1. Einführung

Elektromobilität ist ein wichtiges Element einer klimagerechten Energie- und Verkehrspolitik, denn sie trägt zu weniger CO2-Emissionen bei. Klimaneutralität liegt allerdings nur vor, wenn der Strom aus erneuerbaren Energiequellen genutzt wird. Die deutsche Bundesregierung fördert seit einigen Jahren Elektromobilität mit großem finanziellen Engagement. Seit 2010 gibt es die Nationale Plattform Elektromobilität (NPE) als ein Beratungsgremium der Bundesregierung zur Elektromobilität. Ziel ist es, Deutschland bis 2020 nicht nur zum Leitmarkt, sondern auch zum Leitanbieter für Elektromobilität zu machen [8].

Bis 2020 möchte auch die Stadt Hamburg ihr gesamtes Linienbusnetz auf einen emissionsfreien Betrieb umstellen, um weniger Lärm und CO2 zu emittieren und zudem die Abhängigkeit von fossilen Energieträgern zu reduzieren. 2014 startete BEEDeL (Bewertung des Einsatzes von Elektrobussen mit Dezentraler Ladeinfrastruktur in Metropolen am Beispiel der Hochbahn), ein Kooperationsprojekt der Hamburger Hochbahn, der Hochschule für Angewandte Wissenschaften (HAW), des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme (IVI) und des Verbands Deutscher Verkehrsunternehmen (VDV). Die Hamburger Hochbahn testet im Rahmen des Projektes seit Dezember 2014 auf der Innovationslinie 109 alle zurzeit relevanten alternativen Antriebstechnologien. Dazu gehören auch diesel-elektrische Hybridbusse sowie elektrische Plug-In Busse, deren Batterien an den Endhaltestellen aufgeladen werden können.
Das Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme hat ebenfalls einen elektrischen Plug-In Bus (EDDA [1]) zwischen November 2014 und April 2015 auf einer Linie in Dresden getestet.
Reine Batteriebusse sind noch nicht ausgereift. Die begrenzte Ladekapazität der Batterien, gemessen als Ladezustand SOC (State of Charge), führt zu geringen Reichweiten der Busse und schränkt damit die Flexibilität des Fahrzeugeinsatzes ein. Außerdem haben Batterien nur eine eingeschränkte Lebensdauer. Der Alterungsprozess kann jedoch durch angepasste Lade- und Entladekonzepte herausgezögert werden. Die Bestimmung des State of Health (SOH) von Batterien ist ein aktuelles und komplexes Forschungsthema.

[1] Elektromobilitäts-Demonstration Docking-Anwendung

Abbildung 1.1.: Der Volvo 7900 Electric Hybrid der Innovationslinie 109 an der Endhaltestelle Hauptbahnhof/ZOB in Hamburg beim Ladevorgang.

Die Ladehäufigkeit und die Energieaufnahme ist abhängig vom gewichtsbedingten Energiebedarf des Fahrzeuges, der zurückgelegten Strecken und weiteren Kriterien wie Topographie der Linien und klimatischen Bedingungen, den sogenannten Orts-Geschwindigkeits-Profilen. Die tatsächlichen Anforderungen an den Energiebedarf können daher sehr unterschiedlich sein.

Aussagen über die notwendige Batterieleistung können zurzeit nur durch detaillierte Messungen der Fahrprofile der Busse auf der jeweiligen Linie gemacht werden. Diese recht aufwendigen Messungen führen zu realen Einsatzprofilen, die die Basis für Batterietestzyklen sind. Sie liefern Hinweise auf das Alterungsverhalten der Zellen in Abhängigkeit der Entladetiefe (Vollzyklen bzw. Teilzyklen mit Zwischenladungen) sowie der Ladegeschwindigkeit. Bei Elektro- und Hybridbussen besteht aktueller Forschungsbedarf, da über spezifische Betriebsprofile und die entsprechenden Batteriebelastungen noch wenig bekannt ist [16].

Von 2011 bis März 2015 wurden in der Forschungsgruppe drahtlose Zellsensoren für Fahrzeugbatterien (BATSEN) an der HAW Hamburg, gefördert durch das Bundesministerium für Bildung und Forschung (BMBF), Blei- und Lithiumbatterien für verschiedene Einsatzbereiche untersucht. Ziel war die Erarbeitung von Grundlagen sowie die Implementierung und Anwendung eines drahtlosen Sensornetzes für die Überwachung des Ladezustandes von mehrzelligen großen Fahrzeugbatterien. Typische Anwendungsprofile wurden auf einem Laborprüfstand mit gespeicherten Lade- und Lastverläufen getestet (zykliert). Die Forschungsgruppe BATSEN kooperiert seit April 2015 mit dem Projekt BEEDeL.
Seit April 2015 ist die ehemalige Forschungsgruppe BATSEN Teil des Projekts BEEDeL. Im Rahmen der Kooperation ist diese Arbeit mit folgender Aufgabenstellung entstanden:

Für Elektrobus-Batterien soll ein Prüfstand entwickelt werden, der typische Betriebsprofile (Eletrobuszyklen) simulieren kann. Dieser soll in der Entwicklung von Batterien und von Elektrobussen eingesetzt werden, ohne dass für Versuche bzw. Experimente aufwendige Versuchsfahrten durchgeführt werden müssen.
Zwei vorhandene Zykliergeräte für einen Batterie-Prüfstand dienen als Basis und sollen in einem gesetzten Rahmen (Anzahl der Schaltstufen, Schaltgeschwindigkeiten und Steuerungshardware) optimiert und erweitert werden. Neben der Prüfstands-Überarbeitung sollen Betriebsprofile von realen Elektrobussen ermittelt, und auf ihre Übertragbarkeit auf den Prüfstand untersucht werden. Notwendige Messungen erfolgen auf der Innovationslinie 109 der Hamburger Hochbahn. Im Verlauf der Arbeit hat sich ergeben, dass weitere Messdaten von Betriebsprofilen einer Buslinie in Dresden, welche vom Fraunhofer-Institut ermittelt wurden, zur Verfügung stehen.

Für die Simulation der Betriebsprofile auf dem Prüfstand muss der Zusammenhang zwischen Fahrparametern wie z. B. Beschleunigung, Geschwindigkeit und der Batterieleistung (Batterieströme) untersucht und ermittelt werden.
In einem Probebetrieb des optimierten Prüfstandes werden die Funktionen und verschiedene Einsatz-Möglichkeiten aufgezeigt. Dabei werden die Ermittlung von Batteriekenndaten (Ruhespannung, Ladekennlinie usw.) anhand dreier Batterietypen und die Simulation von zwei Buszyklen dargestellt.

2. Analyse der Rahmenbedingungen

2.1. Lithium-Ionen-Batterien

Um Fahrzeuge mit Verbrennungsmotoren auf elektrisch betriebene Fahrzeuge umzustellen, sind Energiespeicher mit einer hohen Energie- und Leistungsdichte sowie langer Lebensdauer nötig. Lithium-Ionen-Batterien haben eine hohe Energiedichte, wie auch aus dem Ragone Diagramm Abb. 2.1 hervorgeht. Allerdings haben Lithium-Ionen-Batterien im Vergleich zu anderen Speicherformen wie z. B. Superkondensatoren oder Membranspeichern nur eine geringe Leistungsdichte (siehe Abb. 2.2). Sie können nur langsam und kontinuierlich gebzw. entladen werden. Dies ist auf den Innenwiderstand der Batterie zurückzuführen. Die Entladezeit T_{Ent} kann anhand der Division des Energiegehaltes E_N durch die spezifische Leistung P_N der Zelle, wie in Formel (2.1) dargestellt, bestimmt werden [18].

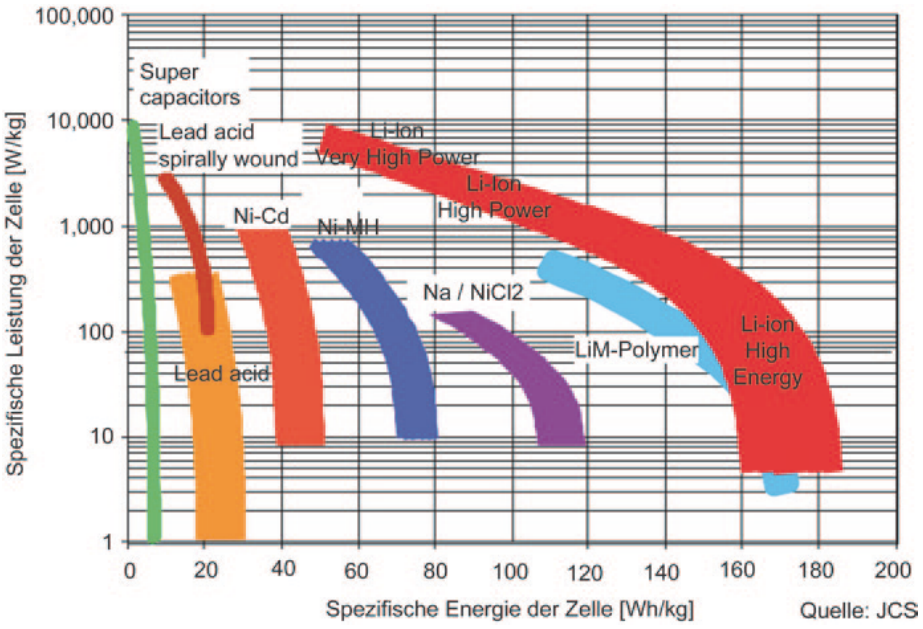


Abbildung 2.1.: Ragone-Diagramm für elektrische Energiespeicher. [18]

$$T_{Ent} = \frac{E_{Bat}}{P_{Bat}} \tag{2.1}$$

Bei schnellem Lastwechsel, wie z. B. beim Bremsen von Fahrzeugen, kann dies zu einem Energieüberschuss führen. Der Batterie ist es nicht möglich, die durch das Bremsen gewonnene Bremsenergie vollständig zu speichern. Die überschüssige Energie müsste in diesem Fall über Bremswiderstände in Wärme umgewandelt oder in anderen Speichertechnologien wie Superkondensatoren gespeichert werden [18]. Auch unter wirtschaftlichen Gesichtspunkten sind Lithium-Ionen-Batterien interessant, da sie sich durch folgende Merkmale auszeichnen:

- eine hohe Energiedichte
- eine geringe Leistungsdichte
- einem guten Wirkungsgrad
- einer relativen Temperaturstabilität
- vollkommen recycelbar

Ein großes Thema ist auch die Alterung bzw. die Lebensdauer der einzelnen Zellen, da diese bis Dato noch wenig erforscht ist.

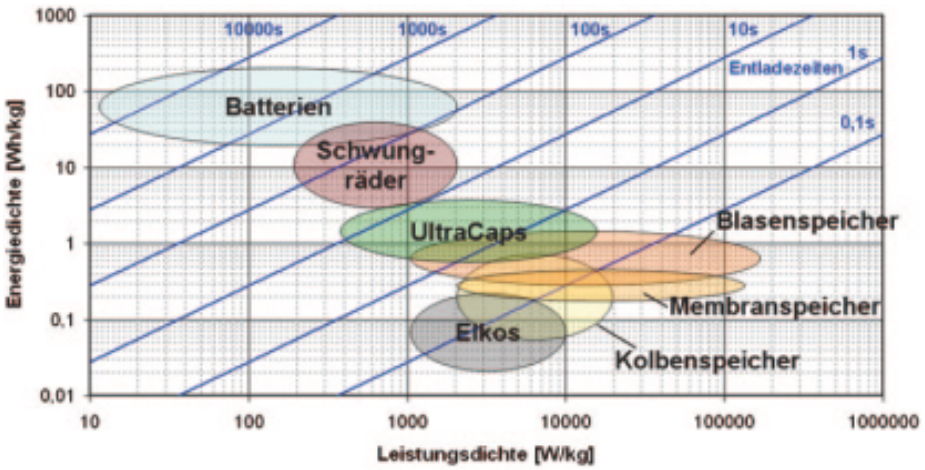


Abbildung 2.2.: Ragone Diagramm für verschiedene Energiespeichertechnologien [18]

2.1.1. Aufbau einer Lithium-Ionen-Batterie

Die Batterie fällt unter die elektrochemischen Energiespeicher, bei denen zwischen Primär- und Sekundärzellen zu unterscheiden ist. Sekundärzellen sind im Vergleich zu Primärzellen weitestgehend reversibel. Sie können im Gegensatz zu Primärzellen wieder aufgeladen werden. Eine Sekundärzelle besteht aus einer Anode (unedleres Metall) und einer Kathode (edleres Metall), die sich in einem Elektrolyten befinden, wie in Abb. 2.3 zu sehen. Das Elektrolyt ist nur durchlässig für Ionen, für Elektronen ist es undurchlässig. Die Konsistenz des Elektrolyten kann flüssig, fest oder geliert sein. Die Elektroden (Anode, Kathode) dienen als Energiekonverter und Energiespeicher, welche durch einen möglichst dünnen und ionenleitfähigen Separator getrennt sind [18]. Des Weiteren bildet sich während der ersten Zyklen der Zelle (Formatierung) eine Lithium-Schicht auf der Anode, die ein weiteres Zersetzen des Elektrolyten verhindert (Solid Electrolyte Interface, SEI) [25]. Während der Lebensdauer einer Zelle nimmt die Dicke dieser Schicht weiter zu wodurch der ohmsche Innenwiderstand der Zelle steigt. Die abgesetzten Lithium-Ionen stehen nicht weiter zur elektrochemischen Reaktion zur Verfügung, welches die Abnahme der Zellkapazität zur Folge hat. Die Zelle altert [24].

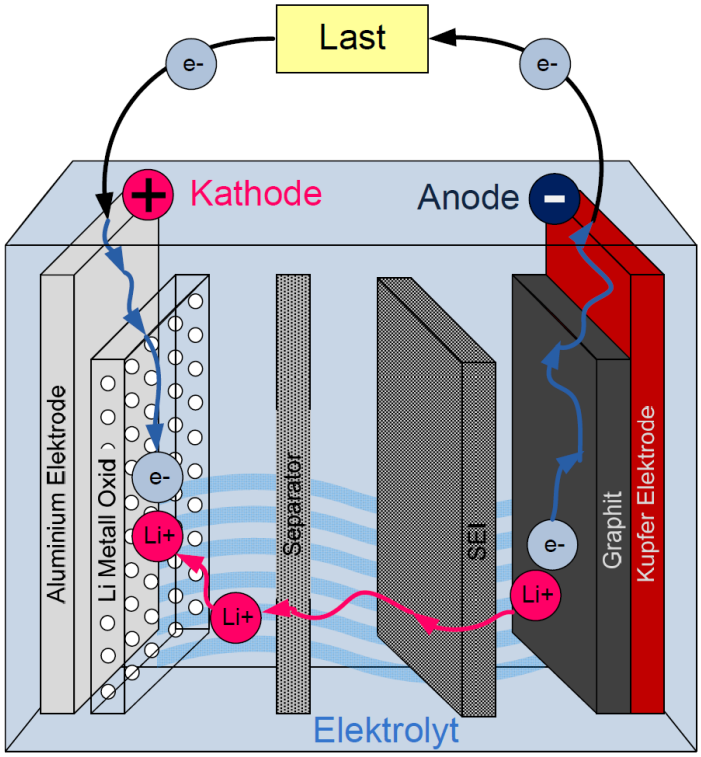


Abbildung 2.3.: Aufbau und Funktion einer Lithium-Ionen-Zelle [31]

Beim Entladen einer Batterie fließen die Elektronen von der Anode über die Last zur Ka-

thode. Durch die Oxidation des unedleren Metalls der Anode und der Reduktion am edleren Metall der Kathode entsteht ein Elektronenfluss. Ein Ionenaustausch über den Elektrolyten ist notwendig, damit ein kontinuierlicher Strom fließen kann. Die Spannung, die zwischen den Elektroden entsteht, ist abhängig von der elektrochemischen Spannung der gewählten Materialien-Paare (siehe Abb. C.1). Die Differenz der elektrochemischen Spannung ergibt die Zell-Nennspannung U_N [18]. Das Anlegen einer externen Spannung an die Elektroden lässt den chemischen Prozess umdrehen, woraufhin die Zelle geladen wird.

In den Bereichen der Elektromobilität kommen folgende Batteriedesigns zum Einsatz [31]:

- Rundzellen
- Prismazellen
- Pouchzellen (Coffee-Bag)

2.1.2. Kenngrößen einer Lithium-Ionen-Batterie

Die Kenngrößen einer Lithium-Ionen-Batterie werden wie folgt definiert:

- Nennspannung U_N

  Die Nennspannung einer Zelle ist abhängig von den verwendeten Elektrodenmaterialien, siehe Abb. C.1 [18].

- Leerlaufspannung U_0

  Die Leerlaufspannung einer Batterie wird im unbelasteten Zustand gemessen. Sie hängt vom SOC der Batterie ab und verhält sich, wie in Abb. 2.5 zu sehen, abschnittsweise linear [18].

- Ruhespannung U_R

  Die Ruhespannung kann nur im unbelasteten Zustand der Batterie gemessen werden und ergibt sich, wenn die inneren Ausgleichsvorgänge der Batterie abgeschlossen sind und sich ein chemisches Gleichgewicht eingestellt hat. Sie lässt sich über temperaturabhängige Zeitfunktionen aus der Leerlaufspannung ermitteln und ist somit auch von dem State of charge (SOC) der Batterie abhängig [18]. Laut [20] kann die Ruhespannung einer Batterie bei Raumtemperatur nach ca. 4 Stunden gemessen werden.

- Klemmspannung U_K

  Die Klemmspannung wird direkt an den Klemmen der Batterie gemessen. Sie kann von der Nennspannung der Batterie abweichen [18].

- Klemmstrom I_K

  Der Klemmenstrom ist der Strom, der in die Batterie hinein bzw. hinaus fließt. Er kann extern mit einem Amperemeter gemessen werden.

- Entladeschlussspannung $U_{\mathrm{dischargeMin}}$

  Die Entladeschlussspannung ist vom Hersteller angegeben und gibt beim Entladen der Batterie an, ab welcher Klemmspannung die Batterie nicht weiter entladen werden darf [18].

- Ladeschlussspannung $U_{\mathrm{chargeMax}}$

  Die Ladeschlussspannung ist die maximale Spannung, die beim Laden der Batterie nicht überschritten werden darf [18].

- Energiegehalt E_N

  Der Energiegehalt einer Zelle ergibt sich aus dem Produkt der Nennspannung und der Nennkapazität [20].

  $$E_N = U_N \cdot Q_N \tag{2.2}$$

- Kapazität Q_{Bat}

  Die Kapazität einer Zelle wird in Amperestunden $\mathrm{A\,h}$ angegeben. Sie kann alternativ auch in Coulomb $\mathrm{C} = \mathrm{A\,s}$ angegeben werden. Die Kapazität einer Zelle hängt sehr von den Entladebedingungen wie z. B. der Temperatur, der Entladeschlussspannung, der Belastung, wie auch von der Vorgeschichte der Zelle dem SOH ab [18].

- Nennkapazität Q_N

  Die Nennkapazität oder auch Nominalkapazität wird meist mit definierten Entladebedingungen, wie der Temperatur und der C-Rate, mit der die Zelle entladen wird, angegeben [18].

- Ladezustand SOC

  Der SOC (State of charge) gibt den Ladezustand der Batterie im Vergleich zur Nennkapazität einer Zelle in Prozent an [18].

- Allgemeinzustand der Batterie SOH

  Der SOH (State of health) gibt den Zustand einer Zelle im Vergleich zu einer neuwertigen Zelle in Prozent an. Dadurch lässt sich die Rest-Lebensdauer einer Zelle feststellen [18].

- Entladetiefe DOD

  Der DOD (Depth of discharge) gibt die auf die Nennkapazität bezogene Entladetiefe an. Dementsprechend stellen $100\,\%$ eine vollständig entladene Batterie dar [18].

- Innenwiderstand R_i

  Der Innenwiderstand einer Batterie wird nach [18] mit der Formel 2.3 definiert. In dieser Formel wird die Temperatur, der SOC und das Alter der Zelle vernachlässigt. Das dazugehörige Ersatzschaltbild ist unter Abb. 2.4 abgebildet.

$$R_i = \frac{U_R - U_K}{I_K} \tag{2.3}$$

  In dem zweiten abgebildeten Ersatzschaltbild Abb. 2.4 rechts ist zu sehen, dass der Innenwiderstand vom Ladezustand SOC abhängig ist. [18]. In anderen Batterie-Modellen geht der Innenwiderstand als Impedanz ein und wird über die elektrochemische Impedanzmessung bestimmt. In dieser Arbeit wird nur mit den unter Abb. 2.4 dargestellten vereinfachten Ersatzschaltbildern gearbeitet.

- C-Rate

  Die C-Rate einer Zelle ergibt sich aus der Division der Nennkapazität durch eine Stunde. Dies bedeutet, dass bei '1C Entladung' die Zelle nach einer Stunde vollständig entladen ist.

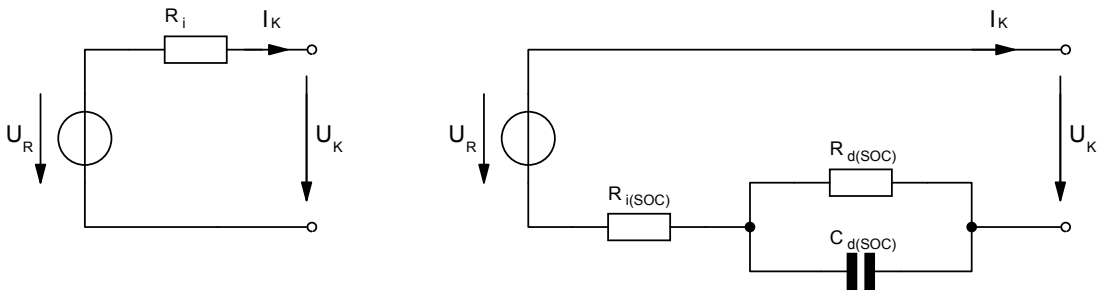


Abbildung 2.4.: Stark vereinfachtes lineares Ersatzschaltbild einer Batterie links und das vereinfachte Ersatzschaltbild einer Batterie in Abhängigkeit des SOC rechts. Die Ersatzschaltbilder sind abgeleitet[1] aus [18].

[1] Die Beschriftung wurde zur Vereinheitlichung der Variablen angepasst.

2.1.3. Kennlinien einer Lithium-Ionen-Batterie

Eine Herausforderung bei Lithium-Ionen-Batterien, insbesondere bei einer Lithium-Eisenphosphat $LiFePO_4$, ist die flache Ruhespannungskennlinie. Obwohl die Kennlinie nahezu linear verläuft, ist der Hub gering wie in Abb. 2.5 zu sehen. Dies erfordert für die Bestimmung des Ladezustandes über die Ruhespannung eine hohe Messauflösung sowie -genauigkeit und ist daher fehleranfällig.

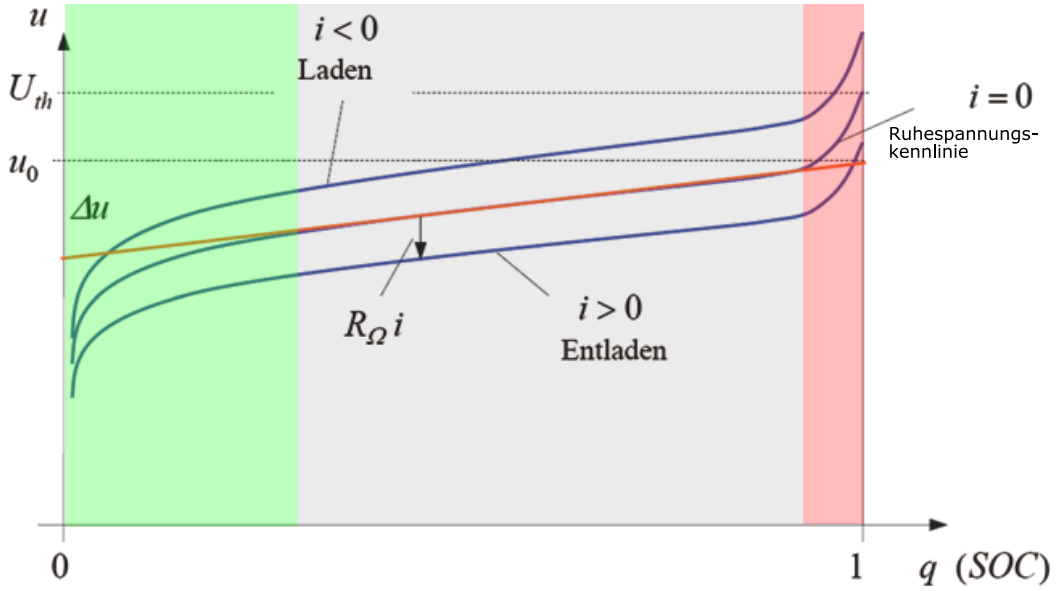


Abbildung 2.5.: Typisches Verhalten der Batteriespannung in Abhängigkeit vom Ladegrad und Strom [18][2]bei einer Lithium-Ionen-Batterie.

Wird eine Lithium-Ionen-Batterie geladen oder entladen, weicht die Klemmspannung von der Ruhespannung ab. Im Falle der Ladung (negativer Strom I_K) ist die Klemmspannung erhöht, da durch den Strom eine zusätzliche Spannung am Innenwiderstand der Zelle abfällt. Im linearen Bereich der Kennlinie (der graue Bereich der Abb. 2.5) gilt das linke Ersatzschaltbild Abb. 2.4. Die Klemmspannung berechnet sich dann aus der Ruhespannung und dem Spannungsabfall über den Innenwiderstand U_i(2.4) [18].

$$U_K = U_R - I_K \cdot R_i = U_R - U_i \tag{2.4}$$

Im linearen Bereich ändert sich die Zellspannung als Funktion des SOC nur wenig. Kurz vor erreichen der Ladeschlussspannung steigt die Klemmspannung unter Last stark an (roter Bereich), da die Zellchemie keine weitere Ladung mehr aufnehmen kann.

[2] Die Grafik wurde farbig ausgefüllt und die Beschriftung angepasst: die Leerlaufkennline wurde zu Ruhespannungskennlinie.

Analog dazu sinkt die Klemmspannung beim Entladen (positiver Strom I_K) unter die Ruhespannung, da über den Innenwiderstand der Zelle eine Gegenspannung abfällt.

Ist die Zelle beinahe entladen, bricht die Zellspannung ein, da die Zellchemie keine ausreichende Ladung mehr liefern kann (grüner Bereich).

Das Lade- und Entladeverhalten der Zelle kann auch durch eine Spannungshysterese unter Last beschrieben werden, die den Wirkungsgrad einschränkt. Dies ist auch in Abb. C.2 zu sehen. Der Wirkungsgrad ist durch die Formel (2.5) gegeben [18].

$$\eta_{\text{Speicher}} = \frac{\text{entnehmbare Energie}}{\text{geladene Energie}} = \frac{I_{\text{Entlade}} \cdot \int_0^{\text{Entladezeit}} U_{\text{Entlade}} \cdot \mathrm{d}t}{I_{\text{Lade}} \cdot \int_0^{\text{Ladezeit}} U_{\text{Lade}} \cdot \mathrm{d}t} \tag{2.5}$$

Da die Klemmspannung stark vom Ladestrom bzw. Entladestrom abhängt, ist die maximale Ladung bzw. Entladung auch von den Strömen abhängig. Dies gilt, wenn die Ströme wie in Abb. 2.6 gezeigt, konstant sind. Eine vollständige Ladung kann über ein Ladeverfahren mit adaptivem Strom, beispielsweise IU-Ladeverfahren (CC-CV), erfolgen [20].

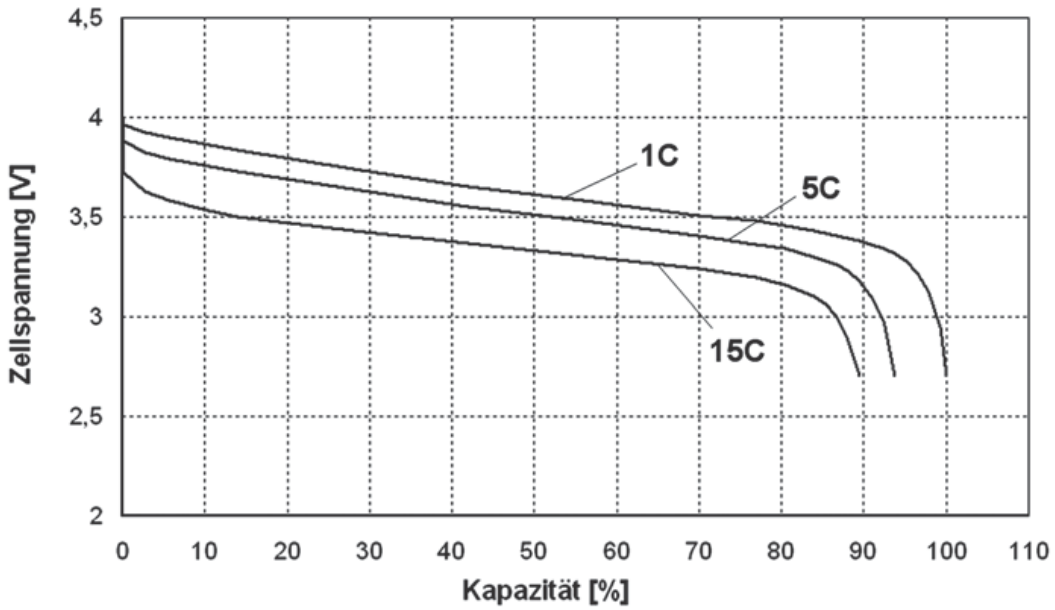


Abbildung 2.6.: Abhängigkeit der Zellspannung und Kapazität vom Entladestrom für eine Lithium-Ionen-Batterie [18]

Wie eingangs bereits beschrieben, sollen Batterien, die in Elektrofahrzeugen eingebaut werden, auch die durch einen Bremsvorgang gewonnene Energie speichern (Rekuperation). Für

eine Rekuperation muss allerdings die Bedingung erfüllt sein, dass die Zelle nicht vollständig geladen sein darf. Der Ladezustand der Batterie (SOC) muss unter einer bestimmten Obergrenze liegen, die in der Literatur mit 65 bis 80 % SOC angegeben wird [18]. Zu beachten ist, dass die maximale Rekuperationsleistung von der maximal aufnehmbaren Leistung der Batterie begrenzt wird. Überschüssige Leistungen müssen über Bremswiderstände oder konventionelle mechanische Bremsen in Wärme umgewandelt werden. Um Leistungsreduktionen zu vermeiden, sollten Lithium-Ionen-Batterie nicht zu tief entladen werden. In der Literatur wird der Betrieb der Batterie in ihrem linearen Bereich empfohlen. Laut Fachliteratur kann 56 % der theoretisch verfügbaren Bremsenergie in der Batterie gespeichert werden [18].

2.1.4. Die Testbatterien

Als Testbatterien stehen in dieser Arbeit drei verschiedene Lithium-Eisenphosphat ($LiFePO_4$) Batterietypen zu Verfügung. Diese zeichnen sich durch unterschiedliche Kapazitäten aus. In der Tabelle 2.1 sind die Kennwerte aus den Datenblättern kurz zusammengefasst.

Tabelle 2.1.: Kennwerte der Batterien

Hersteller	ECC [14]	LiNANO [22]	A123 Systems [21]
Typ	ECC-LFPP 42 A h	Headway 40152SE	ANR26650M1B
Nennspannung	3,3 V	3,2 V	3,3 V
Spezifischer Energiegehalt	134,4 W h bei 0,2C und 25 °C	48 W h	7,59 W h
Nominalkapazität	42 A h bei 0,2C und 25 °C	15 A h	2,5 A h
Innenwiderstand bzw. Impedanz	0,5 bis 1,2 mΩ bei 1 kHz	<5 mΩ	n. a.
Spezifische Energiedichte	120,5 Wh/kg	100 Wh/kg	103 Wh/kg
Ladeschlussspannung	3,65 V	3,65 V	3,6 V
Entladeschlussspannung	2,6 V	2,5 V	2,0 V
Maximaler Entladestrom	3C	5C	20C
Maximaler Ladestrom	3C	3C	4C
Anzahl der verfügbaren Zellen	2	4	1

2.2. Beschreibung der im Feldtest laufenden Elektrobusse

Im Rahmen dieser Arbeit werden Simulationszyklen für einen Batterieprüfstand aus dem Fahrverhalten der Elektrobusse der Innovationslinie in Hamburg und dem EDDA-Bus des Fraunhofer-Instituts für Verkehrs- und Infrastruktursysteme entworfen. In den Abschnitten 2.2.1 und 2.2.2 werden die verschiedenen Busse beschrieben.

2.2.1. Elektrobusse der Hochbahn Innovationslinie 109

Die Hamburger Hochbahn hat sich zum Ziel gesetzt, ab 2020 nur noch emissionsfreie Busse zu beschaffen. Dies wird mit der sinkenden Verfügbarkeit von fossilen Treibstoffen, den wachsenden Anforderungen an die Luftqualität bzw. den reduzierten CO_2 Emissionen und auch der Lärmreduktion begründet. Da es unterschiedliche verkehrliche Rahmenbedingungen wie z. B. verschiedene Einsatzorte, unterschiedliche Linienlängen und die Einhaltung des Fahrplans für die Busse gibt, wurde Anfang Dezember 2014 eine Innovationslinie als Testfeld für verschiedene Bustypen eingerichtet [12]. Die Buslinie 109 ist $9{,}3\,\mathrm{km}$ lang und verläuft zwischen den Endhaltestellen Hauptbahnhof/ZOB und U-Alsterdorf (siehe Abb. C.3). Es werden folgende fünf Innovationsbustypen auf dieser Linie getestet [6]:

- Der Brennstoffzellenbus
- Der Batteriebus mit Brennstoffzelle als Range-Extender
- Der parallele diesel-elektrische Hybridbus
- Der serielle diesel-elektrische Hybridbus
- Der Plug-in-Bus

Im Rahmen dieser Arbeit wird das neue Plug-in-Bus Konzept der Hochbahn betrachtet. In den folgenden Abschnitten werden der Plug-in-Bus (Volvo 7900 Electric Hybrid) und sein großer Bruder (Volvo 7900 Articulated Hybrid), der allerdings nicht Plug-in-fähig ist, kurz beschrieben.

2.2.1.1. Volvo 7900 Electric Hybrid

Bei dem Volvo 7900 Electric Hybrid handelt es sich um einen, in Abb. 2.7 abgebildeten, schnellladefähigen Parallelhybridbus. Zusätzlich zu seinem Dieselmotor hat der Bus noch eine permanenterregte Synchronmaschine, die sowohl generatorisch wie auch motorisch betrieben werden kann. Über eine Kupplung können die beiden Motoren gemeinsam oder

auch unabhängig voneinander eingesetzt werden [18]. Wenn der Bus rein elektrisch gefahren wird und die Batteriekapazität bzw. ein Schwellenwert überschritten wird, wechselt der Bus in den Hybridmodus [40].

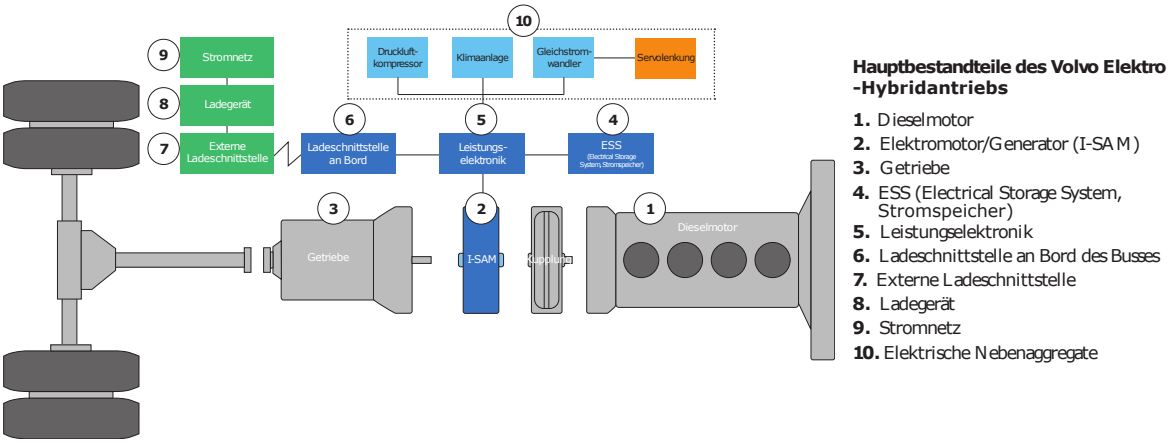
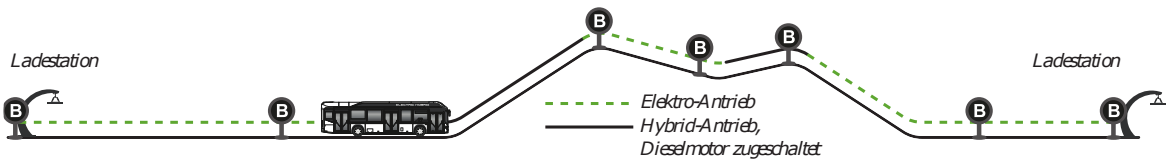


Abbildung 2.7.: Schematische Darstellung des Volvo 7900 Electric Hybrid Antriebs [40]

Nach dem Datenblatt [42] des Volvo 7900 Electric Hybrid besitzt der Bus eine 600 V Lithium-Ionen-Batterie mit einer Kapazität von 19 kW h. Die rein elektrische Reichweite beträgt zwischen 2 km und 5 km, nach Hochbahnangaben sogar 7 km [17]. Der Bus wird an den Endhaltestellen über die am Dach befestigte Ladeschienen (siehe Abb. C.4) geladen. Dazu fährt der Bus unter die von der Firma Siemens entwickelten Ladestation (Abb. C.5). Ein Pantograph senkt sich von der Ladestation auf die Ladeschienen des Busses herab und lädt den Bus 6 min lang auf. Zu den 6 min Ladezeit wird noch ein Puffer von 4 min aufgeschlagen. Daraus resultiert eine Standzeit von 10 min je Endhaltestelle [33]. Im Falle einer Verspätung wird die Standzeit verkürzt, um die Haltestellen laut Fahrplan anfahren zu können (Fahrplantreue). Wenn der 4 min Puffer bereits ausgereizt wurde, kann dies zu einer Verkürzung der Ladezeit und kann somit bei rein elektrischem Betrieb zu einer kürzeren Reichweite führen. Der Ladestand der Batterie wird dem Busfahrer über eine Ladeanzeige in der Konsole angezeigt, wie in Abb. C.6 dargestellt.



Dies ist ein Beispiel für den Betrieb von Volvo Elektro-Hybridbussen, bei dem die Distanz zwischen den Ladestationen ca. 10 km beträgt. Die Aufladung an den Ladestationen ermöglicht einen vollelektrischen Fahrbetrieb auf ca. 70-80 % der Strecke (abhängig von Betriebszyklen, Topografie und Nebenverbrauchern).

Abbildung 2.8.: Exemplarische Fahrt eines Volvo 7900 Electric Hybrid Busses [40]

Abb. 2.8 zeigt wie der Bus rein elektrisch fahren, und in welchen Situationen der Hybridantrieb (Diesel zugeschaltet) laufen muss. Die wichtigsten Kenndaten aus dem Datenblatt [42] wurden in der Tabelle 2.2 zusammengefasst.

Tabelle 2.2.: Kennwerte des Volvo 7900 Electric Hybrid [42]

Hersteller	Volvo
Typenbezeichnung	7900 Electric Hybrid
Gesamtlänge	12,134 m
Gesamthöhe	3,28 m
Gesamtbreite	2,55 m
Zulässiges Gesamtgewicht	19 t
Maximale Geschwindigkeit	n. a.
Höchstleistung E-Motor	150 kW
Motorleistung Diesel	177 kW
Energiespeichertechnologie	Lithium-Ionen-Batterie
Batterie Nennspannung	etwa 600 V
Energiegehalt der Bus Batterie	19 kW h
Zeitbedarf Schnellladung	max. 6 min
Max. Schnellladeleistung	max. 150 kW

2.2.1.2. Volvo 7900 Articulated Hybrid

Der Volvo 7900 Articulated Hybrid ist der großen Bruder des unter 2.2.1.1 beschriebenen Volvo 7900 Electric Hybrid. Bei dem Volvo 7900 Articulated Hybrid handelt es sich um einen Gelenkbus (siehe Abb. C.8), der allerdings nicht schnellladefähig und nur mit einer kleinen Batterie ausgestattet ist. Die Antriebskomponenten sind baugleich zu denen des Volvo 7900 Electric Hybrid. In der Tabelle 2.3 sind die Kenndaten des Busses zusammengefasst.

Tabelle 2.3.: Kennwerte des Volvo 7900 Articulated Hybrid [41]

Hersteller	Volvo
Typenbezeichnung	7900 Articulated Hybrid
Gesamtlänge	18,134 m
Gesamthöhe	3,28 m
Gesamtbreite	2,55 m
Zulässiges Gesamtgewicht	29 t
Maximale Geschwindigkeit	n. a.
Höchstleistung E-Motor	150 kW
Motorleistung Diesel	177 kW
Energiespeichertechnologie	Lithium-Ionen-Batterie
Batterie Nennspannung	etwa 600 V
Energiegehalt der Bus Batterie	2,4 kW h
Zeitbedarf Schnellladung	nicht möglich
Max. Schnellladeleistung	nicht möglich

2.2.2. EDDA-Bus

Durch das BEEDeL-Projekt mit der Hochbahn hat die BATSEN-Projektgruppe einen Kontakt mit dem Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme IVI aufgebaut. Das Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme hat in Zusammenarbeit mit Projektpartnern einen schnellladefähigen Plug-in-Elektrobus entwickelt (siehe Abb. C.9). Als Energiespeicher dienen dem EDDA-Bus NMC (Lithium Nickel Manganese Cobalt Oxide) Batterien. Das Batteriesystem wird auf dem Dach des Busses montiert und besteht aus drei Teilbatterien mit jeweils 621,6 V. Jede Teilbatterie besteht aus einer Reihenschaltung von sechs Submodulen zu je 103,6 V. Ein Submodul beinhaltet 28 in Reihe geschaltete Batteriezellen mit 46 A h und einer Zellspannung von 3,7 V. Die Gesamtkapazität des Batteriesystem beträgt 85,8 kW h [19]. In dem Zeitraum von November 2014 bis zum April 2015 wurde auf einem 14,4 km langen Teilabschnitts der Linie 61 der Dresdener Verkehrsbetriebe ein ausführlicher Praxistest durchgeführt. Der EDDA-Bus ähnelt dem Volvo 7900 Electric Hybrid 2.2.1.1, da sie beide schnellladefähig (Plug-In-fähig) sind, nahezu dieselbe Ladezeit benötigen und die gleiche Länge von 12 m besitzen. Dies macht die Auswertung der EDDA-Messdaten sehr interessant. In Tabelle 2.4 sind die Kennwerte des EDDA-Busses zusammengefasst.

Tabelle 2.4.: Kennwerte des EDDA-Busses [9]

Hersteller	Göppel Bus GmbH
Typenbezeichnung	Göppel Go4City (Umbau Batteriebus)
Gesamtlänge	11,981 m
Gesamthöhe	3,360 m
Gesamtbreite	2,250 m
Zulässiges Gesamtgewicht	18 t
Maximale Geschwindigkeit	85 km/h
Nennleistung E-Motor	145 kW
Motorleistung Diesel	n. a.
Energiespeichertechnologie	Lithium-Ionen-Batterie (NMC)
Batterie Nennspannung	621,6 V
Energiegehalt der Bus Batterie	85,8 kWh
Batteriekapazität einer Zelle	46 Ah
Zeitbedarf Schnellladung	4 $\frac{\text{kWh}}{\text{min}}$ bis zu 8,5 min
Max. Schnellladeleistung	240 kW Schnellladung (Begrenzung Ladestation) bzw. 470 kW Pulsladung (Begrenzung Batteriespeicher)

2.3. Fahrdynamik eines Elektrobusses

Um von der Fahrdynamik eines Fahrzeuges auf die elektrischen Fahreigenschaften des Fahrzeuges zu schließen, sind Berechnungsmodelle bzw. Ersatzschaltbilder notwendig. Ausgangspunkt für diese Arbeit war die Hypothese, dass die Beschleunigung und der Strom (bzw. die Leistung, bei der Annahme einer konstanten Batteriespannung) der Batterie zueinander proportional sind. Die ersten Messungen fanden unter dieser Annahme statt (siehe 3.2). Bei der weiteren Analyse ergab sich ein anderer Zusammenhang, der im Folgenden dargestellt wird. Hierzu wird zunächst die Beschleunigungsleistung entwickelt und darauf basierend die Leistungsgleichung hergeleitet.

2.3.1. Entwicklung der Beschleunigungsgleichung

Die Kräfte, die an dem Fahrzeug greifen und somit für das dynamische Verhalten des Fahrantriebes verantwortlich sind, werden in Abb. 2.9 veranschaulicht. Für die Fortbewegung in X-Richtung (Fahrtrichtung) greifen verschiedene Kräfte wie der aerodynamische Widerstand

F_l, die wirkenden Gewichtskräfte des Fahrzeuges F_{gF} und des Rades F_{gR}, die Reaktionskraft F_{Rx} zwischen Rad und Karosserie, die Traktionskraft F_x und der Rollwiderstand F_r. Das hier beschriebene Modell basiert auf der Quelle 'Antriebe für umweltfreundliche Fahrzeuge' [32].

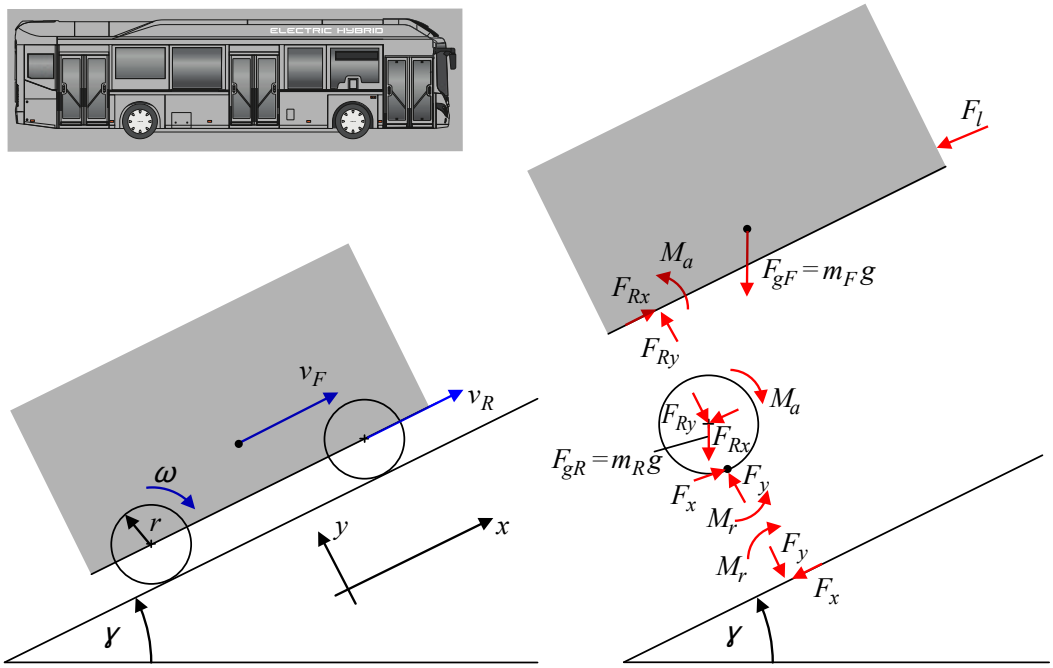


Abbildung 2.9.: Kräfte und Drehmomente am Fahrzeug [32][3].

- Aerodynamische Widerstand F_l
- Gewichtskräfte des Fahrzeuges F_{gF}
- Gewichtskräfte des Rades F_{gR}
- Reaktionskraft zwischen dem Rad und der Karosserie F_{Rx}
- Traktionskraft F_x
- Rollwiderstand F_r

Die dynamische Gleichung [32] für das Fahrzeug lautet:

$$m_F \cdot v_F' = m_F \cdot a_F = F_{Rx} - F_{gF} - F_l \tag{2.6}$$

und des Rades:

$$m_R \cdot v_R' = m_R \cdot a_R = F_x - F_{Rx} - F_{gR} \tag{2.7}$$

[3] Abbildung wurde an die Parameter angepasst.

Aus dem angreifenden Antriebsdrehmoment M_a, dem Widerstandsdrehmoment der Rollreibung M_R, dem Radius des Rades r, dem Trägheitsmoment des Rades J_R und der dazugehörigen Winkelbeschleunigung φ ergibt sich die dynamische Gleichung (2.8). Diese beinhaltet allerdings nur das Trägheitsmoment des Rades, nicht aber die restlichen rotierenden Komponenten eines Fahrzeuges wie z. B. das Getriebe und den Motor.

$$J_R \cdot \omega' = J_R \cdot \varphi = M_a - M_r - r \cdot F_x \tag{2.8}$$

Des Weiteren müssen die kinematischen Gleichungen berücksichtigt werden. Durch die Annahme, dass sich Fahrzeug und Rad gleich schnell bewegen

$$v = v_R = V_F \tag{2.9}$$

und sich die Masse der Räder und die des Fahrzeuges zur Gesamtmasse des Fahrzeugs zusammenfassen lässt,

$$m_{\text{ges}} = m_F + m_R \tag{2.10}$$

lassen sich die Parameter F_{gR} und F_{gF} zur Hangabtriebskraft F_g zusammenfassen

$$F_g = F_{gR} + F_{gF} = (m_F + m_R) \cdot g \cdot \sin(\gamma) = m_{\text{ges}} \cdot g \cdot \sin(\gamma) \tag{2.11}$$

Die Kombination der Hangantriebskraft und der beiden dynamischen Gleichungen (2.6) und (2.7) ergibt folgende Gleichung:

$$m_{\text{ges}} \cdot v' = F_x - F_g - F_l \tag{2.12}$$

Das Widerstandsdrehmoment (2.13) und das Antriebsdrehmoment (2.14) setzen sich wie in den angegebenen Gleichungen zusammen.

$$M_r = F_r \cdot r \tag{2.13}$$

$$M_a = F_a \cdot r \tag{2.14}$$

Durch Einsetzen in die Gleichung (2.8) ergibt sich die Formel:

$$J_R \cdot \varphi = (F_a - F_r - F_x) \cdot r \tag{2.15}$$

Wird nun angenommen, dass sich das Rad ideal verhält [29], gilt die Formel

$$a = \varphi \cdot r \tag{2.16}$$

Durch Umstellen der Gleichung (2.12) und (2.15) nach F_x und der unter (2.16) dargestellten Annahme, kann die Bewegungsgleichung für das Fahrzeug hergeleitet werden.

$$m_{\text{ges}} \cdot a + F_g + F_l = F_x = F_a - F_r - \frac{J_R \cdot \varphi}{r} \tag{2.17}$$

$$a \cdot \left(m_{\text{ges}} + \frac{J_r}{r^2} \right) = F_a - F_r - F_g - F_l \tag{2.18}$$

Durch Einführung des Beschleunigungswiderstandes

$$F_b = a \cdot \left(m_{\text{ges}} + \frac{J_r}{r^2} \right) \tag{2.19}$$

lässt sich die Bewegungsgleichung allgemein schreiben.

$$F_b = F_a - F_r - F_g - F_l \tag{2.20}$$

Die für die Bewegungsgleichung beteiligten Größen lassen sich wie folgt annähern:

- Der aerodynamische Widerstand (Luftwiderstand) F_l

$$F_l = \frac{\rho_l}{2} \cdot A_x \cdot c_w \cdot v^2 \tag{2.21}$$

  mit der spezifischen Dichte der Luft ρ_l, der wirksamem Querschnittsfläche des Fahrzeuges A_x, dem Luftwiderstandsbeiwert c_w und der Fahrzeuggeschwindigkeit v.

- Der Rollwiderstand F_r (2.13) lässt sich nach [18] wie folgt beschreiben.

$$F_r = f_R \cdot m_{\text{ges}} \cdot g \cdot \cos(\gamma) \tag{2.22}$$

  mit dem Rollwiderstandsbeiwert f_R, der Gesamtmasse des Fahrzeuges m_{ges}, der Erdbeschleunigung g und dem Steigungswinkel γ.

- Die Hangabtriebskraft F_g (2.12)

$$F_g = m_{\text{ges}} \cdot g \cdot \sin(\gamma) \tag{2.23}$$

  mit der Gesamtmasse des Fahrzeuges m_{ges}, der Erdbeschleunigung g und dem Steigungswinkel γ.

- Der Beschleunigungswiderstand F_b (2.19) lässt sich nach [36] und der Vernachlässigung des Trägheitsmoments des Rades J_R auch wie folgt angeben.

$$F_b = m_{\text{ges}} \cdot a \tag{2.24}$$

mit der Gesamtmasse des Fahrzeuges m_{ges} und der Fahrzeugbeschleunigung a.

In der Literatur [29] wird die Antriebskraft F_a auch mit der Zugkraft Z bezeichnet. Da es gängige Zugkraft-Geschwindigkeits-Diagramme in der Literatur gibt, wird an dieser Stelle die Zugkraft Z eingeführt.

$$Z = F_r + F_g + F_l + F_b \tag{2.25}$$

2.3.2. Die aus der Beschleunigungsgleichung resultierende Leistungsgleichung

Um von der Fahrdynamik auf die Leistung eines Fahrzeuges zu schließen, wird zunächst die Leistung an den Antriebsrädern P_R betrachtet. Dazu gilt nach [29] unter der Annahme, dass es sich um ein Fahrzeug mit einem Einachsantrieb handelt, folgende Gleichung:

$$P_R = \frac{M_R}{r} \cdot s \cdot v = Z \cdot s \cdot v \tag{2.26}$$

mit dem Schlupf s, dem Radius des Rades r, dem Drehmoment des Rades M_R und der Geschwindigkeit v. Wird nun der Schlupf des Fahrzeug wie nach [29] beschrieben vernachlässigt $s \sim 1$, ergibt sich folgende vereinfachte Gleichung

$$P_R = Z \cdot v \tag{2.27}$$

Durch Einsetzen der einzelnen Komponenten ergibt sich die Formel (2.28) für die Radleistung P_R, die sich aus der Geschwindigkeit multipliziert mit den Leistungsanteilen:

- Zeile 1: des Rollwiderstandes (2.22),
- Zeile 2: der Hangabtriebskraft (2.23),
- Zeile 3: des Beschleunigungswiderstandes (2.24) und
- Zeile 4: des aerodynamischen Widerstandes (2.21)

zusammensetzt.

$$\begin{aligned} P_R = {} & f_R \cdot m_{ges} \cdot g \cos(\gamma) \cdot v \\ & + m_{ges} \cdot g \cdot \sin(\gamma) \cdot v \\ & + m_{ges} \cdot a \cdot v \\ & + \tfrac{\rho_l}{2} \cdot A_x \cdot c_w \cdot v^3 \end{aligned} \tag{2.28}$$

Die Batterieleistung kann aus der Leistung des Rades und dem Wirkungsgrad des Antriebs, bestehend aus dem Umrichter, dem Motor bzw. Generator und dem Getriebe, berechnet

werden. In Abb. 2.10 ist der Antrieb mit den beteiligten Baugruppen als Blockschaltbild dargestellt.

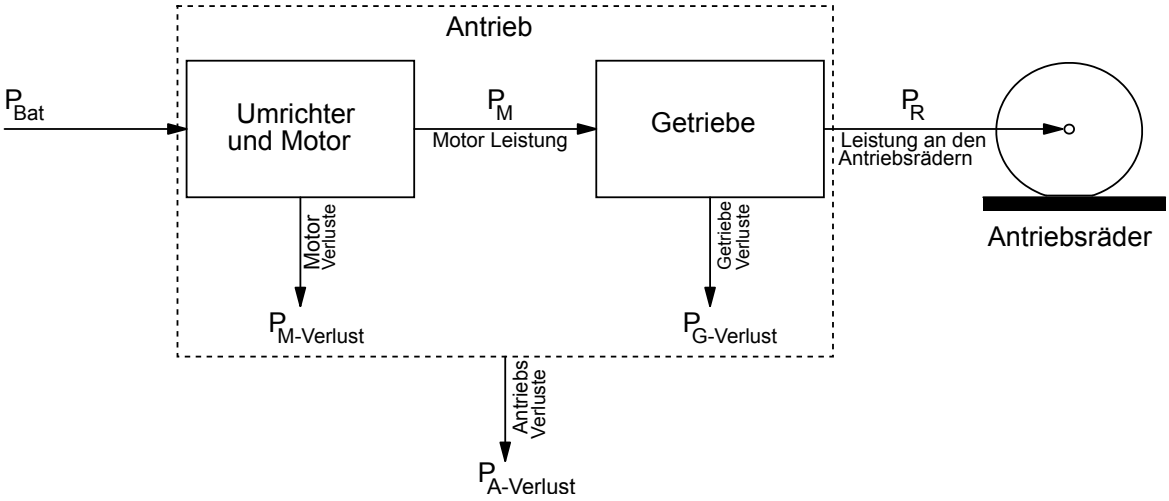


Abbildung 2.10.: Blockschaltbild des Antriebes und der im Antrieb entstehenden Verluste. Abbildung in Anlehnung an [29] erstellt

Die Verluste über den Antrieb ergeben sich wie in der Formel (2.29) vereinfacht dargestellt. Die Motor- und die Getriebeverluste werden zusammengefasst.

$$P_{\mathrm{A_{Verlust}}} = P_{\mathrm{M_{Verlust}}} + P_{\mathrm{G_{Verlust}}} = P_{\mathrm{Bat}} \cdot (1 - \eta_{\mathrm{A}}) \tag{2.29}$$

Durch Addition der Antriebsverlustleistung und der Radleistung ergibt sich die Formel für die Leistung der Batterie.

$$P_{\mathrm{Bat}} = P_{\mathrm{A_{Verlust}}} + P_{\mathrm{R}} = \frac{P_R}{\eta_{\mathrm{A}}} \tag{2.30}$$

In dem Buch 'Hybridfahrzeuge' [18] wird von einem Wirkungsgrad von Elektroantrieben von über $90\,\%$ geschrieben. Beispielsweise bei dem 'Scania Hybrid Concept Bus' ist ein Wirkungsgrad von bis zu $94\,\%$ angegeben. Darauf aufbauend, wird für die weiteren Berechnungen und Simulationen ein Antriebswirkungsgrad von $\eta_{\mathrm{A}} = 90\,\%$ angenommen.

Durch die Aufgeführte Herleitung der Batterieleistung aus dem Fahrverhalten wird die anfänglich angenommene Hypothese, dass der Strom der Batterie und die Beschleunigung des Busses proportional zueinander sind, widerlegt. Dies geht eindeutig aus der Formel (2.28) hervor, da die Leistung sowohl von v, $a \cdot v$ und v^3 abhängig ist.

In Abschnitt 3.2.2 wird das hier hergeleitete Modell mit reellen Parametern simuliert.

3. Konzeption des Elektrobuszyklus

Um den Einfluss des Fahrverhaltens der Elektrofahrzeuge auf die Batterie, wie z. B. Temperaturabhängigkeiten, Alterung, etc. zu untersuchen, ist es notwendig das Fahrverhalten eines Busses in Prüfständen zu simulieren. Dies ermöglicht die Untersuchung der Batterien außerhalb eines Busses in einer stabilen und einstellbaren Rahmenbedingung.

In den folgenden Abschnitten wird die Konzeption von Elektrobus-Zyklen aus ihrem Fahrverhalten beschrieben. Dazu werden unter Abschnitt 3.1 die Messmethoden zur Aufnahme des Fahrverhaltens dargestellt und unter Abschnitt 3.2 die Erstellung von Zyklen auf den in BATSEN-Projekt bestehenden Prüfstand, der unter Kapitel 4 vorgestellt wird, beschrieben.

3.1. Messmethoden zur Aufnahme des Fahrverhaltens

Um das Fahrverhalten bzw. Beschleunigungsverhalten von den Elektrohybridbussen der Linie 109 in Hamburg (Kapitel 2.2.1) zu erfassen, wird ein Datenlogger benötigt, der die Beschleunigung der x-,y- und z-Achse (wie in Abb. 3.1 dargestellt) und evtl. noch weitere Daten, wie z. B. die Geschwindigkeit und die Rotation, aufzeichnet. Solch ein Datenlogger wird parallel zu dieser Arbeit in dem BATSEN-Projekt von Felix Groth im Rahmen seiner Bachelorarbeit „Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen“ [15] entwickelt. Da sich dieser Datenlogger noch in der Entwicklung befindet, wurde nach alternativen Aufnahmemethoden gesucht.

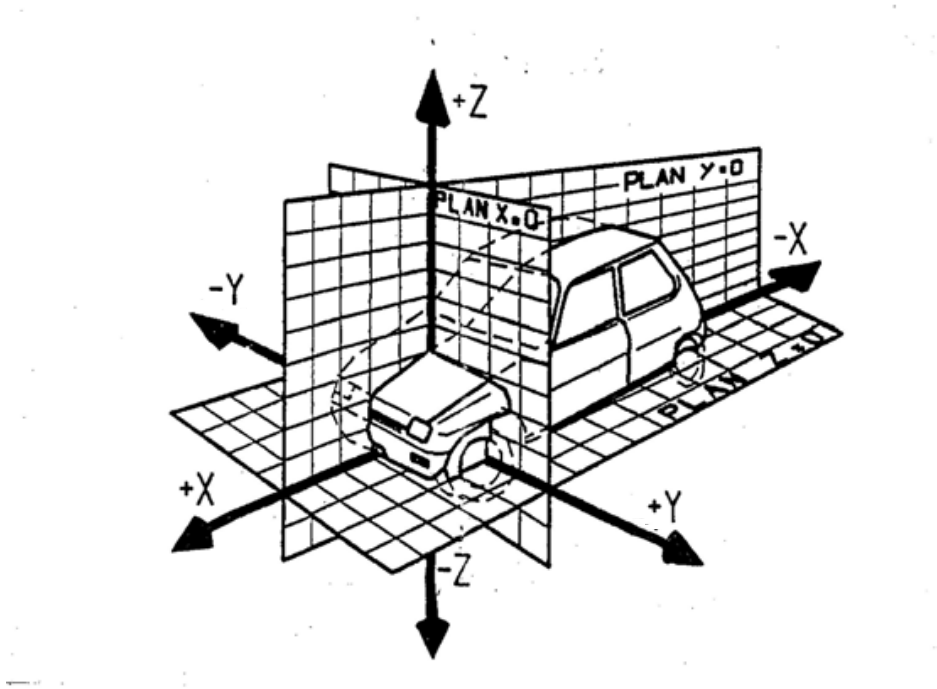


Abbildung 3.1.: Gängiges Fahrzeug-Koordinatensystem. Die Beschleunigung wird positiv und das Bremsen negativ dargestellt [30] [1].

3.1.1. Das Smartphone zur Aufnahme von Bus-Beschleunigungsdaten

Smartphones sind heutzutage kleine Alleskönner. Sie sind ausgestattet mit den verschiedensten Sensoren, unter anderem Gyroskopen und Beschleunigungssensoren. Außerdem besitzt jedes Smartphone interne und auch externe Speicher (SD-Karten), welche zur Messwertaufzeichnung verwendet werden können. Daher wurde in einem Versuch das Smartphone als alternativer Datenlogger getestet. In dem folgenden Abschnitt wird das Smartphone als Datenlogger kurz beschrieben und gezeigt, warum es sich nicht als zuverlässiges Messsystem eignet.

[1] Die Beschriftung der X- und Y-Achse wurde umgedreht. Die Beschleunigung in Fahrtrichtung wird positiv dargestellt.

3.1.1.1. Beschreibung des Messsystems

In der Tabelle 3.1 finden sich die Kennwerte des Smartphones, mit dem die Beschleunigungswerte aufgezeichnet wurden. Für dieses Smartphone wurden verschiedene Applikations (App's) getestet wie z. B.:

- Sensor Box for Android
- Physics toolbox V1.2.9
- Sensor Kinetics V2.1.1
- Sensor Kinetics Pro V2.1.2

'Sensor Kinetics Pro' überzeugte von der Bedienung am meisten. Es ermöglicht die erfassten Messdaten direkt als CSV-Datei zu speichern. Die Samplerate konnte von 'Fastest' über 'Spiel' bis hin zu 'Normal' eingestellt werden. Was dies im speziellen bedeutet war der App nicht zu entnehmen. Das gängige Android Koordinatensystem ist wie in Abb. 3.2 definiert. Die CSV-Datei wurde in MATLAB ausgewertet.

Tabelle 3.1.: Kennwerte des Smartphones

Hersteller	Sony
Typ	Sony Z1 Compact
Betriebsystem	Android Version 5.0.2
Modellnummer	D5503
Prozessor	Qualcomm Snapdragon 800 MSM8974

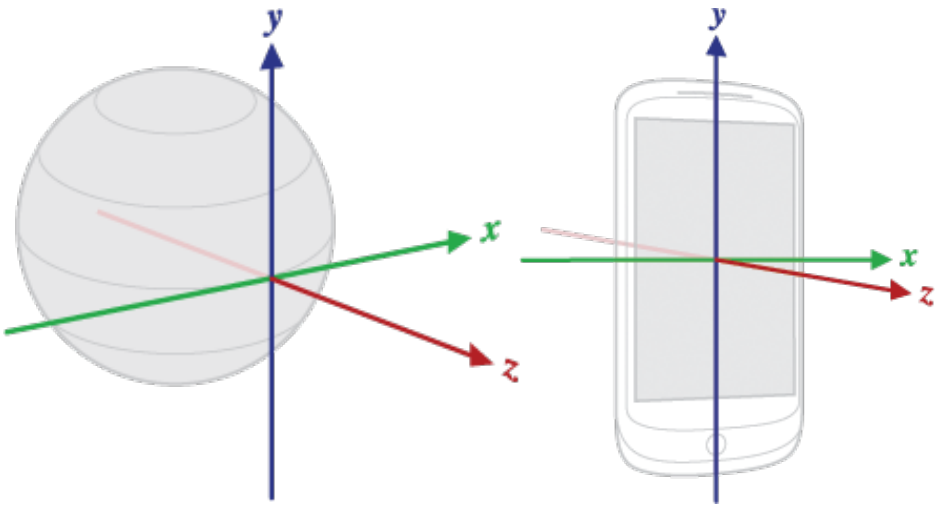


Abbildung 3.2.: Die gängige Definition des Android-Smartphone-Koordinatensystems (rechts) mit dem Weltkoordinatensystem (links) nach der 'SensorEvent API' [4]

3.1.1.2. Probebetrieb des Smartphones als Datenlogger

Um das Smartphone als Beschleunigungssensor zu testen, wurde eine Testfahrt in dem Volvo articulated Hybrid 2.2.1.2 unternommen. Dazu wurde das Smartphone in den Flugzeugmodus versetzt, um möglichst viele Störfaktoren, wie z. B. Telefonate zu eliminieren. Des Weiteren wurde darauf geachtet, dass möglichst wenige Hintergrundfunktionen während der Messung laufen. Für die Messung wurde das Smartphone an einer geraden horizontalen Fläche mit doppelseitigem Klebeband fixiert. Die X-Richtung des Smartphones zeigte dabei in Fahrtrichtung. Die bei der Messfahrt entstandene CSV-Datei wurden in MATLAB importiert und dargestellt. In Abb. 3.3 ist ein Auszug der Messdaten enthalten. Es sind die Standzeiten des Busses an den Haltestellen und an einer Ampel sowie die Beschleunigungs- und Bremsphasen deutlich zu erkennen.

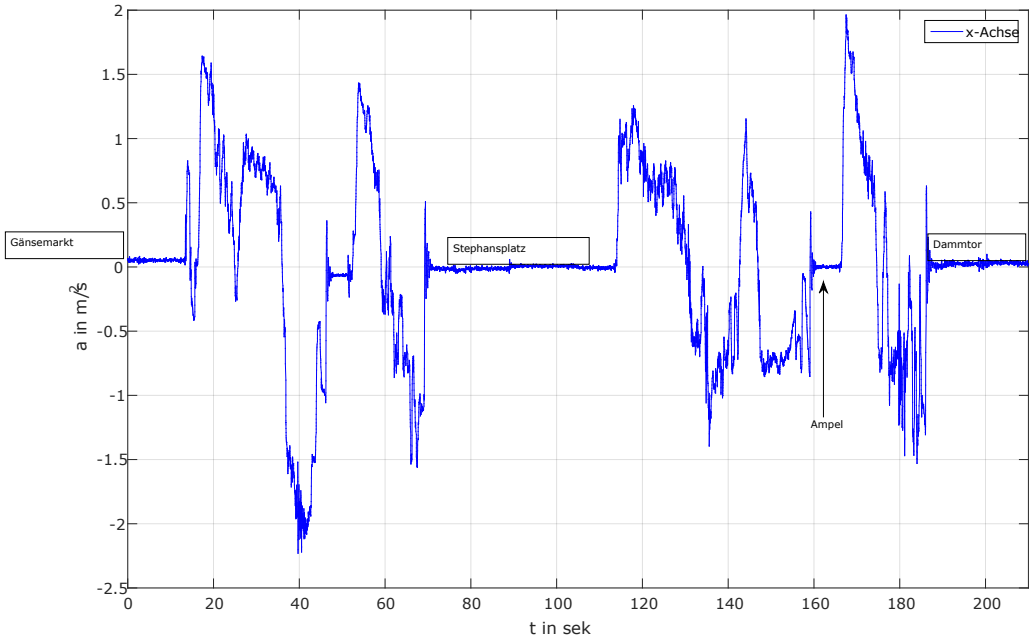


Abbildung 3.3.: Exemplarischer Plot, der von Smartphone aufgezeichneten Beschleunigungsmessdaten in Fahrtrichtung des Gelenkbuses Linie 109 zwischen den Haltestellen Gänsemarkt und Dammtor

Bei der näheren Untersuchung der Daten stellte sich heraus, dass die Messwerte nicht äquidistant abgetastet wurden. Durch eine Lagrange Interpolation [7] wäre eine Aufbereitung möglich gewesen, wenn nicht, wie unter Abb. 3.4 dargestellt, mehrere Messwerte zu einem Zeitpunkt erfasst worden wären. Scheinbar weist die App einem Zeitpunkt mehrere Messwerte zu. Zu vermuten ist, dass das Android-Betriebssystem den App's Zeitscheiben zuteilt, in denen die App's laufen dürfen. Ist eine dieser Zeitscheiben zu Ende, werden kurzzeitig keine Messwerte in der Datenlogger-App mehr aufgenommen (grüner Bereich in der Abb. 3.4). Sie verbleiben in einem internen Buffer. Teilt das Betriebssystem der App die nächste Zeitscheibe zu, werden alle Messungen, die in dem grünen Bereich stattgefunden haben, einem Zeitpunkt zugewiesen (dem roten Bereich).

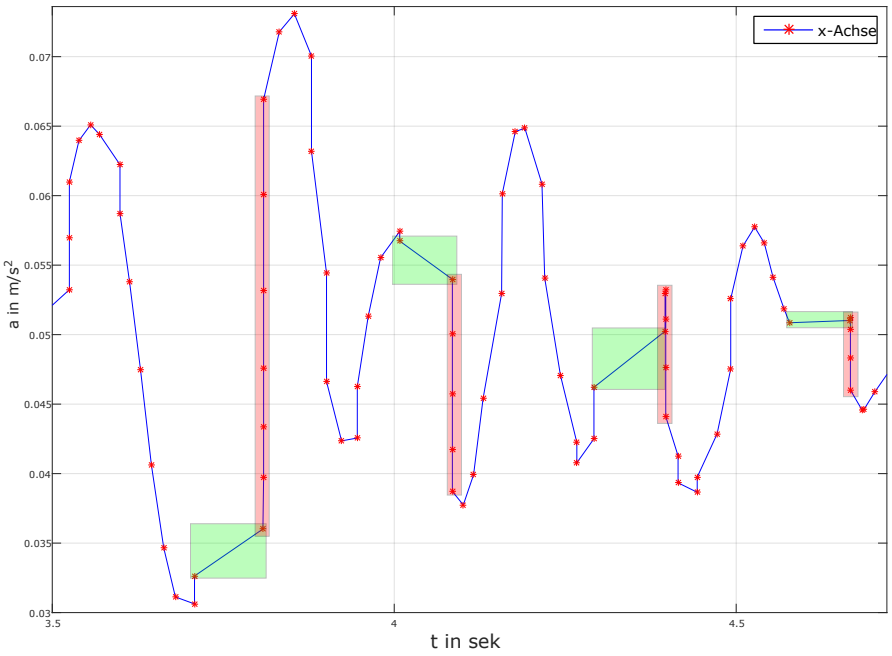


Abbildung 3.4.: Vergrößerter Ausschnitt der vom Smartphone aufgezeichneten Messdaten. Deutlich zu sehen ist, dass über einen längeren Zeitraum keine Messdaten (grün) und danach zu einem Zeitpunkt mehrere Messpunkte aufgezeichnet wurden (rot).

Die Abb. 3.5 zeigt die Häufung der Zeitwerte, welche die gleiche Differenz

$$\Delta t_n = t_{n+1} - t_n \tag{3.1}$$

zu ihrem Nachbarzeitwert haben. Beachtlich ist hierbei, dass von 13901 Zeitwerten allein 4000 Messwerte einen Abstand von $\Delta t_n = 0$ besitzen. Dies bedeutet, dass $28{,}77\,\%$ der Messwerte nicht zuzuordnen und somit nicht nutzbar sind.
Eine ausführliche Recherche über die internen Sensoren eines Smartphones führte zu der Studienarbeit mit dem Titel „Android Smartphone als Fahrzeug Datenlogger“ [30]. Die Programmierung einer solchen Datenlogger App stand im Fokus dieser Arbeit. Außerdem wird darauf hingewiesen, dass ein Smartphone kein Echtzeitsystem ist und das bei einer vollen Auslastung der Prozessoren keine Messdaten mehr aufgenommen werden können. Dies ist mit der oben beschriebenen Hypothese konform

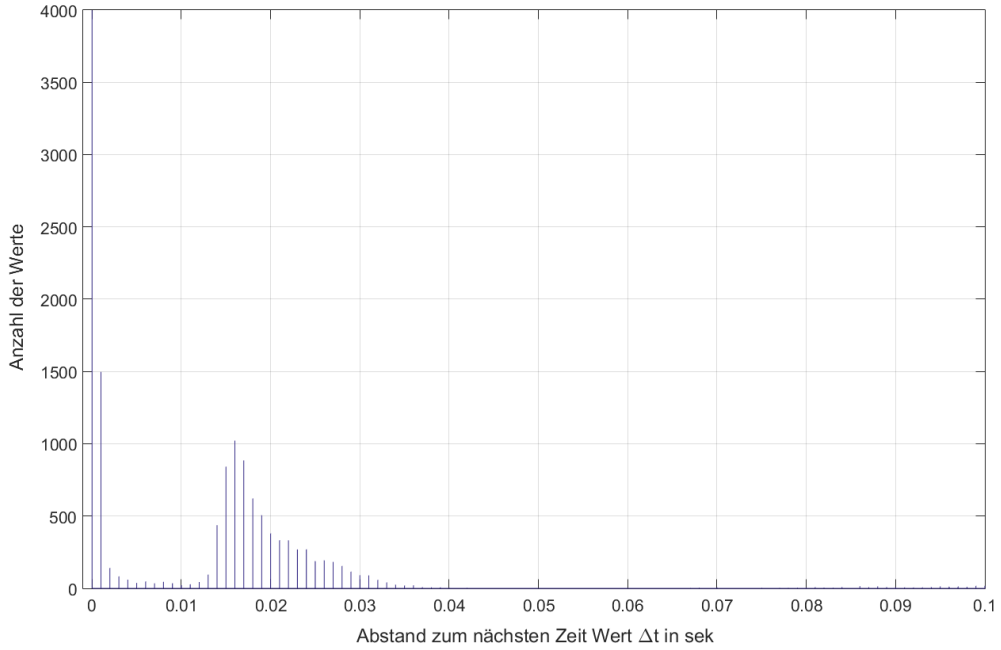


Abbildung 3.5.: Histogramm zeigt die Differenz $\Delta t_n = t_{n+1} - t_n$ zum nächsten Zeitwert bei der Smartphone-Messung.

3.1.1.3. Fazit

Da die Messwerte des Smartphones nicht äquidistant und somit fehlerhaft sind, fiel eine Entscheidung gegen das Messsystem 'Smartphone als Datenlogger'. Die Aufarbeitung der Messdaten hätte zu einem zu hohen zeitlichen Aufwand geführt.

3.1.2. Das Messsystem mit 3-Achsen Beschleunigungssensor zur Aufnahme der Bus-Beschleunigungsdaten

Nachdem sich das Smartphone als Datenlogger als nicht zuverlässlich herausgestellt hat, wurde nach einem alternativen Messsystem gesucht. Die Wahl fiel auf ein Messsystem, das von der Firma AIS zur Verfügung gestellt wurde. Das Messsystem 'PULSE' [11] ist eine Plattform für Schall- und Schwingungsmessungen von der Herstellerfirma Brüel & Kjær. Als Schwingungsaufnehmer wurde ein 3-Achsen Beschleunigungsaufnehmer ADXL335 [3] der Firma ANALOG-DEVICES mit einer externen Beschaltung, wie unter Abb. C.17 gezeigt, verwendet.

Der Vorteil dieses Messsystems ist, dass es durch eine 12 V Batterie versorgt werden kann und in einem Rollkoffer siehe Abb. C.16 mobil einzusetzen ist. Alle drei Beschleunigungssignale (x,y und z) lassen sich parallel mit einer Abtastfrequenz von $4096\,\mathrm{Hz}$ aufzeichnen und in ein MATLAB-Format (m-File) konvertieren. Vor der Messfahrt wurde der Sensor kalibriert. Hierzu wurde die Erdbeschleunigung $g = 9{,}81\,\mathrm{m/s^2}$ genutzt.

3.1.2.1. Der 3-Achsen Beschleunigungsaufnehmer

Der Beschleunigungsaufnehmer ADXL335 [3] der Firma ANALOG-DEVICES wird auf einer Platine geliefert. Zum praktischen Einsatz musste eine Stomversorgung und ein mechanisches Gehäuse mit Kontaktsteckern aufgebaut werden. Außerdem ist eine Funktionsüberprüfung notwendig. So musste die Empfindlichkeit $\frac{\mathrm{mV}}{\mathrm{m/s^2}}$, Eigenfrequenzen und der Frequenzgang ermittelt werden, da in den Herstellerangaben nur typische Kenndaten enthalten sind.

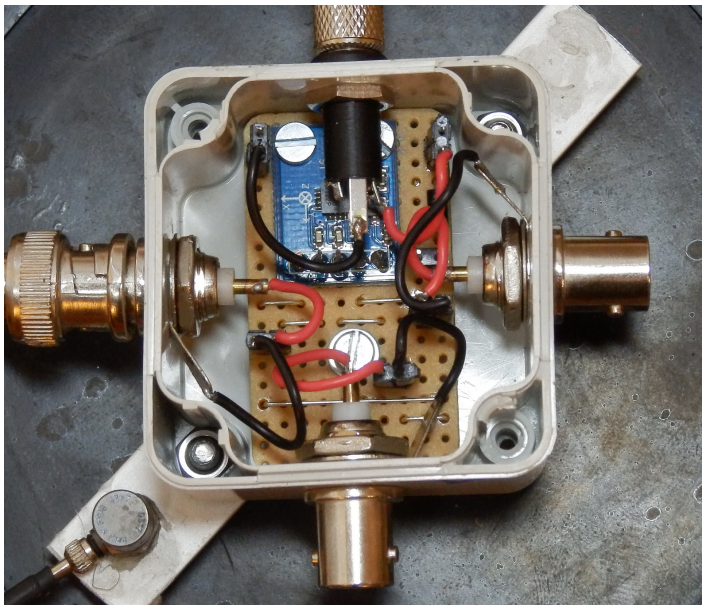


Abbildung 3.6.: Der Beschleunigungssensor mit drei BNC-Buchsen und Stromversorgungs-Buchse

Der Sensor wurde auf eine Platine gelötet und starr in einem Gehäuse montiert (siehe Abb. 3.6). Eine starre Montage ist notwendig, damit Resonanzfrequenzen des Sensoraufbaus die Messungen nicht verfälschen. Eine Überprüfung des Resonanzverhaltens wurde durch eine Bestimmung der dynamischen Übertragungsfunktion des Sensors durchgeführt. In Abb. B.2 ist der Messaufbau zur Bestimmung des Frequenzganges dargestellt.

Auf dem Schwingerreger (Shaker) wurden der Sensor und ein Referenz-Piezoelektrischer-Beschleunigungsaufnehmer montiert. Als Anregungssignal wurde ein Sinus-Sweep von 1 bis 400 Hz verwendet.

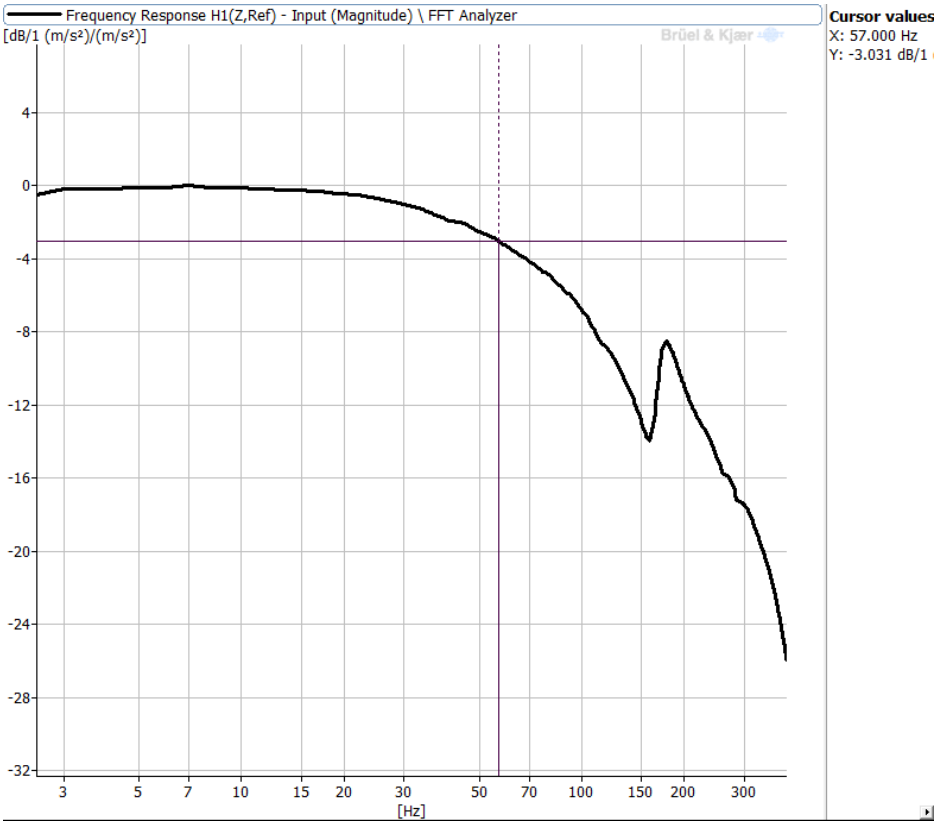


Abbildung 3.7.: Frequenzgang des Beschleunigungsaufnehmers in Z-Richtung

In Abb. 3.7 ist der ermittelte Frequenzgang für die Z-Richtung dargestellt. Auf der Y-Achse der Übertragungsfunktion ist das Amplitudenverhältnis des Beschleunigungsaufnehmers zum Referenzaufnehmer in dB aufgetragen. Der Cursor steht auf der oberen 3 dB Grenzfrequenz bei $F_{-3\,\mathrm{dB\ mess}} = 57$ Hz. Der nutzbare Frequenzbereich des Sensors liegt zwischen 0 Hz und 57 Hz. Durch die externen 0,1 µF Kondensatoren (wie in Abb. C.17 abgebildet), die meist eine Toleranz von $\pm 10\,\%$ besitzen, ergibt sich aus der Formel (3.2) des Datenblattes [3] ein Bereich (Worst Case) für die Grenzfrequenz des Beschleunigungssensors.

$$\begin{aligned} F_{-3\,\mathrm{dB}} &= \frac{1}{2 \cdot \pi \cdot (32\,\mathrm{k\Omega} \pm 15\,\%) \cdot C} \\ &= \frac{1}{2 \cdot \pi \cdot (32\,\mathrm{k\Omega} \pm 15\,\%) \cdot (0{,}1\,\mu\mathrm{F} \pm 15\,\%)} \\ &= 39{,}32\,\mathrm{Hz} \cdots 65{,}01\,\mathrm{Hz} \end{aligned} \tag{3.2}$$

Somit liegt die aus dem Frequenzgang bestimmte Grenzfrequenz $F_{-3\,\mathrm{dB\ mess}} = 57\,\mathrm{Hz}$ sehr genau in dem aus dem Datenblatt ermittelten Bereich. Die Überhöhung bei $180\,\mathrm{Hz}$ deutet auf eine Eigenfrequenz des Sensoraufbaus hin.

3.1.2.2. Probebetrieb des Beschleunigungs-Sensors und des Brüel&Kjær Mess-Systems

Um das Messsystem als Datenlogger zu testen, wurde eine Testfahrt in dem Volvo articulated Hybrid 2.2.1.2 unternommen. Bei der Testfahrt handelt es sich um dieselbe Busfahrt wie bei der Smartphone-Testfahrt 3.1.1. Beide Systeme wurden parallel getestet. Für die Messung wurde der Koffer an einer geraden Fläche im Bus platziert und befestigt. Die bei der Messfahrt entstandenen m-Files wurden in MATLAB importiert. In Abb. 3.8 ist ein Auszug der Messdaten dargestellt. Deutlich sind die Standzeiten des Busses an den Haltestellen und an einer Ampel zu erkennen. Die Beschleunigungs-(positive Beschleunigung z. B. 108 bis 118 s) und Bremsphasen (negative Beschleunigung z. B. 127 bis 137 s) sind ebenfalls sichtbar.

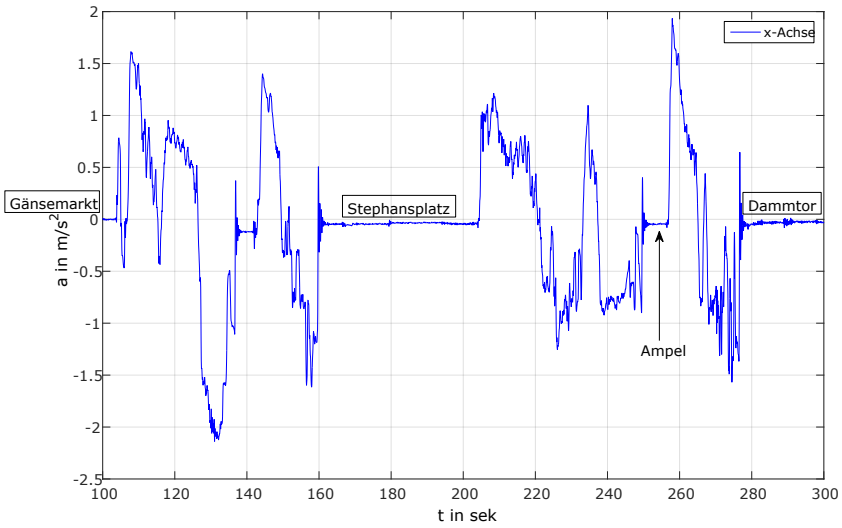


Abbildung 3.8.: Exemplarischer Darstellung der von dem Messsystem aufgezeichneten Beschleunigungsmessdaten in Fahrtrichtung des Gelenkbusses der Linie 109 zwischen den Haltestellen Gänsemarkt und Dammtor.

In Abb. 3.9 ist das zeitlich gemittelte Frequenzspektrum der Beschleunigung dargestellt. Es ist deutlich zu erkennen, dass der Hauptanteil der Beschleunigung im Frequenzbereich zwischen 0 Hz und 5 Hz liegt (gelb markierter Bereich). Für die gesamte Messung (ca. 27 min)

beträgt der zeitlich gemittelte Gesamteffektivwert der Beschleunigung $a_{\text{Total}} = 0{,}63\ \frac{\text{m}}{\text{s}^2}$. Demgegenüber steht ein Beschleunigungseffektivwert für den Frequenzbereich von 0 bis 5 Hz von $a_{\,0\,\text{Hz}-5\text{Hz}} = 0{,}62\ \frac{\text{m}}{\text{s}^2}$ (siehe Cursor-Wert 'Delta' in Abb. 3.9). Es ergibt sich somit ein Verhältnis von:

$$\frac{a_{\,0\,\text{Hz}-5\text{Hz}}}{a_{\,\text{Total}}} = \frac{0{,}62\ \frac{\text{m}}{\text{s}^2}}{0{,}63\ \frac{\text{m}}{\text{s}^2}} = 98{,}4\,\% \tag{3.3}$$

Im Frequenzbereich von 0 bis 5 Hz liegt somit ein Anteil von 98,4 % der Gesamtbeschleunigung. Bei der Ermittlung des Beschleunigungseffektivwerts für den Frequenzbereich von 0 bis 1 Hz ergab sich ein Verhältnis von 95,2 %. Aus diesem Grund können die Messdaten für die weiteren Untersuchungen, Analysen und zur Datenreduktion mit einem 1 Hz Tiefpassfilter gefiltert werden.

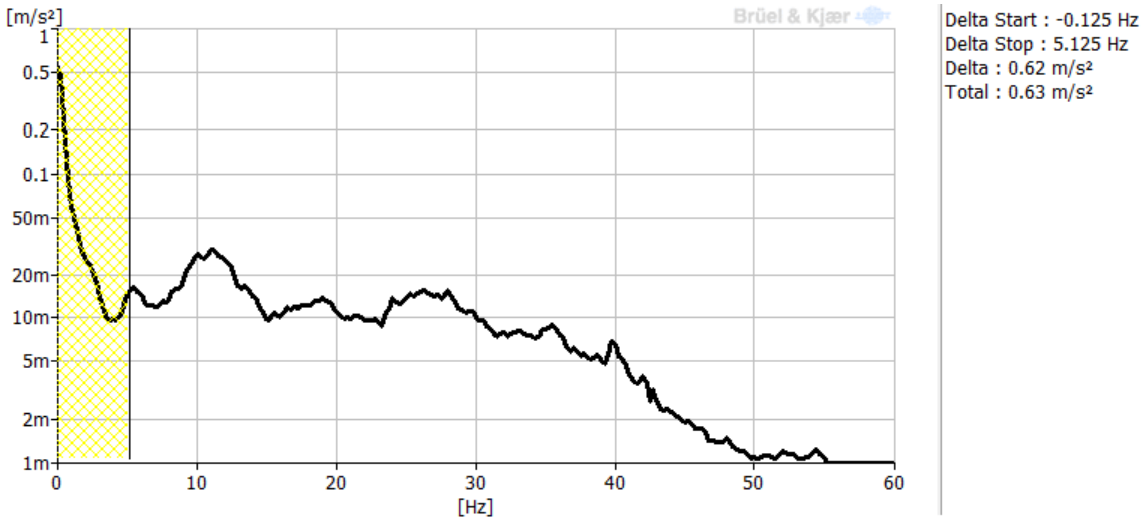


Abbildung 3.9.: Zeitlich gemitteltes Effektivwertspektrum der Messfahrt für den Frequenzbereich von 0 bis 60 Hz. In Abb. B.3 ist das Spektrum von 0 bis 400 Hz dargestellt.

3.1.2.3. Fazit

Dieses Messsystem eignet sich sehr gut für die Aufnahme der Bus-Beschleunigungsdaten. Allerdings stellte sich bei der weiteren Auswertung heraus, dass die alleinige Aufnahme der Beschleunigung nicht ausreichend ist um einen Bus-Fahrzyklus genau zu beschreiben (siehe 3.2). Da das Mess-System nur eine Leihgabe war, konnte zeitlich bedingt nur der Volvo Articulated Hybrid vermessen werden.

3.1.3. EDDA-Messdaten des Fraunhofer Instituts

Die durch das Projekt BEEDeL entstandenen Kontakte ermöglichten den Zugang zu Messdaten des EDDA-Busses 2.2.2. Vom Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme wurden Messdaten von verschiedenen Messungen zur Verfügung gestellt. Die Daten befinden sich in einem MATLAB m-File und sind in einer MATLAB-Datenstruktur hinterlegt. Die Daten beinhalten fünf Fahrten mit dem EDDA Bus. Während einer Fahrt wurden mehrere Messgrößen erfasst, die in der Tabelle 3.2 dargestellt sind.

Tabelle 3.2.: Darstellung der Messgrößen des EDDA-Busses, die in der Strukture 'Daten' erfasst wurden

Bezeichnung	Datentyp	Abtastrate
Dateiname	String (char)	n. a.
Datum	Cell-Array (cell)	n. a.
GPS_Geschw	Vektor (double)	1 Sample/s
GPS_Hoehe	Vektor (double)	1 Sample/s
Latitude	Vektor (double)	1 Sample/s
Longitude	Vektor (double)	1 Sample/s
Spg_Panto_V	Vektor (double)	10 Samples/s
Spg_Zwischenkreis_V	Vektor (double)	10 Samples/s
Strom_Batt_1_Bremswiderstand_A	Vektor (double)	10 Samples/s
Strom_Bordnetzumrichter_A	Vektor (double)	10 Samples/s
Strom_DPU4_Batt_2_3_A	Vektor (double)	10 Samples/s
Strom_DPU5_Fahrmotor_A	Vektor (double)	10 Samples/s
Strom_Panto_1_A	Vektor (double)	10 Samples/s
Strom_Panto_2_A	Vektor (double)	10 Samples/s
UTC_Zeit	Cell-Array (cell)	1 Sample/s
GPS_valid	Cell-Array (cell)	n. a.
Weg	Vektor (double)	1 Sample/s
Zeit	Vektor (double)	1 Sample/s nicht äquidistant

3.1.3.1. Aufbereiten der Messdaten

Die Messdaten wurden in MATLAB importiert und analysiert. Laut dem Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme wurden die Messwerte mit $10\,\frac{\text{Samples}}{\text{s}}$ aufgenommen. Bei der Analyse der Messdaten ergab sich, dass die aufgenommene Messgrößen unterschiedlich abgetastet wurden, wie in Tabelle 3.2 aufgeführt. Die interessanten GPS-Daten wurden nur sekündlich aufgezeichnet. Da der Zeitvektor, wie in dem Histogramm Abb. C.18

abgebildet nicht äquidistant ist, wurde bei der weiteren Auswertung alle Messdaten auf einen künstlichen $1\ \frac{\text{Samples}}{\text{s}}$ Zeitvektor bezogen. Die Vektoren, die mit $10\ \frac{\text{Samples}}{\text{s}}$ abgetastet wurden, wurden zur Vermeidung von Alias-Effekten mit $0{,}5\,\text{Hz}$ tiefpassgefiltert und danach $1\ \frac{\text{Samples}}{\text{s}}$ neu abgetastet (Resampling). Aus den Daten wurde ein Prinzip-Schaltbild des EDDA-Busses entworfen, welches in Abb. B.4 dargestellt ist. Das Prinzip-Schaltbild wurde in Absprache mit dem Projektleiter[9] des EDDA-Busses erstellt und enthält die Messstellen des Busses.

In der Abb. 3.10 ist ein Auszug einer Fahrt des EDDA-Busses dargestellt. Die gesamte Fahrt ist in Abb. C.19 abgebildet. Die Beschleunigung wurde aus der Geschwindigkeit abgeleitet. Zu erwähnen ist, dass diese Daten aus einer GPS Messung resultieren, also einer Positionsbestimmung. Die Geschwindigkeit ist somit bereits die erste Ableitung des Weges nach der Zeit. Durch die zweite Ableitung des Weges nach der Zeit ergibt sich die Beschleunigung. Zu beachten ist, dass durch eine Fehlerfortpflanzung die Messwerte beeinflusst werden. Diese Fehler waren in der Arbeit von Felix Groth [15] bereits zu sehen. Der Fehler wurde bei dem Vergleich der Beschleunigungssensor-Messdaten mit den ermittelten GPS Beschleunigungsdaten deutlich. In der Abb. 3.11 ist der Strom in den Fahrmotor und die Beschleunigung des Busses als Zeitsignalauszug über $5\,\text{min}$ dargestellt. Dieser Auszug stammt aus dem Zeitsignal in Abb. C.20.

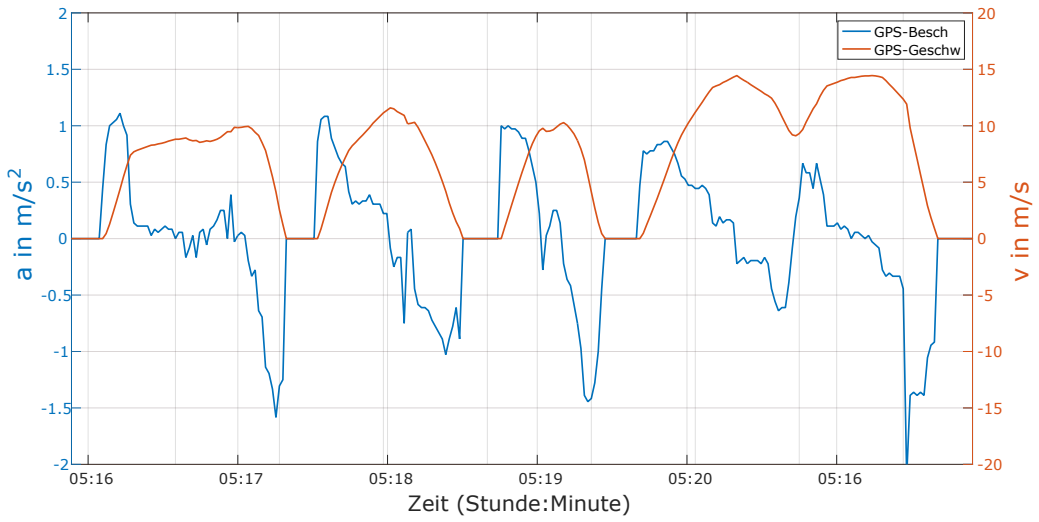


Abbildung 3.10.: Zeitlicher Ausschnitt einer Fahrt des EDDA-Busses. Dargestellt sind die Geschwindigkeit und die aus der Geschwindigkeit abgeleitete Beschleunigung des Busses.

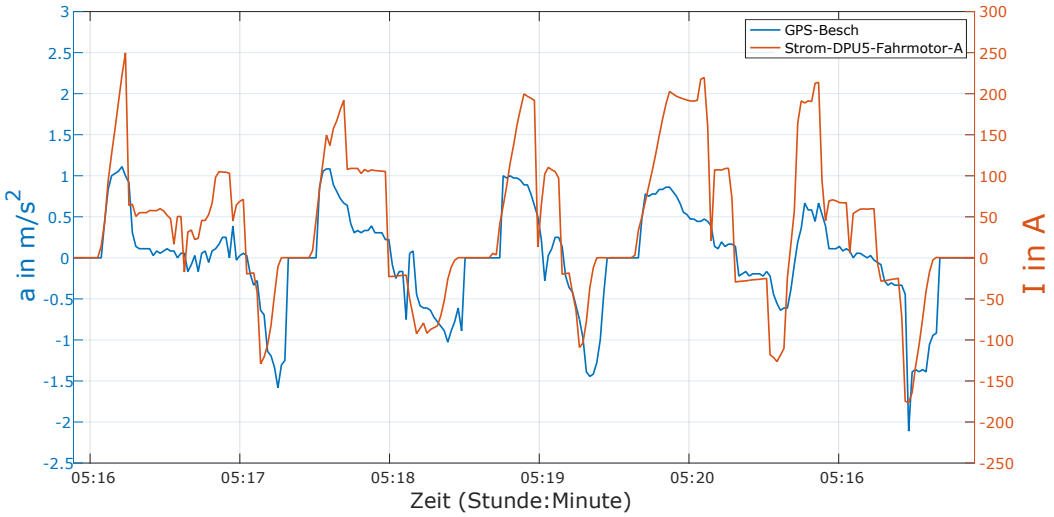


Abbildung 3.11.: Zeitlicher Ausschnitt einer Fahrt des EDDA-Busses. Dargestellt sind die aus der Geschwindigkeit abgeleitete Beschleunigung und der Stom in den Fahrmotor.

3.1.3.2. Fazit

Die Messwerte des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme sind sehr informativ, da sie den aus dem Fahrverhalten resultierenden Strom des Busses zeigen. Eine erste Betrachtung der Abb. 3.11 zeigt, dass ein Zusammenhang zwischen der Beschleunigung eines Busses und den in den Fahrmotor fließenden Strom gibt. In Kapitel 3.2.2.1 wird dieses Verhalten näher analysiert.

3.2. Entwurf und Erstellung von Elektrobuszyklen für den Prüfstand

Aus den Aufzeichnungen des Fahrprofils aus Abschnitt 3.1.2 und aus den Messdaten des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme (siehe Abschnitt 3.1.3) wurden in den folgenden Abschnitten zwei Zyklen entwickelt. Diese unterscheiden sich folgendermaßen:

- Der erste Zyklus ist dahingehend ausgelegt, dass er auf den gemessenen Beschleunigungsdaten des Volvo Articulated Hybrid basiert und den Prüfstand durch eine hohe dynamische Belastung beansprucht.

- Der zweite Zyklus basiert auf den wirklichen Strömen der unter Abschnitt 3.1.3 dargestellten Messdaten des EDDA-Busses. Des Weiten wurde der Zusammenhang zwischen dem Fahrverhalten und seinem Einfluss auf die Leistung an der Batterie untersucht.

3.2.1. Entwurf und Erstellung eines Zyklus auf Basis der statistischen Verteilung der gemessenen Beschleunigungen (Zyklus-I)

In dem Zyklus der Innovationslinie konnte nicht direkt auf die Leistung der Batterie geschlossen werden, da für die Formel (2.30) die Geschwindigkeit des Busses bekannt sein müsste. Die Ermittlung der Geschwindigkeit aus den Beschleunigungsdaten war nicht möglich. Dies wird in Unterkapitel 3.2.1.4 beschrieben. Bedingt durch die Hardware des Zykliergerätes, den technischen Grenzen des aktuellen und optimierten Prüfstandes sowie dem aufgenommenen Fahrverhalten des Busses, ergeben sich folgende Vorgaben für die Erstellung und Auslegung eines Bus-Zyklus:

- Es gibt maximal fünf Stromstufen für Lade- und Entladevorgänge
  - 2 Ladestufen/Ladepfade
  - 2 Entladestufen/Lastpfade
  - 1 Ruhestufe/keine Belastung.
- Die Stromstufen können maximal 1-mal pro Sekunde umgeschaltet werden.
- Der maximale Ladestrom beträgt $20\,\mathrm{A}$.
- Der maximale Entladestrom beträgt $25\,\mathrm{A}$.
- Der Einfluss des Fahrverhaltens auf den Strom der Batterie ist aufgrund der unbekannten Geschwindigkeit des Busses, wie in 3.2.1.4. dargestellt, nicht bekannt.

Das übernommene Zykliergerät, der Prüfstand und deren Funktionen sowie die notwendigen Optimierungen werden in Kapitel 4 beschrieben. Aus der gemessenen Beschleunigung und der statistischen Verteilung wurde ein Zyklus entwickelt. Für die Erstellung eines Zyklus wurden in Abstimmung mit dem Projekt BATSEN folgende Randbedingungen und Vorgehensweisen definiert:

- Der Zyklus basiert auf der statistischen Verteilung der gemessenen Beschleunigungen, bei einer gleichmäßigen Verteilung (Häufigkeiten) der einzelnen Stromstufen.
- Die maximalen Ströme des Prüfstandes sollen ausgenutzt werden.
- Die Batterien sind bei einer möglichst hohen Strombelastung ($>1\,\mathrm{C}$) zu betreiben.

- Die Berücksichtigung einer ca. 50 % Rekuperation wie in 2.1.3 beschrieben.

Weitere Vorgaben ergaben sich aus den vorhandenen drei Batterietypen, die sich im Wesentlichen durch Ihre Kapazitäten und den maximalen Strömen unterscheiden. Die Kenndaten der Batterien sind in Tabelle 2.1. aufgeführt. Aus der Kombination dieser Vorgaben ergibt sich somit ein Zyklus mit fünf Laststufen und einer zeitlichen Auflösung von 1 s. Mit den vorhandenen Batterien können mit diesem Prüfstand folgende in Tabelle 3.3 dargestellten maximalen Strombelastungen realisiert werden.

Tabelle 3.3.: Die maximalen Strombelastungen der Batterien auf dem Prüfstand

Batterietyp	Laden	Entladen
ECC	0,48 C	0,6 C
LiNANO	1,33 C	1,66 C
A123	8 C	10 C

Die ECC-Batterie ist somit für den Prüfstand und der gestellten Anforderung (>1 C) nicht geeignet. Die ECC-Batterie kann aufgrund ihrer Größe mit diesem Prüfstand unter der Randbedingung einer möglichst hohen Strombelastung nur in einem geringen Bereich ihrer Leistungsfähigkeit belastet werden. Daher wird der entwickelte Zyklus nicht an dieser Batterie getestet. Da die A123-Batterie laut Datenblatt für maximal 4 C Ladestrombelastung ausgelegt ist, wurde eine zusätzliche Untersuchung zur Erhöhung der Strombelastung mit 8 C durchgeführt (siehe Kapitel 6.2.2.2). Bei dieser Untersuchung stellten sich keine Auffälligkeiten dar, die gegen das Laden der Batterie mit 8 C sprachen.

3.2.1.1. Analyse und statistische Verteilung der gemessenen Beschleunigungen

Der gemessene zeitliche Beschleunigungs-Verlauf (siehe Ausschnitt in Abb. 3.8) wurde unter Berücksichtigung der Häufigkeitsverteilung statistisch analysiert(siehe Abb. 3.12).
Da die Messdaten mit einer Abtastung von 4096 Samples/s vorliegen, wurde eine Tiefpassfilterung (gleitende Mittelwertbildung) und Neuabtastung (Resampling) mit 1 Sample/s durchgeführt. Somit entspricht die Y-Achse der Häufigkeitsverteilung der Anzahl der 1 Sample/s in der jeweils gleichen Beschleunigungsklasse. Insgesamt wurden 201 Klassen (X-Achse) mit einer Breite von etwa 0,023 m/s^2 gewählt.

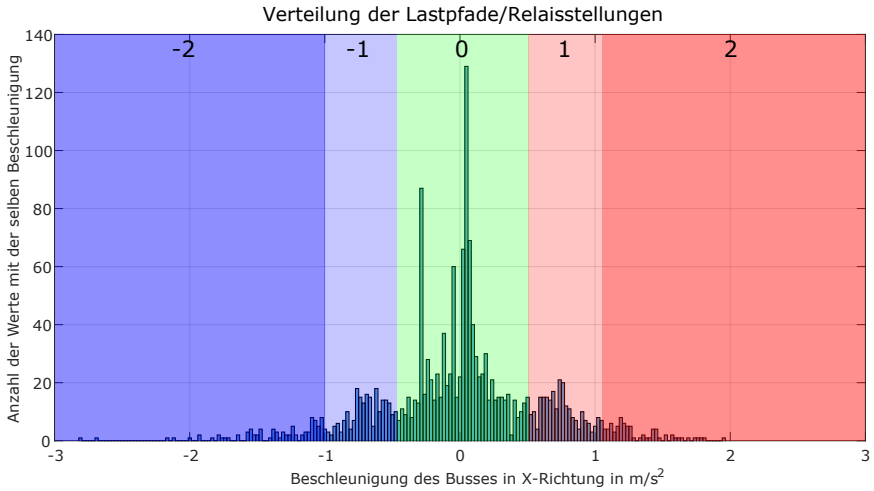


Abbildung 3.12.: Häufigkeitsverteilung der Beschleunigung.

Deutlich zu erkennen ist der große Anteil der Werte in Klasse $0\,\mathrm{m/s^2}$. Diese zeigt die Standzeiten bzw. Phasen, in denen der Bus keine Beschleunigung erfährt. Die Klassen von $-0{,}45$ bis $0{,}5\,\mathrm{m/s^2}$ sollten zu der Standzeit bzw. den Phasen, in denen der Bus keine Beschleunigung erfährt, hinzu gezählt werden, da die Erdanziehungskraft einen starken Einfluss auf die Messwerte hat. Dies geschieht z. B. bei einem Halt an einer Steigung oder dem einseitigen Absenken des Busses für den bequemen Zustieg der Fahrgäste an einer Haltestelle. Zu sehen ist dieser Offset beispielhaft in Abb. 3.8 zur Standzeit an der Ampel. Ein Offsetabgleich wird am Anfang der Fahrt bei Stillstand an einer möglichst ebenen Stelle durchgeführt. Allerdings wird dieser einmalig erfolgte Offsetabgleich für die gesamte Fahrt angewendet.

3.2.1.2. Anpassung der statistischen Daten an die fünf Laststufen des Prüfstandes

Zur Anpassung der Beschleunigungsdaten (Häufigkeitsverteilung in Abb. 3.12) auf den Prüfstand ist eine neue gleichmäßige Verteilung notwendig. Insgesamt stehen vier Leistungspfade und ein Ruhepfad (kein Lastrelais angezogen) zur Verfügung. Die 201 Klassen in der Häufigkeitsverteilung (Abb. 3.12) müssen somit auf fünf Klassen verteilt werden. Jede dieser neuen Klassen entspricht einem Lastpfad/ einer Relaisstellung des Prüfstandes wie im Folgenden dargestellt:

- Ruhephase
  - Die beschleunigungslose Phase (Ruhephase oder Klasse 0) wird durch die Klassen von $-0{,}45$ bis $0{,}5\,\mathrm{m/s^2}$ gebildet (siehe grüne Hinterlegung in Abb. 3.12). Für die Umsetzung im Prüfstand bedeutet dies, dass kein Relais angezogen hat und kein Strom fließt.

- 2 Beschleunigungsphasen - Batterie wird entladen
  - Die erste Beschleunigungsphase (Klasse 1) wird durch die Klassen von $0{,}5$ bis $1\ \mathrm{m/s^2}$ gebildet (siehe hellblaue Hinterlegung in Abb. 3.12). Die obere Grenze von $1\ \mathrm{m/s^2}$ ergibt sich aus der gleichmäßigen Aufteilung der verbleibenden Häufigkeiten in die Klassen 1 und 2.
  - Die zweite Beschleunigungsphase (Klasse 2) wird durch die Klassen von 1 bis $3\ \mathrm{m/s^2}$ gebildet (siehe blaue Hinterlegung in Abb. 3.12).
- 2 Bremsphasen - Batterie wird geladen
  - Die erste Bremsphase (Klasse -1) wird durch die Klassen von $-0{,}45$ bis $-1\ \mathrm{m/s^2}$ gebildet (siehe hellrote Hinterlegung in Abb. 3.12). Die Untergrenze von $-1\ \mathrm{m/s^2}$ ergibt sich aus der gleichmäßigen Aufteilung der verbleibenden Häufigkeiten in den Klassen -1 und -2.
  - Die zweite Beschleunigungsphase (Klasse -2) wird durch die Klassen von 1 bis $3\ \mathrm{m/s^2}$ gebildet (siehe rote Hinterlegung in Abb. 3.12).

In Abb. 3.13 ist die neue Verteilung dargestellt. Die Achsenbeschriftungen beziehen sich auf die Häufigkeiten der einzelnen Lastpfade/Relaisstellungen und deren Klassen.

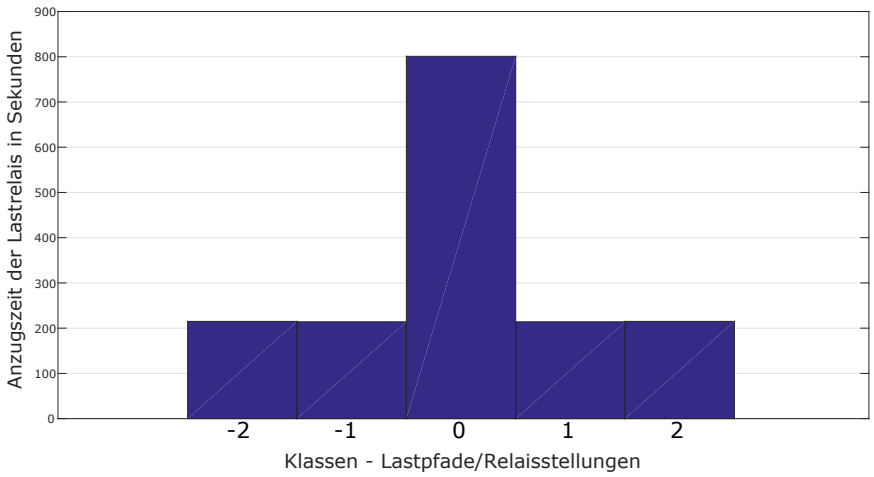


Abbildung 3.13.: Verteilung der Lastpfade/Relaisstellungen

Die Abb. 3.14 zeigt den zeitlichen Verlauf der Beschleunigung und der daraus resultierenden Klassen bzw. Lastpfade/Relaisstellungen.

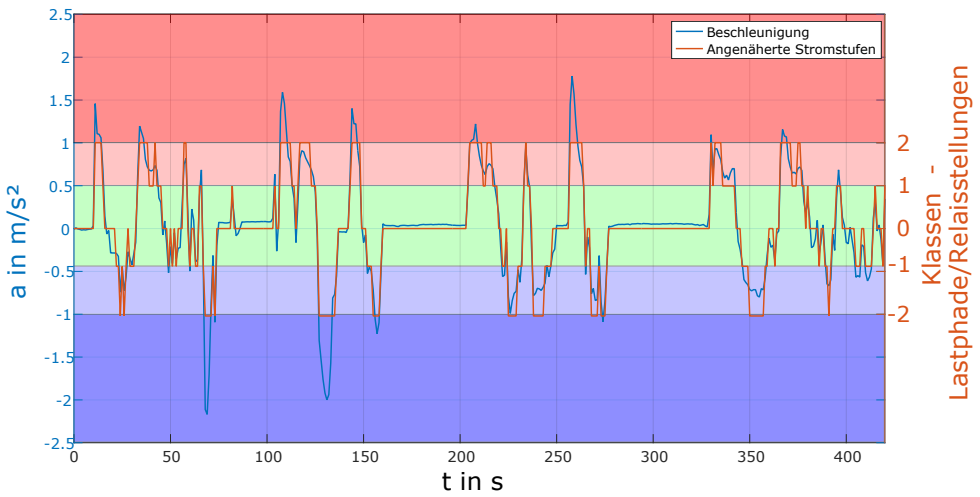


Abbildung 3.14.: Ausschnitt des zeitlichen Verlaufs der Beschleunigung sowie der einzelnen Klassen - Lastpfade/Relaisstellungen

Die tatsächlich aufgezeichnete Beschleunigung und der daraus resultierende Verlauf der Klassen bzw. Lastpfade/Relaisstellungen haben eine Dauer von 27 min. Aus technischen Gründen ist eine Kürzung auf 7 min notwendig. Dies ist dadurch begründet, dass der Zyklus nach der Konvertierung in die 'config.txt' bereits 132 der maximal möglichen 150 Zykliereinträge (in Abschnitt 5.1.4 vergrößert dargestellt) beansprucht. Durch die ebenfalls entwickelte Loop-Funktion lässt sich dieser Zyklus beliebig oft wiederholen und verschachteln. Der in Abb. 3.14 abgebildete 7 min lange Zyklus wurde durch vier Loops auf 28 min für den Probebetrieb der Zellen verlängert.

3.2.1.3. Festlegungen der Ströme für die einzelnen Lastpfade/Relaisstellungen

Für den Zyklus-I wurden für die LiNANO- und A123-Batterie folgende Lade- und Entlade-Ströme festgelegt (siehe Tabelle 3.4). Diese Ströme wurden so gewählt, dass die Batterie über den Verlauf des Zyklus entladen wird.

Tabelle 3.4.: Festlegung der Lade- und Entladeströme für den Zyklus Zyklus-I

Batterietyp	Laden 2 mit Relais 4	Laden 1 mit Relais 3	Entladen 1 mit Relais 2	Entladen 2 mit Relais 1	Dauer des Zyklus
LiNANO	16 A	8 A	12 A	24 A	28 min
A123	15 A	7,5 A	15 A	25 A	28 min

3.2.1.4. Berechnung der Geschwindigkeiten aus den gemessenen Beschleunigungen

Die Beschleunigungswerte wurden integriert, um die Geschwindigkeit zu berechnen. Dadurch, dass die Beschleunigungswerte, je nach Lage des Busses mit einem Offset versehen waren, driftete die Geschwindigkeit stark ab. Es wurden Versuche unternommen, um den Integrationsfehler durch Ausblenden der Standzeiten und/oder durch eine Neukalibrierung an jeder Haltestelle zu minimieren. Dies erwies sich als zeitintensiv und erfolglos. Dadurch war es nicht möglich, aus dem Fahrverhalten über die Formel (2.30) die Leistung bzw. den Strom der Batterie zu bestimmen.

Eine parallele Messung der Geschwindigkeit über GPS zu den Beschleunigungsdaten ist empfehlenswert und wurde in der Arbeit von Felix Groth [15] implementiert.

3.2.2. Entwurf und Erstellung eines Zyklus aus den EDDA-Daten (Zyklus-E)

In diesem Abschnitt wird aus den vom Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme bereitgestellten Daten ein Zyklus für den Prüfstand erstellt. Dazu wurden sowohl die zur Verfügung gestellten Daten zur Fahrdynamik als auch die elektrischen Daten näher untersucht.

3.2.2.1. Ermittlung der Leistungsprofile aus den gemessenen Beschleunigungen und Geschwindigkeiten

In dem Abschnitt 2.3.1 wurde die aus der Fahrdynamik eines Elektrobusses mathematisch resultierende Batterieleistung hergeleitet. Basierend auf dieser Herleitung und den Kenndaten des EDDA-Busses wurde ein zweiter Zyklus erstellt. Dieser basiert auf den Formeln (2.28) und (2.30), woraus sich zusammengefasst die Formel (3.4) ergibt.

$$
\begin{aligned}
P_R = \Bigg(& f_R \cdot m_{\text{ges}} \cdot g \cos(\gamma) \cdot v \\
& + m_{\text{ges}} \cdot g \cdot \sin(\gamma) \cdot v \\
& + m_{\text{ges}} \cdot a \cdot v \\
& + \tfrac{\rho_l}{2} \cdot A_x \cdot c_w \cdot v^3 \Bigg) \quad / \quad \eta_{\text{A}}
\end{aligned}
\tag{3.4}
$$

Die Kenndaten des EDDA-Busses sind aus der Tabelle 2.4 entnommen. Des Weiteren wurden folgende Variablen definiert bzw. angenommen. An dieser Stelle werden die Variablen kurz zusammengefasst und beschrieben.

- Der Rollwiderstandsbeiwert wurde mit $f_R = 0{,}01$ aus der Literatur [29] entnommen.
- Der Steigungswinkel wurde mit $\gamma = 0°$ gewählt, da für die Strecke die Topographie nicht bekannt war.
- Der Luftwiderstandsbeiwert $c_w = 0{,}9$ wurde der Literatur [29] entnommen.
- Die Temperatur $\vartheta = 8\,°\mathrm{C}$ wurde für den 1.12.2014, an dem die EDDA-Messfahrt stattgefunden hat, im Raum Dresden über das Internet ermittelt [43].
- Der Luftdruck $p = 1021\,\mathrm{hPa}$ wurde analog zur Temperatur für den 1.12.2014 ermittelt [43].
- Die aus der Temperatur und dem Luftdruck resultierende Luftdichte berechnet sich nach der Formel (3.5) [36].

$$\rho_l = 1,293 \cdot \frac{273}{\vartheta + 273} \cdot \frac{p}{1013} \cdot \frac{\mathrm{kg}}{\mathrm{m}^3} = 1{,}2661\,\frac{\mathrm{kg}}{\mathrm{m}^3} \tag{3.5}$$

- Der Antriebswirkungsgrad wurde, wie im Abschnitt 2.3.2 erläutert, mit $\eta_A = 90\,\%$ angenommen.
- Die wirksame Querschnittsfläche des Fahrzeuges ergibt sich durch die Höhe und Breite des Busses (siehe (3.6)).

$$A_x = h \cdot b = 3{,}36\,\mathrm{m} \cdot 2{,}25\,\mathrm{m} = 7{,}56\,\mathrm{m}^2 \tag{3.6}$$

Anhand dieser Annahmen bzw. Parameter und der Formel für die Zugkraft (2.25) lässt sich für verschiedene Beschleunigungen folgende Abb. 3.15 erstellen.

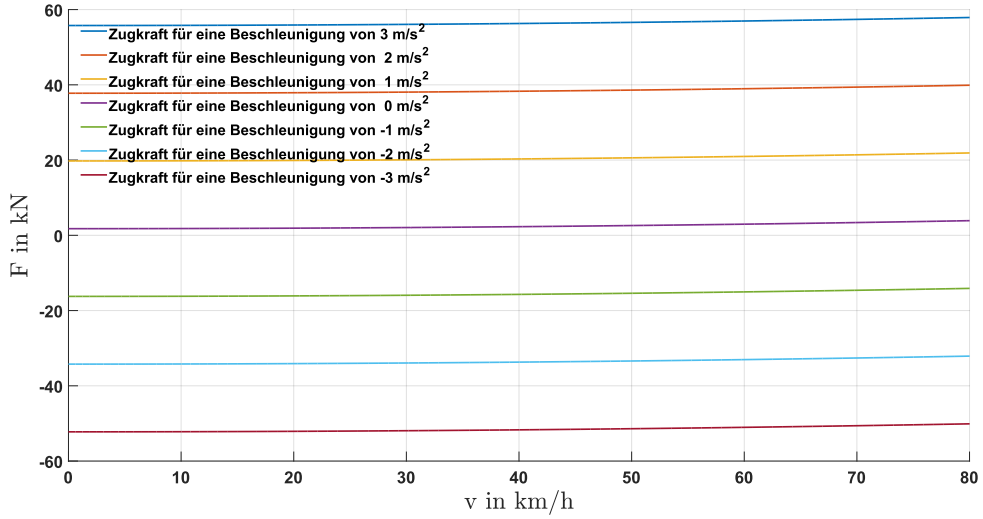


Abbildung 3.15.: Benötigte Zugkraft bei verschiedenen Beschleunigungen und keiner Steigung, am Beispiel des EDDA Busses.

Wird die Zugkraft mit der Geschwindigkeit multipliziert, wie in der Gleichung (2.28) beschrieben, ergibt sich die Radleistung, die in der Abbildung Abb. 3.16 abgebildet ist.

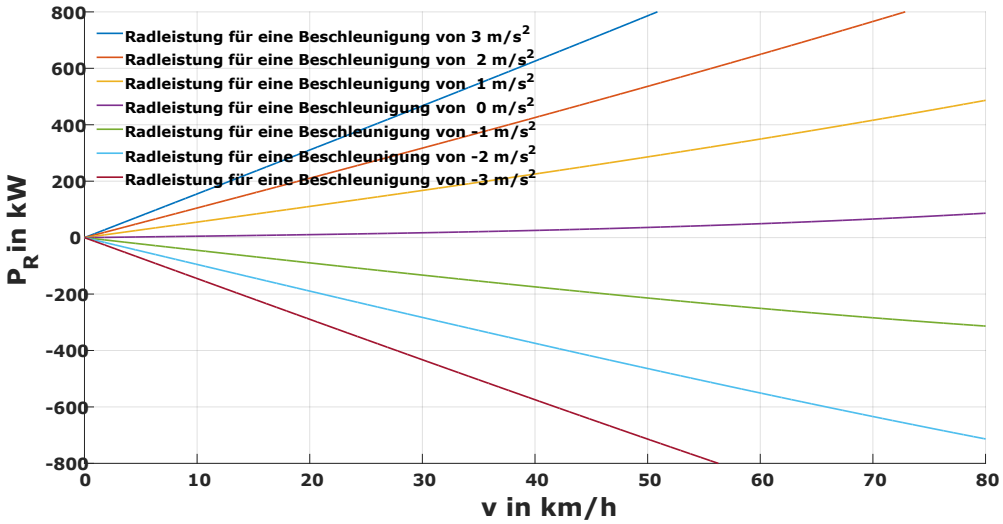


Abbildung 3.16.: Benötigte Radleistung bei verschiedenen Beschleunigungen und keiner Steigung, am Beispiel des EDDA Busses.

In der Abb. 3.16 ist deutlich der nicht proportionale Anstieg der benötigten Radleistung bei

einer Geschwindigkeitserhöhung zu erkennen. Dieser ist auf den Einfluss des aerodynamischen Widerstandes zurückzuführen, der wie in Formel (2.28) verdeutlicht, mit v^3 in die Radleistung eingeht.
In Abb. 3.17 ist die Radleistung in Abhängigkeit von der Beschleunigung bei verschiedenen konstanten Geschwindigkeiten dargestellt. Deutlich ist der proportionale Zusammenhang zwischen Leistung und Beschleunigung zu erkennen.

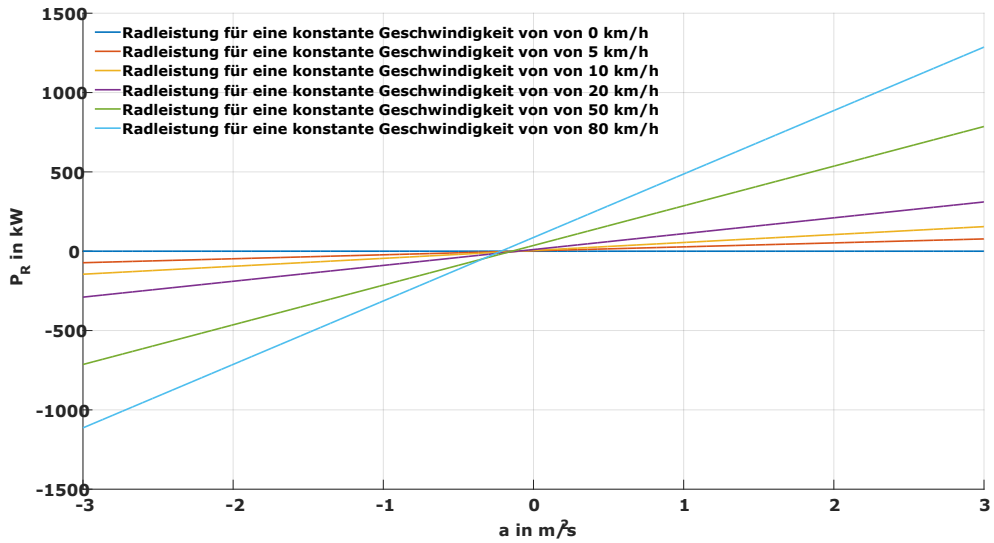


Abbildung 3.17.: Benötigte Radleistung in Abhängigkeit der Beschleunigung bei konstanten Geschwindigkeiten und keiner Steigung.

Durch die Eingabe der gemessenen Geschwindigkeiten und Beschleunigungsdaten des EDDA-Busses in das Modell (3.4), ergibt sich unter Berücksichtigung des Wirkungsgrades das Leistungsprofil Abb. 3.18. In dem Vergleich 3.2.2.2 wird die modellierte Leistung mit der tatsächlichen, durch die Messungen des Stromes und der Spannung ermittelten elektrischen Leistung, gegenübergestellt. Die Auswertung ergibt, dass die Leistungen, die durch das Modell ermittelt wurden, gut mit den wirklichen bzw. realen gemessenen Leistungen übereinstimmen.

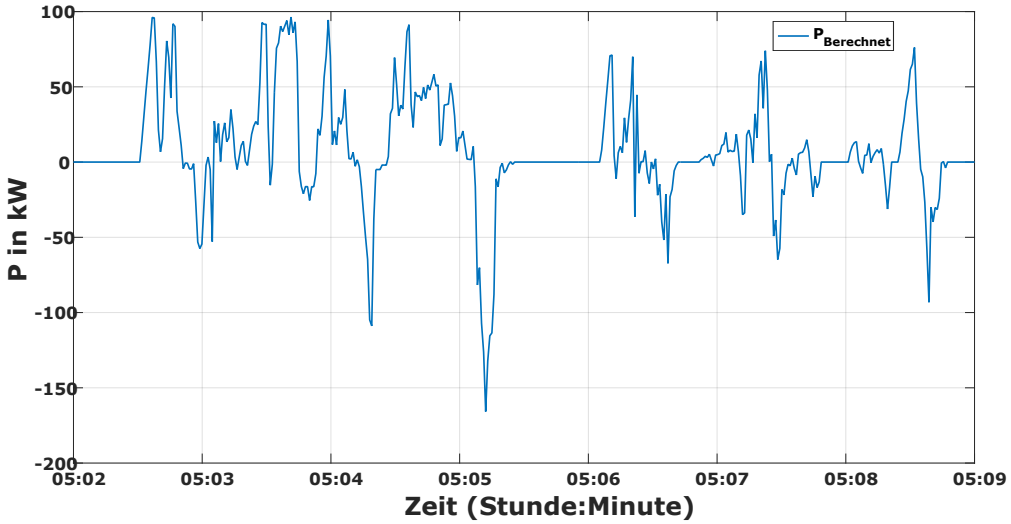


Abbildung 3.18.: Ein 7 min Auszug[2] des berechneten Leistungsprofils des EDDA Busses. Positive Leistung entspricht einem Beschleunigungsvorgang und negative Leistung einem Bremsvorgang (Rekuperation).

3.2.2.2. Ermittlung des Leistungsprofils aus den gemessenen Strömen und Spannungen

Da bei den EDDA Messdaten auch die Spannungen und die Ströme mit aufgezeichnet wurden, lässt sich das entwickelte Modell mit Leistungen im Feldbetrieb vergleichen. Über die gemessene Zwischenkreisspannung und den Strom in dem Fahrmotor lässt sich die Leistung des Fahrmotors $P_{\text{Fahrmotor}}$ errechnen und darstellen.

$$P_{\text{Fahrmotor}} = I_{\text{Fahrmotor}} \cdot U_{\text{Zwischenkreis}} \tag{3.7}$$

Ein Vergleich der im Modell bestimmten Leistung (Kapitel 3.2.2.1) und der elektrischen Leistung ist in Abb. 3.19 dargestellt. Der vollständige Plot der Fahrt ist im Anhang Abb. C.21 abgebildet. Zu erkennen ist eine Verzögerung von ca. 2 s (in rot markiert) zwischen den beiden Leistungsprofilen. Gründe für diese Verzögerung können die Trägheit des Busses, Auflösung des GPS oder die sekündliche Abtastung des GPS sein.

Durch eine Zeitkorrektur von 2 s lassen sich die beiden Leistungskurven übereinander legen (siehe Abb. 3.20). Zu sehen ist, dass die berechnete zurückgewonnene Leistung größer ausfällt als die am Fahrmotor gemessene. Zu vermuten ist, dass der Bus zusätzlich zu den elektrischen Bremsen noch mechanisch gebremst wird.

[2] Das vollständige berechnete Leistungsprofil ist im Anhang Abb. C.21 abgebildet.

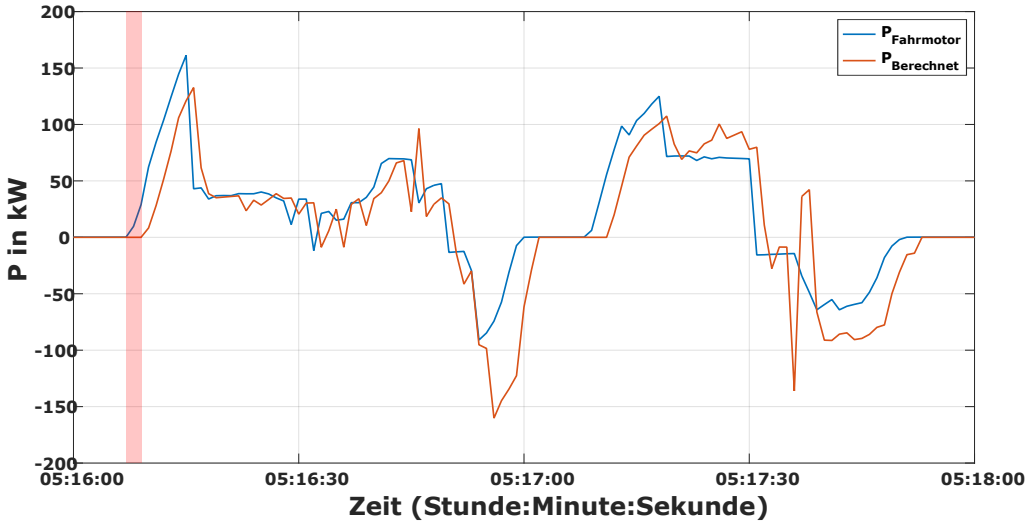


Abbildung 3.19.: Die Abbildung zeigt den Ausschnitt[3] einer Fahrt des EDDA-Busses. Zu sehen ist die tatsächliche elektrische Fahrleistung und die aus dem Modell errechnete elektrische Leistung des Busses.

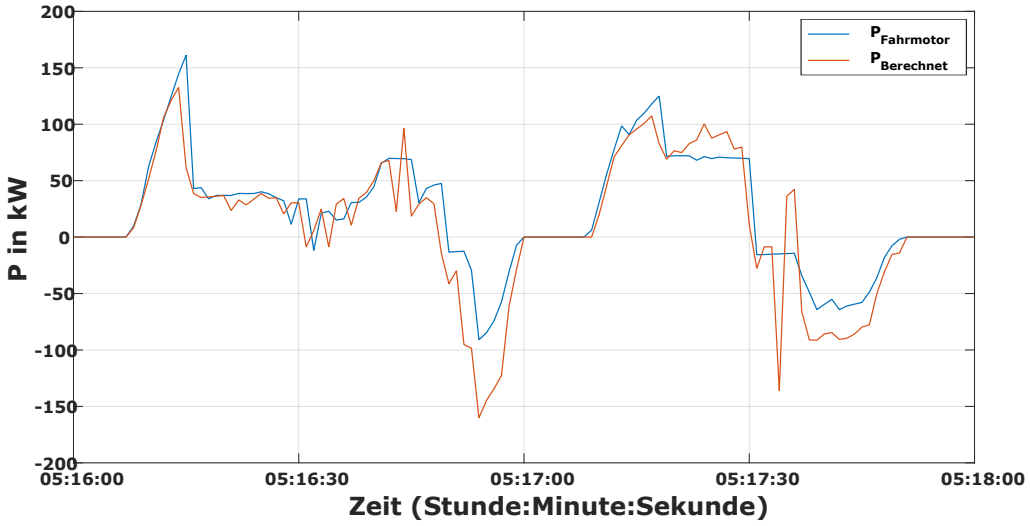


Abbildung 3.20.: Die Abbildung zeigt den Ausschnitt einer Fahrt des EDDA-Busses. Zu sehen ist die tatsächliche elektrische Fahrleistung und die aus dem Modell errechnete elektrische Leistung des Busses ohne die 2 s Verzögerung.

[3] Das vollständige Vergleich der Leistungsprofile ist im Anhang Abb. C.21 dargestellt.

3.2.2.3. Erstellung des EDDA-Zyklus aus den gemessenen Strömen

Bei der Erstellung des EDDA-Zyklus wurden nicht die Leistungen aus dem Modell, sondern die aufgezeichneten Ströme genutzt. Dazu wurde mit den Stromdaten $I_{\mathrm{Bat2.3}}$ der Batterie 2 und der Batterie 3 gearbeitet. Da der Batterieaufbau (siehe Kapitel 2.2.2) bekannt ist, kann der Strom für eine einzelne Zelle berechnet werden. Batterie 2 und Batterie 3 sind über den DC/DC-Wandler parallel geschaltet. Der Strom $I_{\mathrm{Bat2.3}}$ teilt sich somit in den Strom I_{Bat2} und I_{Bat3} auf. Durch die Annahme, dass er gleiche Strom in die beiden Batterien fließt, lässt sich der Strom in einer Batterie wie in Formel (3.9) vereinfacht darstellen.

$$I_{\mathrm{Bat2.3}} = I_{\mathrm{Bat2}} + I_{\mathrm{Bat3}} = 2 \cdot I_{\mathrm{Bat}} \tag{3.8}$$

$$I_{\mathrm{Bat}} = \frac{I_{\mathrm{Bat2.3}}}{2} \tag{3.9}$$

In der Abbildung 3.21 ist der Strom I_{Bat} dargestellt. Dadurch, dass eine Batterie des EDDA-Busses (wie in Kapitel 2.2.2 beschrieben) aus 168 in Reihe geschalteten $46\,\mathrm{Ah}$ Zellen besteht, ist der Strom in jeder Zelle gleich. Deutlich zu erkennen ist in der Darstellung Abb. 3.21, dass die Batterie mit maximal $90\,\mathrm{A}$ entladen und mit $50\,\mathrm{A}$ geladen wird. Dies entspricht einer Entladung von ca. $2\,\mathrm{C}$ und einer Ladung von ca. $1\,\mathrm{C}$. Des Weiteren zeigt die Darstellung Abb. 3.21, dass die Batterie stärker entladen als geladen (rekuperiert) wird. Durch die Integration der positiven ($40\,\mathrm{kAs}$) und der negativen ($8\,\mathrm{kAs}$) Ströme ergibt sich ein Verhältnis von $26\,\%$. Auch zu sehen ist, dass die Belastung sehr stufig ist, welches die Wahl der Stufen vereinfacht.

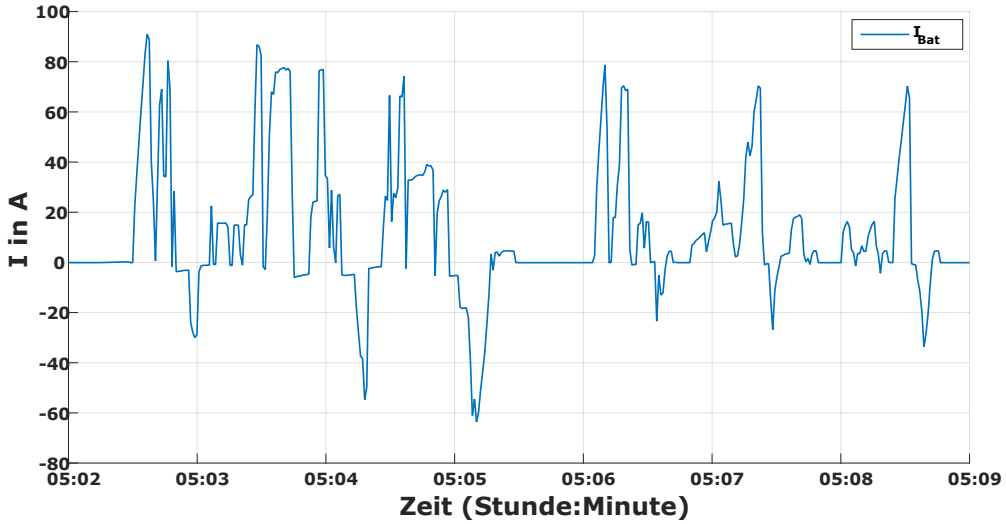


Abbildung 3.21.: Ein Auszug[4] des Stromes in die Batterie des EDDA Busses.
Positiver Strom (Entladung) - Beschleunigungsvorgang
Negativer Strom (Ladung) - Bremsvorgang (Rekuperation)

Für die Erstellung dieses Zyklus wurden in Abstimmung mit dem Projekt BATSEN folgende Randbedingungen und Vorgehensweisen definiert:

- Der Zyklus basiert auf dem Zeitverhalten des EDDA-Busses.
- Die fünf Strompfade des Prüfstandes werden aus den gemessenen Strömen des EDDA-Busses abgeleitet.
- Der Batterietyp wird nach den vom Prüfstand gegebenen Rahmenbedingungen gewählt.

Bei der Erstellung dieses Zyklus wurden die Stromstufen, wobei Minus die Ladung der Batterie und Plus die Entladung der Batterie kennzeichnet, wie folgt gewählt.

- Im Bereich von $\pm 2{,}3\,\mathrm{A}$ gibt es eine Ruhezone (0) in der kein Strom fließt (grün).
- Im Bereich von $2{,}3$ bis $46\,\mathrm{A}$ gibt es die erste Entladestufe (+1) mit einer C-Rate von $1\,\mathrm{C}$ (hellblau).
- Der Strom, der größer als $46\,\mathrm{A}$ ist, ergibt die zweite Entladestufe (+2) mit einer C-Rate von $2\,\mathrm{C}$ (blau).
- Im Bereich von $-2{,}3$ bis $-23\,\mathrm{A}$ gibt es die erste Ladestufe (-1) mit einer C-Rate von $0{,}5\,\mathrm{C}$ (hellrot).

[4] Das vollständige Stromprofil der Batterie ist im Anhang Abb. C.22 dargestellt.

- Der Strom, der kleiner als $-23\,\mathrm{A}$ ist, ergibt die zweite Ladestufe (-2) mit einer C-Rate von $1\,\mathrm{C}$ (rot).

In der Abb. 3.22 ist ein Ausschnitt des Zyklus zu sehen, in dem die einzelnen Stufen noch einmal enthalten sind.

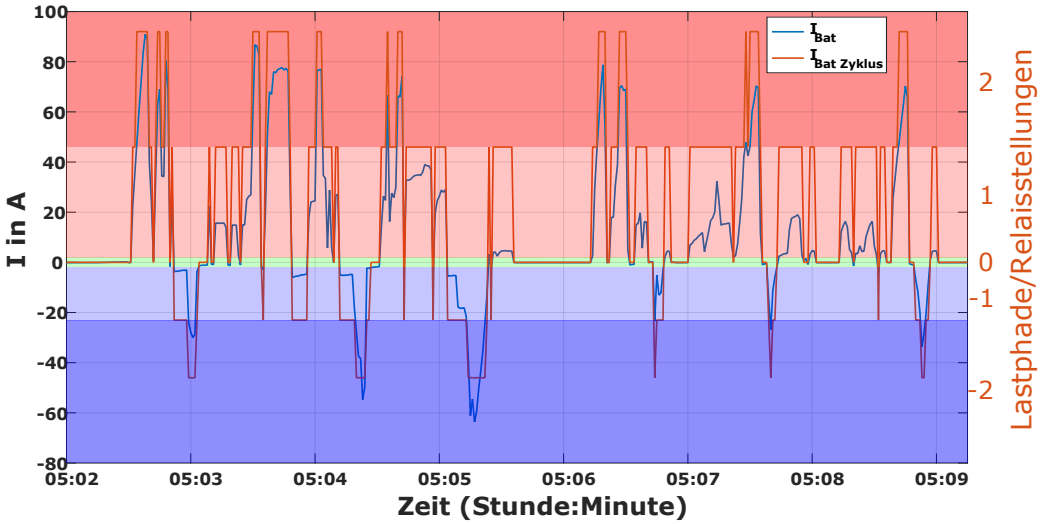


Abbildung 3.22.: Ein Auszug[5] des Stromverlaufs in die Batterie des EDDA Busses. Positiver Strom entspricht einem Beschleunigungsvorgang, ein negativer Strom einem Bremsvorgang (Rekuperation). Die Strompfade sind farbig dargestellt.

Durch die Angaben in C lässt sich dieser Zyklus ohne Weiteres auf andere Batterien übertragen. Die A123-Batterie wird zur Zyklierung dieses Zyklus verwendet. Wie schon in der Tabelle gezeigt, ist diese die einzige Batterie, die mit einer C-Rate von 2C entladen werden kann.
Wie schon in dem Abschnitt 3.2.1 beschrieben, ist die Begrenzung der 'config.txt' des Zykliergerätes ein Problem. Aus diesem Grunde musste der Zyklus auf den in Abb. 3.22 abgebildeten Ausschnitt von $7\,\mathrm{min}$ begrenzt werden. Durch die Verwendung von sechs Loops wurden die $7\,\mathrm{min}$ auf $42\,\mathrm{min}$ verlängert und beschreiben dadurch annäherungsweise eine vollständige Fahrt des EDDA-Busses.

[5] Das vollständige Stufenprofil des EDDA-Busses ist im Anhang Abb. C.23 abgebildet.

3.2.2.4. Festlegungen der Ströme für die einzelnen Lastpfade/Relaisstellungen

Für den Zyklus-E wurden für die A123-Batterie folgende Lade- und Entlade-Ströme festgelegt (siehe Tabelle 3.5).

Tabelle 3.5.: Festlegung der Lade- und Entladeströme für den Zyklus Zyklus-E

Batterietyp	Laden 2 mit Relais 4	Laden 1 mit Relais 3	Entladen 1 mit Relais 2	Entladen 2 mit Relais 1	Dauer des Zyklus
A123	2,5 A	1,25 A	2,5 A	5 A	42 min

4. Konzeption der Prüfstandsanpassungen

Die Projektgruppe BATSEN testet und erprobt Batterien in einem eigens dafür eingerichteten Batterielabor. In diesem Labor werden Batterien modifiziert, Prüfstände aufgebaut und sowohl Kurzzeit- als auch Langzeittests von Batterien und Halbzellen durchgeführt. In dem folgenden Abschnitt wird der bereits vorhandene, von Martin Kusche [26] übernommene, Batterie-Zyklier-Prüfstand beschrieben. Des Weiteren werden neue Anforderungen an den Prüfstand definiert. Diese sind nötig, um die unter Kapitel 3.2 entwickelten Zyklen an den in Tabelle 2.1.4 aufgeführten $LiFePO_4$-Batterien mit dem optimierten Prüfstand testen und zyklieren zu können.

4.1. Das Zykliergerät

Das Herzstück des gesamten Messaufbaus bildet das sogenannte Zykliergerät. Das Zykliergerät ist ein ARM-Mikrocontroller gesteuertes Gerät, dass es ermöglicht, eine Batterie überwacht zu laden und zu entladen. Es übernimmt die minutenweise Messwertaufzeichnung von Strom, Spannung und Temperatur von bis zu 12 in Reihe geschalteten Zellen und schreibt die Messwerte auf eine SD-Speicherkarte. Über vier Relais können bis zu vier verschiedene Lasten bzw. Quellen an die Batterie angeschlossen werden, wobei allerdings nur ein einziges der vier Relais zugleich geschaltet werden kann (Hardwareverriegelung). Über eine auf der SD-Karte hinterlegte *config.txt* lässt sich das Zykliergerät konfigurieren. In Abb. 4.1 ist ein Prinzipaufbau des Zykliergerätes zu sehen.

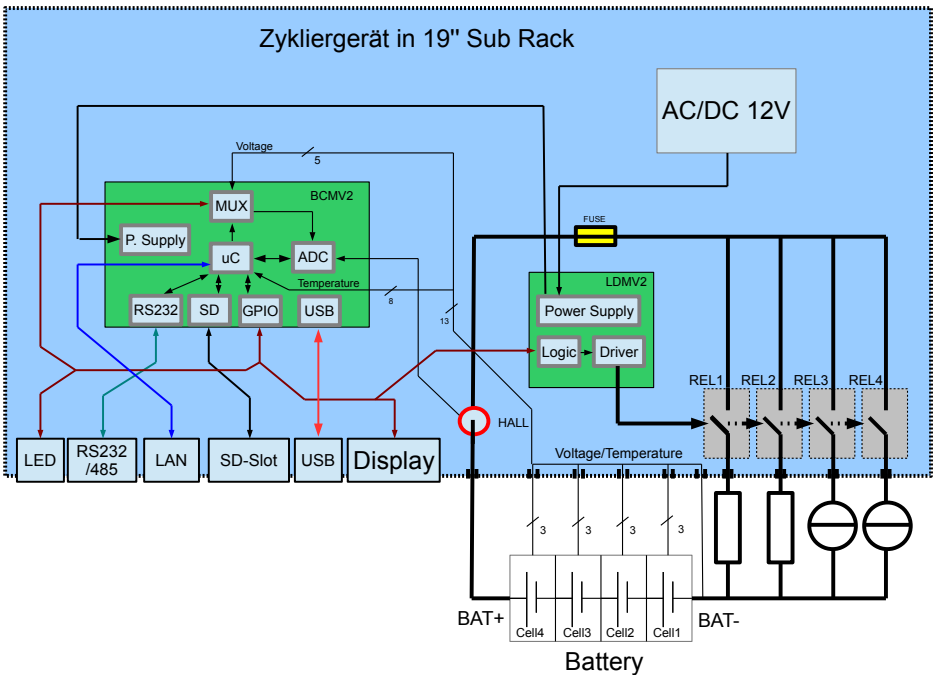


Abbildung 4.1.: Prinzipschaltbild des Zykliergerätes am Beispiel von vier Zellen [44]

4.2. Weitere Komponenten des Prüfstandes

4.2.1. Der Temperaturschrank

Da die Batterieeigenschaften, wie z. B. der SOC, stark temperaturabhängig sind und Batterien nur in einem eingeschränkten Temperaturbereich betrieben werden dürfen, ist der Aspekt der Temperatur sehr wichtig. Der Temperaturschrank 'ESPEC ARU-0680' ermöglicht es, einen Prüfling, in diesem Fall die Batterie, bei verschiedenen Temperaturen zu betreiben. Dazu kann der Schrank in einem konstanten Modus oder in einem programmierbaren Modus gefahren werden. Des Weiteren können durch einen Temperaturschrank auch reproduzierbare Messungen durchgeführt werden, da das Umfeld konstant ist. In der Tabelle 4.1 werden die Kenndaten des Temperaturschrankes aufgeführt.

Tabelle 4.1.: Kennwerte des Temperaturschrankes Typ ARU-0680 der Firma: ESPEC [10]

Bezeichnung	Daten
Temperaturbereich	-75 bis $180\,°C$
Temperaturabweichung	$\pm 0{,}3\,K$
Temperaturänderungs-geschwindigkeit durch Heizen	$6{,}3\,K/min$
Schnittstellen	RS-485/RS-232C

4.2.2. Lastwiderstände und aktive Lasten

Die Entladung der Batterie kann über passive Lastwiderstände (Kennwerte in Tabelle 4.2 dargestellt) oder auch über eine aktive elektronische Last erfolgen.

Tabelle 4.2.: Kennwerte der passiven Lastwiderstände

Name	Lastpoti 1	Lastpoti 2	Lastwiderstand [5]
Typ	Potentiometer	Potentiometer	Widerstand
Anzahl	2	1	40
Nennwiderstand	$0{,}66\,\Omega$	$1\,\Omega$	$0{,}47\,\Omega$
Nennstrom	$25\,A$	$20\,A$	n. a.
Belastbarkeit	n. a.	n. a.	$50\,W$

Die elektronische Last kann in dem so genannten Konstantstrombetrieb eingesetzt werden und dient somit als Stromsenke. Der Vorteil einer elektronischen Last ist es, dass sie, trotz sich verändernder Spannung, den eingestellten Strom konstant hält. Dies ermöglicht die konstante Belastung einer Batterie, unabhängig von ihrem Ladezustand. Die in dem Prüfstand verwendete Last wird in der Tabelle 4.3 anhand ihrer Kenndaten beschrieben. Der Prüfstand wurde zu Beginn der Arbeit mit einer elektronischen Last übernommen. Für die Zyklierung der Busprofile wurde eine zweite baugleiche elektronische Last erworben.

Tabelle 4.3.: Kennwerte der elektronischen Last Typ 3227.1 der Firma Statron

Bezeichnung	Daten
Lastgleichspannung	1 bis 80 V
Lastgleichstrom	0,005 bis 25,5 A
Lastwiderstand	0,01 bis 16 000 Ω
Stabilität des 25A Laststroms	0,1 % / 50 mA
Nennbelastbarkeit	200 W
maximale Belastbarkeit <5 min	400 W

4.2.3. Die Spannungs- bzw. Stromquellen

Für die Ladung der Batterien stehen verschiedene lineare Gleichspannungsquellen zur Verfügung. In der Tabelle 4.4 sind die technischen Daten der Quellen dargestellt.

Tabelle 4.4.: Kennwerte der Spannungs- bzw. Stromquellen

Name	NGT 35	PE 1644 2	PE 1516
Hersteller	Rohde & Schwarz	Philips	Philips
Anzahl	1	1	2
Konstantstrom	0 bis 5 A	0 bis 10 A	0 bis 20 A
Konstantspannung	0 bis 5 V	0 bis 40 V	0 bis 30 V

4.3. Vorhergehende Arbeiten über den Aufbau des Zykliergerätes

Das erste Zykliergerät wurde als Prototyp von Thomas Wisniewski in seiner Bachelorarbeit mit dem Titel „Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Mikrocontroller sowie Messdatenverwaltung und Netzwerkanbindung“ [44] konzipiert. Diese Arbeit hatte zum Ziel, einen Prüfstand für mehrere Batteriezellen zu konzipieren. Dieser sollte sowohl messtechnische als auch überwachungsstechnische Aufgaben durch Spannungs-, Strom- und Temperaturmessungen bei der Batteriezyklierung übernehmen. Dazu sollte über eine Umschaltmatrix ein messtechnischer Zugang zu den verschiedenen Zellen ermöglicht werden. Des Weiteren sollte es dem Zykliergerät möglich sein, Lade- und Entladeströme der Batterien über Lastrelais zu steuern. Das von Wisniewski entwickelte Zykliergerät wird im Folgenden als „Zykliergerät W“ bezeichnet.

In der Bachelorarbeit „Batteriezellen-Zyklierprüfstand mit ARM-Mikrocontroller Software sowie Mess- und Leistungsschaltung“ [34] von Johannes Röhn wurde ein zweites Zykliergerät gebaut, welches höhere Lade- und Entladeströme durch größere Lastrelais zulässt. Zudem wurde die Software erweitert. Das hier entstandene Zykliergerät wird im Folgenden als „Zykliergerät R“ bezeichnet.

Martin Kusche hat mit seiner Bachelorarbeit „Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller“ [26] an den Stand des Zykliergerätes R angeknüpft. Er hatte die Aufgabe, den neuen Europäischen Fahrzyklus für Elektrofahrzeuge auf dem Prüfstand zu implementieren und zu testen. Des Weiteren sollte Martin Kusche Probleme, die in den vorangegangenen Erprobungen erkannt worden sind, untersuchen und beheben, wie z. B. die Empfindlichkeit des verwendeten ADC gegen Überspannungen und fehlerhaften Masseführungen. Dazu wurde eine neue Analog-Digital-Wandler (ADC) Platine entwickelt.

In den genannten Bachelorarbeiten wurde mit der von Thomas Wisniewski entwickelten Platine V2.0 gearbeitet. Unabhängig von den Bachelorarbeiten hat der wissenschaftliche Mitarbeiter Dipl. -Ing. Matthias Schneider eine modifizierte Platine V2.1 entwickelt, die bislang nicht in die bestehenden Zykliergeräte eingebaut und somit auch nicht getestet wurde. Diese Platine wird in Abschnitt 5.2.1 für einen Neuaufbau des Zykliergerät verwendet.

Da zwei Zykliergeräte den Ausgangspunkt dieser Arbeit bildete, werden die Kennwerte bzw. Unterschiede der beiden Geräte in der Tabelle 4.5 erläutert.

Tabelle 4.5.: Kennwerte der Zykliergeräte

	Zykliergerät R	Zykliergerät W
Hardwarestand Hauptplatine	V2.0	V2.0
Hardware Erweiterungen	ADC-Platine	keine
Anzahl Lastrelais 30 A	3	3
Anzahl Lastrelais 500 A	1	0
Anzahl Temperatur und Spannungskanäle	12	12
Maximale Zellspannung	5 V	5 V
Maximaler Laststrom	500 Abzw. 30 A	30 A
Sicherung	500 A Schmelzsicherung	30 A Schmelzsicherung
Max. Anzahl Messungen pro Minute	1	1
Max. Anzahl Lastwechsel pro Minute	1	1
Softwarestand	20140912-Proto2Röhn-Kali	20140805-Proto1 WisinewskiReparatur
Kalibrierung	quadratische Kalibrierung	lineare Kalibrierung

4.4. Inbetriebnahme der Zykliergeräte

Einer der wesentlichen Unterschiede der beiden Geräte besteht darin, dass der erste der vier Lastpfade des Zykliergerätes R über ein Lastrelais verfügt, mit dem es Ströme mit bis zu 500 A schalten kann. Dies und die Einschränkung, dass das Zykliergerät W zu Beginn dieser Arbeit nur eingeschränkt verfügbar war, führte dazu, dass die Untersuchungen und Anpassungen an dem Zykliergerät R durchgeführt wurden. Das Zykliergerät W wurde erst später auf den Soft- und Hardwarestand des Zykliergerätes R nachgerüstet.

Bei der Inbetriebnahme des Zykliergerätes R wurde dieses mit einem Zellsimulator, bestehend aus einer Spannungsquelle/Frequenzgenerator und einem Widerstandsnetzwerk (Aufbau des Zellsimulators siehe B.1), getestet. Dieser Aufbau hat den Vorteil, dass der Test nicht im Batterielabor sondern am Arbeitsplatz durchgeführt werden kann. Bei der Inbetriebnahme und dem Probebetrieb des Zykliergerätes stellte sich heraus, dass die Messabläufe

nicht zuverlässig ausgeführt wurden. Dies führte zu einer detaillierten Fehlersuche, die unter B.6 dokumentiert ist.

Des Weiteren wurden die Schaltzeiten der Lastrelais und die Messrelaissteuerung näher untersucht, da sich bei den ersten Messungen (siehe 3.1.2) ergab, dass die Messung und Lastrelaisumschaltung je Minute viel zu grob für die Simulation eines Busverhaltens wäre. Damit zusammenhängend wurde auch die Messauflösung der Spannungsmessungen näher untersucht 4.4.2.

Zusätzlich zu der elektronischen Last passive passiven-Lastwiderständen zu Verfügung standen, wurden unter 4.4.3 der Innenwiderstand der Lastpfade des Zykliergerät ermittelt. da dieser einen wesentlichen Anteil einer passiven Last ausmachen können.

4.4.1. Untersuchung des Zeitverhaltens der Lastrelais

Das Zykliergerät R kann mit dem Relais 1 Ströme bis zu 500 A und mit dem Relais 2 bis 4 Ströme bis zu 30 A schalten. In Tabelle 4.6 sind die wesentlichen Kennwerte der beiden Relaistypen gegenübergestellt.

Tabelle 4.6.: Die wesentlichen Kennwerte der Lastrelais mit einem Nennstrom von 500 A und einem Nennstrom von 30 A.

	500 A Lastrelais [39]	30 A Lastrelais [37]
Typ	LEV200 Series Contactor	V2.0
Hersteller	tycp	TE
max kontinuierlicher Strom	500 A	30 A
Garantierte mechanische Schaltvorgänge	100 000	10 000 000
Benötigte Zeit zum Schließen (mit prellen)	40 ms	25 ms
Benötigte Zeit zum Öffnen	12 ms	25 ms
Innenwiderstand	0,2 mΩ	n. a.

Um die tatsächlichen Schaltvorgänge bzw. Schaltzeiten der Relais zu ermitteln, wurde mit einem Oszilloskop die Anzugs- bzw. Abfalldauer gemessen. Die Relais werden über eine MOSFET Schaltung angesteuert. Für die Ermittlung der Schaltzeiten wurde sowohl an dem Steuereingang als auch an dem Lastausgang des Lastrelais mit dem Oszilloskop gemessen.

Um das Schalten des Lastrelais am Ausgang zu detektieren, wurde eine Prüfspannung angelegt. Der Messaufbau ist in Abb. C.11 skizziert. Die Messung ist exemplarisch an einem Lastwechsel zwischen dem Lastpfad 1 und dem Lastpfad 2 durchgeführt. In der Abb. 4.2 ist das Anziehen des Relais 1 und das Abfallen des Relais 2 und in Abb. 4.3 das Abfallen des Relais 1 und das Anziehen des Relais 2 zu sehen. Deutlich sichtbar sind die farbig (grün und blau) markierten Totzeiten zwischen den steigenden bzw. fallenden Flanken. In der Tabelle 4.7 sind die gemessenen An- und Abfallzeiten (Totzeiten) dargestellt.

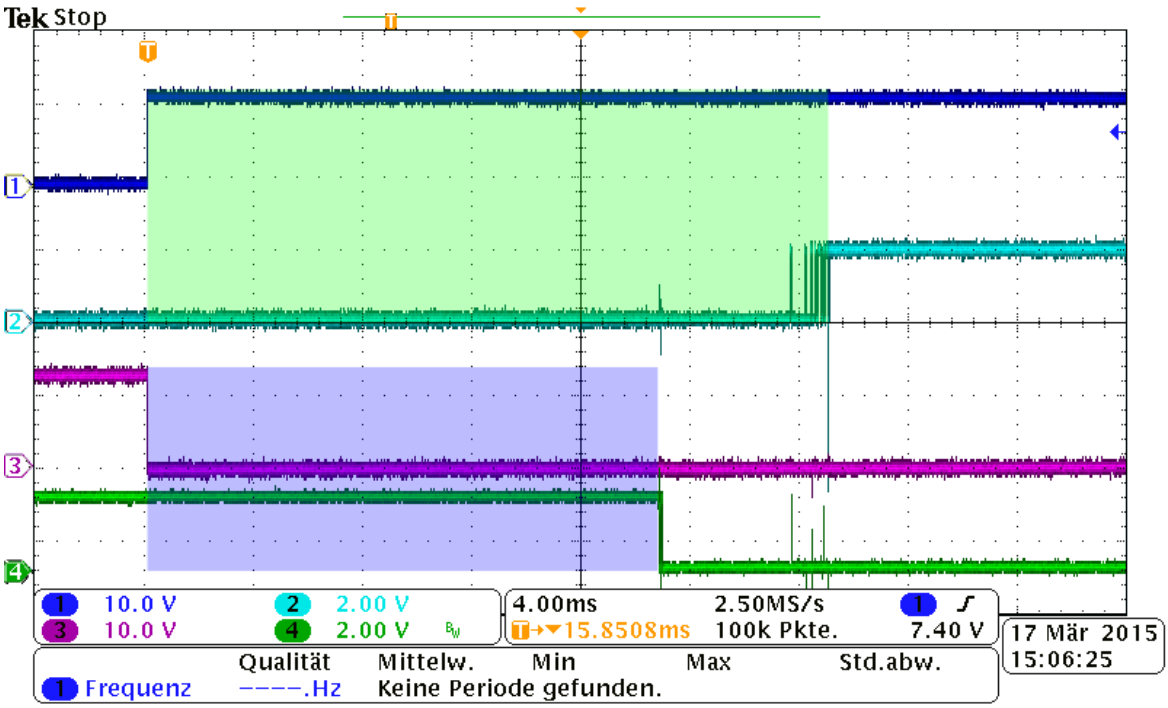


Abbildung 4.2.: Messung der Anzugszeit des Relais 1 und Abfallzeit des Relais 2, wobei Ch1 die Steuerspannung des Relais 1, Ch2 die Lastspannung des Relais 1, Ch3 die Steuerspannung des Relais 2 und Ch4 die Lastspannung des Relais 2 darstellt.

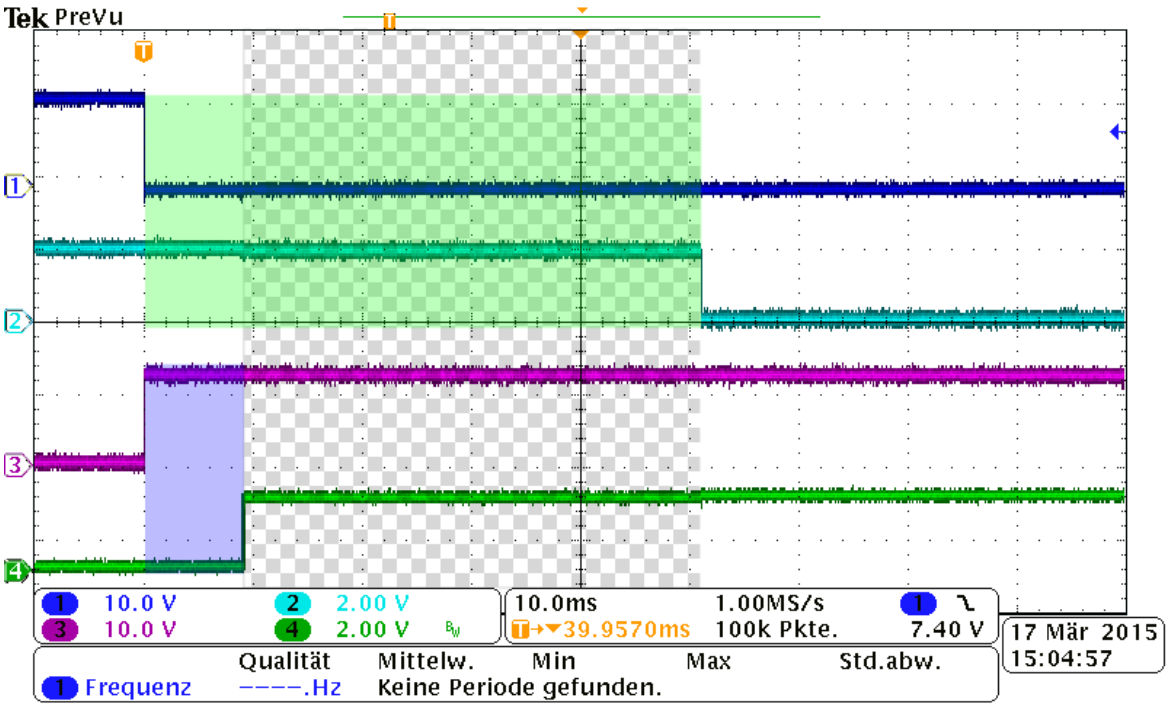


Abbildung 4.3.: Messung der Abfallzeit des Relay 1 und Anzugszeit des Relay 2, wobei Ch1 die Steuerspannung des Relay 1, Ch2 die Lastspannung des Relay 1, Ch3 die Steuerspannung des Relay 2 und Ch4 die Lastspannung des Relay 2 darstellt.

Tabelle 4.7.: Darstellung der gemessenen An- und Abfallzeiten (Totzeiten) aus den Oszilloskopmessungen Abb. 4.2 und Abb. 4.3

	500 A Lastrelais [39]	30 A Lastrelais [37]
Anzugszeit	25,6 ms	9 ms
Abfallzeit	50 ms	19,2 ms

Bei den Messungen wurde ein Problem mit den Schaltzeiten beobachtet. In der Abb. 4.3 ist in dem karierten Bereich zu erkennen, dass zur selben Zeit sowohl Relais 1 als auch Relais 2 angezogen sind. Dies ist darauf zurückzuführen, dass beide Relais ihr Steuersignal zum selben Zeitpunkt bekommen, das 500 A Lastrelais jedoch eine längere Abfallzeit benötigt, um vollständig zu öffnen als das 30 A Lastrelais zum Anziehen braucht. Diese Überschneidung tritt auch bei der Lastumschaltung zwischen zwei 30 A Lastrelais auf (siehe Abb. C.12). Im Fehlerfall hat dies zur Folge, dass die angeschlossene Batterie und das an dem jeweiligen Leistungspfad angeschlossene externe Gerät parallel geschaltet werden. Dies konnte durch

eine Softwareanpassung, die über eine zeitverzögerte Ansteuerung der Lastrelais realisiert ist, behoben werden.

4.4.2. Untersuchung des Zeitverhaltens der Strom- und Spannungsmessungen im Zykliergerät

Die Zellspannungsmessung an dem Zykliergerät erfolgt über einen externen Analog-Digital-Wandler (ADC) [2], der über Relais an die zu messende Zelle geschaltet wird (siehe Abb. C.13). Der ADC misst nacheinander die Zellspannungen 1 bis 12. Des Weiteren misst der ADC zusätzlich mit seinem zweiten Kanal den Strom. Die Strommessung selbst erfolgt über ein HALL-Sensor-Modul, das eine analoge Spannung ausgibt, die von dem ADC gemessen wird. Pro Messzyklus können somit 13 Messungen, wenn alle 12 Spannungskanäle genutzt werden, durchgeführt werden.

Der ADC wird im 'Single Conversion Mode' mit einer Abtastrate von $f_{ADC} = 16{,}7\,\frac{\text{Samples}}{\text{s}}$ betrieben. Mit der Formel (4.1) [2]. Dies ergibt eine Wandlungszeit t_{con} von:

$$t_{con} = \frac{2}{f_{ADC}} = 119{,}7\,\text{ms} \tag{4.1}$$

Mit einem Logic Analyzer konnte die Länge einer Messung (siehe Abbildung C.14 und C.15) des externen ADC überprüft werden. Dazu wurde die SPI-Schnittstelle des ADC überwacht und die Wandlungszeit t_{conM} gemessen. Diese ist etwas länger als die erwartete Wandlungszeit t_{con}.

$$t_{conM} = 3\,752\,370\,\mu\text{s} - 3\,630\,110\,\mu\text{s} = 122{,}26\,\text{ms} \tag{4.2}$$

Durch Aufsummieren der Messzeiten von zwölf Spannungsmessungen und der Strommessung ergibt sich eine gesamte Messzeit pro Messvorgang von:

$$t_{\text{Mess12+1}} = 122{,}26\,\text{ms} \cdot (12 + 1) = 1{,}589\,\text{s} \tag{4.3}$$

Würde nur eine Zelle gemessen werden, ergäbe sich eine Gesamtmesszeit pro Messvorgang von:

$$t_{\text{Mess1+1}} = 122{,}26\,\text{ms} \cdot (1 + 1) = 0{,}2445\,\text{s} \tag{4.4}$$

4.4.3. Bestimmung der Lastpfad-Innenwiderstände

Um das Zykliergerät auch mit passiven Lasten wie z. B. Lastwiderständen zu betreiben, wurden die Innenwiderstände der Zykliergeräte ermittelt. Diese setzen sich aus mehreren Einzelwiderständen zusammen:

dem Übergangswiderstand $R_{contact}$,

dem Kabelwiderstand und den Steckverbindungen R_{cable},

dem Sicherungswiderstand R_{fuse} und

dem Relaiswiderstand R_{relay}

Für die Bestimmung des Innenwiderstandes wurde bei angezogenem Lastrelais ein Konstantstrom von 4 A durch die Lastpfade des Zykliergerätes vorgegeben und der Spannungsabfall an den Anschlüssen gemessen. Der Messaufbau ist in Abb. 4.4 und die Ergebnisse in Tabelle 4.8 dargestellt.

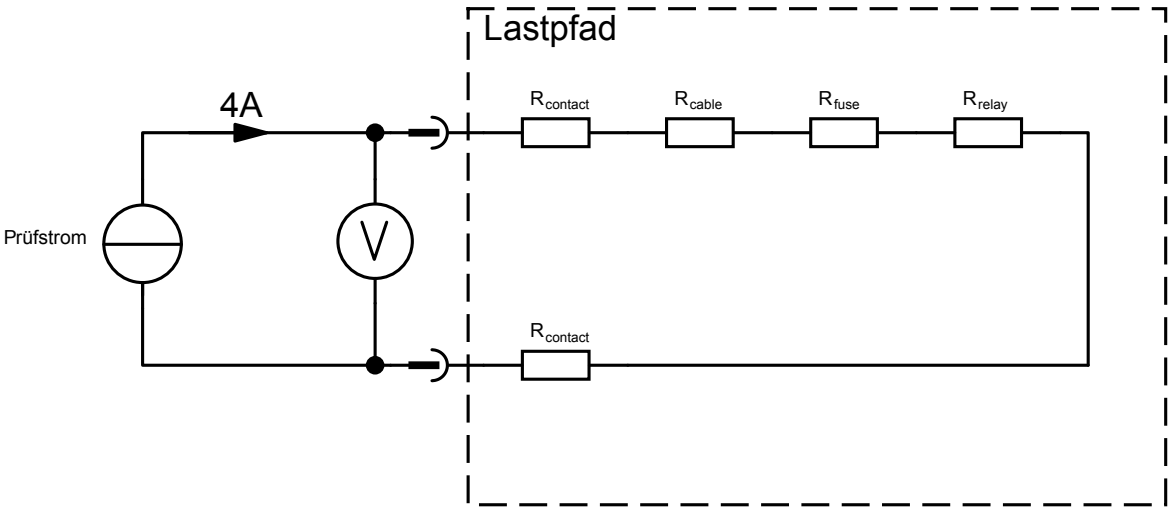


Abbildung 4.4.: Abbildung einer Innenwiderstandsmessung an einem Lastpfad

Tabelle 4.8.: Innenwiderstände der Zykliergeräte

Innenwiderstand des Lastpfad	Zykliergerät R	Zykliergerät W
Relais 1	$\frac{133\,\text{mV}}{4\,\text{A}} = 33{,}25\,\text{m}\Omega$	$\frac{149\,\text{mV}}{4\,\text{A}} = 37{,}42\,\text{m}\Omega$
Relais 2	$\frac{166\,\text{mV}}{4\,\text{A}} = 41{,}50\,\text{m}\Omega$	$\frac{225\,\text{mV}}{4\,\text{A}} = 56{,}32\,\text{m}\Omega$
Relais 3	$\frac{190\,\text{mV}}{4\,\text{A}} = 47{,}50\,\text{m}\Omega$	$\frac{198\,\text{mV}}{4\,\text{A}} = 49{,}41\,\text{m}\Omega$
Relais 4	$\frac{153\,\text{mV}}{4\,\text{A}} = 38{,}20\,\text{m}\Omega$	n. a.

Eine Überschlagsrechnung mit der ECC $LiFePO_4$ Batterie ($R_i = 1{,}2\,\text{m}\Omega$ und $U_N = 3{,}3\,\text{V}$) ergibt für einen Strom von $I_L = 20\,\text{A}$ bereits einen Lastwiderstand von:

$$R_{\text{Last}} = \frac{U_N - I_L \cdot R_i}{I_L} = \frac{3{,}3\,\text{V} - 20\,\text{A} \cdot 1{,}2\,\text{m}\Omega}{20\,\text{A}} = 163{,}8\,\text{m}\Omega \tag{4.5}$$

Bei höheren Strömen nimmt der Widerstandswert weiter ab. Wie aus der Tabelle 4.8 ersichtlich, macht der Innenwiderstand des Zykliergerätes bei einem Strom von $20\,\text{A}$ bereits ca. $25\,\%$ des Lastwiderstandes aus. Daraus folgt, dass der Innenwiderstand des Zykliergerätes bei der Auslegung von passiven Lasten nicht zu vernachlässigen ist.

4.5. Anforderungen an das Zykliergerät

Bei der Inbetriebnahme und dem Testbetrieb zeigten sich Unzuverlässigkeiten im Messablauf, die zu einer umfangreichen Fehlersuche und Fehlerbehandlung führte. Des Weiteren fehlten für die Zyklierung der Buszyklen Zyklus-I und Zyklus-E Funktionen am Zykliergerät, die es zu implementieren galt. Die notwendigen Funktionen werden nachfolgend aufgeführt.

4.5.1. Spannungskalibrierung

Da die Spannungsmessung an den Batterien, wie in 2.1.3 beschrieben, möglichst genau sein muss, ist eine regelmäßige Kalibrierung des externen ADC [2] notwendig. Diese stellte sich allerdings als äußerst aufwendig heraus, da sie nur im Debugging Mode des Mikrocontroller ausgeführt werden kann und wozu ein Computer mit der 'Code Composer Studio' (CCS) Software notwendig ist. Des Weiteren basiert die derzeitige quadratische Kalibrierung auf drei Stützstellen über dem gesamten Spannungsbereich. Eine Erweiterung dieser Funktion und eine Implementierung einer halbautomatischen Kalibrierfunktion in dem Zykliergerät wäre sinnvoll. Wenn sich das Zykliergerät im Idle-Modus (keine Zyklierung erfolgt) befindet soll über einen UART-Befehl die Kalibrierfunktion, über einen Computer (Terminal), aufgerufen werden.

Je Spannungskanal sollen 20 verschiedene Spannungen in einem Messbereich von 0 bis $5\,\text{V}$ an den zu messenden Kanal des Zykliergerätes gelegt werden. Parallel dazu wird mit einem genauen Voltmeter z. B. Tektronix-Messgerät (zur Referenz) gemessen und der gemessene Wert in einem MATLAB-Skript notiert. Das Zykliergerät speichert die Messwerte in digitalisierter Form auf der SD-Karte innerhalb einer Textdatei ab und gibt zusätzlich die Werte über die UART-Schnittstelle auf die Konsole des Terminals aus. Die Anzahl der zu kalibrierenden Kanäle ist über die auf der SD-Karte befindlichen 'config.txt' einzustellen. Das Zykliergerät

soll durch den entstehenden Spannungshub am jeweiligen Kanal erkennen, dass eine neue Spannung am Eingang angelegt oder die Spannung entfernt wurde. Diese Spannungshübe sollen dem Zykliergerät auch als Trigger für eine neue Messwertaufzeichnung dienen. Über ein MATLAB-Skript soll die quadratische Kalibrierung, wie sie von Martin Kusche [26] für drei Stützstellen entworfen wurde, für 20 Stützstellen durchgeführt werden.

4.5.2. Neuauswahl der Temperatursensoren

Bei den zwölf Zell-Temperaturmessungen handelt es sich um einen analogen Spannungsteiler, der aus einem NTC-Widerstand und einem $10\,\mathrm{k}\Omega$ Widerstand besteht und an einem Mikrocontroller Eingang angeschlossen ist. Durch die Toleranzen der Widerstände ist eine Kalibrierung jeder einzelnen Temperaturmessung notwendig.

Bei den zwölf Zell-Temperaturmessungen handelt es sich um analoge NTC Widerstände [1]. In der Kombination mit Widerständen fungieren sie als Spannungsteiler und sind jeweils an einem ADC Eingang des Mikrocontrollers angeschlossen. Die Schwierigkeiten der Kalibrierung der Temperaturmessung ist auf die Toleranzen des NTC-Widerstandes, Spannungsteilerwiderstandes und der Spannungsquelle zurückzuführen. Es müsste wie bei der Spannungskalibrierung jeder Kanal einzeln kalibriert werden. Eine Lösung ohne eine Kalibrierung ist daher vorzuziehen. Eine Temperaturkalibrierung an den Zykliergeräten wurde bis dato nicht durchgeführt.

In dem Projekt BATSEN wurde ein anderer Typ von Temperatursensoren erfolgreich in Verbindung mit ARM-Mikrocontrollern getestet und erprobt. Diese Temperatursensoren vom Typ DS18B20 der Firma Maxim Integrated [28] kommunizieren über einen 1-Wire Bus, der nur aus einer Daten- und einer Masseleitung besteht. Über eine einfache MOSFET-Schaltung lässt sich dieser Sensor an eine UART-Schnittstelle des ARM-Mikrocontrollers anbinden. Da es sich bei der Schnittstelle um einen Bus handelt, ist auch die Messung von mehreren Sensoren über die gleiche Schnittstelle möglich. Dies ist im Vergleich zu den zwölf NTC Spannungsteilern eine große Hardwareersparnis. Da die 1-Wire Temperatursensoren bereits von Haus aus eine Genauigkeit von $0{,}5\,\%$ in einem Temperaturbereich von -10 bis $85\,^{\circ}\mathrm{C}$ besitzen, ist eine weitere Kalibrierung nicht zwingend notwendig.

4.5.3. Vergrößerung der Konfigurations-Datei

Die derzeitige Konfigurationsdatei des Zykliergerätes *config.txt* kann maximal 100 Zyklusschritte (Lastwechsel) beinhalten. Die Einführung von Schleifen und eine Vergrößerung der maximalen Anzahl der Schritte ist notwendig, um komplexere und längere Zyklen simulieren zu können.

4.5.4. Anpassung des Zeitverhaltens der Lastrelais-Steuerung und der Messsteuerung

Durch die Grenzfrequenzbestimmung in Kapitel 3.1.2.1 wurde dargestellt, dass $95{,}2\,\%$ der aufgenommenen Beschleunigungsdaten in einem Bereich von 0 bis $1\,\mathrm{Hz}$ die Busbeschleunigung beschreiben und die restlichen $4{,}8\,\%$ der Messdaten die $>1\,\mathrm{Hz}$ sind vernachlässigt werden können. Um diesen Beschleunigungszyklus auf dem Zykliergerät durch Einhaltung des Nyquist-Shannon-Abtasttheorems exakt zyklieren zu können, müssten die Lastpfade alle $<500\,\mathrm{ms}$ umschaltbar sein.
Da das Zykliergerät durch seine vier Stufen nur sehr grob auflösen kann und bei Einzellenbetrieb die Messzeit der Spannungs- und Strommessung (siehe Abschnitt 4.4.2) bei $0{,}245\,\mathrm{s}$ liegt, wurde entschieden, dass das Zykliergerät nur im Einzellenbetrieb zu betreiben ist. Weiter wurde, zugunsten eines stabilen Betriebes, festgelegt, dass die Strom- und Spannungsmessung höchstens zwei mal pro Sekunde erfolgen soll. Es sollen pro Lastrelaisumschaltung zwei Spannungs- und Stromessungen durchgeführt werden. Somit ist die Lastrelaisumschaltung mit $1\,\mathrm{Hz}$ zu betreiben. Außerdem ist die Ansteuerung der Lastrelais anzupassen (siehe 4.4.1).

5. Prüfstandsanpassung

5.1. Softwareanpassungen für das Zykliergerät

Im Folgenden werden die Softwareänderungen die an dem Zykliergerät durchgeführt wurden beschrieben. Von Beginn der Übernahme an lief das Zykliergerät nur phasenweise stabil. Die Fehlersuche und Softwarekorrektur hat, in dieser Arbeit, einen großen Teil der Bearbeitungszeit ausgemacht. Sie wird im Anhang Dokumentiert. Weiterwird die Implementierung neuer Funktionen und die Anpassung des Zykliergerätes an die Anforderungen aus 4.5 dargestellt. Die Implementierungen wurden vorerst nur an dem Zykliergerät R durchgeführt und nach der Erprobung auch auf dem Zykliergerät W übertragen.

5.1.1. Ergänzungen und Erweiterungen von Kalibrierfunktionen im Zykliergerät

Eine neue Kalibrierfunktion für die Kalibrierung der Spannungsmessung *calibrate_adc_quadratisch()* wurde für das Zykliergerät in der *config.c* programmiert. Diese Funktion kann über den Befehl *calVolt*, der im *config* Menü geführt ist, aufgerufen werden. Dies ermöglicht den Aufruf der Funktion von einem Terminal über die serielle Schnittstelle. Durch den Befehl wird eine globale variable *cal_Flag* gesetzt und die Kalibrierfunktion aus der Main-Loop (siehe Flussdiagramm B.4) aufgerufen. Die Kalibrierfunktion wurde in einem Nassi-Shneiderman-Diagramm siehe B.5 dokumentiert. Durch die beiden programmierten Unterfunktionen *calibrate_adc_quadratisch_waitLowToHigh()* und *calibrate_adc_quadratisch_waitHighToLow()* lässt sich das Anlegen und das Entfernen einer Spannung an dem jeweiligen Kanal detektieren. Die Quadratische Kalibrierung basiert auf der Messung von 20 Spannungswerten, die an den Eingängen, nacheinander anzulegen sind. Anschließend werden die im Zykliergerät ermittelten Werte mit den Vorgabewerten in MATLAB verglichen und die für die für die quadratische Korrektur notwendigen drei Parameter bestimmt. Diese Korrekturparameter werden über die Konfigurationsdatei des Zykliergerätes (*'config.txt'*) wieder an das Zykliergerät übergeben. Zur Darstellung am LC-Display wurden drei neue Ausgaben mit jeweiligem Displayhandler programmiert. Diese werden hier beschrieben.

- *CALLPLUG* ist die Aufforderung an Channel (Ch.: X) die Spannung (V.: Y) anzulegen, wie in der Abb. 5.3 zu sehen.
- *CALLMEAS* sagt aus, dass eine Messung durchgeführt wird, die Kabel sind nicht zu entfernen.
- *CALLPULL* sagt aus, dass die Messung fertig ist und die Kabel gezogen werden sollen. Des Weiteren kann über die Betätigung des rechten Buttons des Zykliergerätes die Messung wiederholt werden. Intern wird im Falle einer Betätigung das *calibrateRedoFlag* gesetzt.

Die Kalibrierung kann durch die Betätigung des linken Buttons die Kalibrierung abgebrochen werden. Durch die Betätigung wird das *calibrateStopFlag* gesetzt.

Ein vollständiger Kalibriervorgang für einen Kanal wird nachfolgend beschrieben.

1. Das Zykliergerät über USB mit einem Computer verbinden und die Verbindung über ein Terminal-Programm wie z. B. HTerm oder putty aufbauen. Dazu sind folgende Einstellungen notwendig:
   - Speed (baud): 115200
   - Data bits: 8
   - Stop bits: 1
   - Parity: None
2. Eine Rohde & Schwarz Spannungsquelle (NGT 35) 0 bis $5\,\mathrm{V}$ und ein Tektronix Multimeter (DM4020) sind mit vier Kabeln so miteinander zu verschalten, dass bei der Messung der Masse-Sammelpunkt und der Plus-Sammelpunkt sich am Zykliergerät befinden. Der Masse-Sammelpunkt kann, wie auf Abb. 5.1 zu sehen schon an dem Zykliergerät angeschlossen werden.

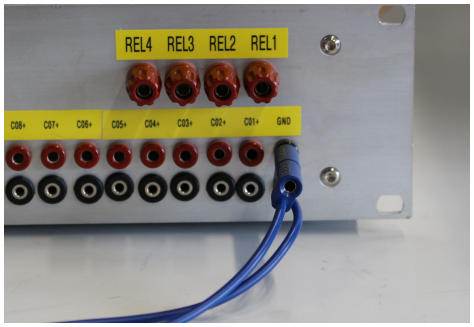


Abbildung 5.1.: Anschluss der Masse am Zykliergerät

3. Das Zykliergerät wird eingeschaltet und über das Terminal wird der Befehl *config* gesendet, um von dem Hauptmenü des Zykliergerätes in das Konfigurationsmenü zu gelangen. Im Konfigurationsmenü wird durch Absenden des Befehls *'calVolt'* die Kalibrierfunktion ausgeführt. Auf dem Display des Zykliergerätes erscheint folgende Ausgabe, wie in Abb. 5.2 dargestellt. Durch das Senden eines *'?'* oder *'help'* an das Zykliergerät wird eine Liste der möglichen Befehle in dem jeweiligen Menü im Terminal angezeigt.

Abbildung 5.2.: Displayhandler *CALLPLUG*

4. An der Spannungsquelle, die mit dem Textronik Multimeter kontrolliert wird, wird die gewünschte Spannung eingestellt und in dem Matlabskript *quadratischeKalibrierung.m* (siehe Anhang D) eingetragen. Es empfiehlt sich das Zykliergerät für den Bereich zu kalibrieren, in dem es später auch benutzt wird, um bei der späteren Messung eine möglichst hohe Genauigkeit zu erzielen. Soll beispielsweise das Zykliergerät in einem Bereich von 2 bis $3{,}9\,\mathrm{V}$ eingesetzt werden, dann sind die 20 Spannungen (siehe Tabelle 5.1) nacheinander vorzugeben. Zu beginnen ist mit dem kleinsten Spannungswert. Des Weiteren ist zu beachten, dass die Kalibrierung nur in dem Bereich $0{,}2$ bis $5\,\mathrm{V}$ einzusetzen ist. Diese Werte ergeben sich aus der maximalen Eingangsspannung $5\,\mathrm{V}$ des ADC und der Schaltschwelle der Kalibrierfunktion von $0{,}2\,\mathrm{V}$ um das Anlegen einer Spannung zu detektieren.

Tabelle 5.1.: Beispiel eines Spannungsbereis für die quadratische Kalibrierung

Messung	Spannung in V
1	2,0
2	2,1
3	2,2
4	2,3
5	2,4
6	2,5
7	2,6
8	2,7
9	2,8
10	2,9
11	3,0
12	3,1
13	3,2
14	3,3
15	3,4
16	3,5
17	3,6
18	3,7
19	3,8
20	3,9

5. Als nächster Schritt wird das positive Spannungs-Kabel an den zu kalibrierenden Kanal (hier Kanal 1) des Zykliergerätes angeschlossen, wie in Abb. 5.3 zu sehen.

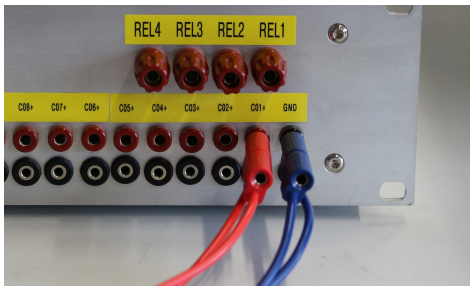


Abbildung 5.3.: Anschluss des positiven Spannungszweigs am Zykliergerät

Durch das einstecken des Steckers in die Buchse wird die Messung gestartet. Das Display springt um und zeigt den Bildschirm (siehe Abb. 5.4). Der Stecker bleibt solange stecken bis die Anzeige measurment WAIT erlischt.

Abbildung 5.4.: Displayhandler *CALLMEAS*

6. Nach dem das Zykliergerät die Messung abgeschlossen hat, springt das Display um und zeigt an, dass die Kabel vom Zykliergerät gezogen werden können (siehe Abb. 5.5). In diesem Zustand kann die Messung auch durch die Betätigung des rechten Buttons wiederholt werden. Bei der Kalibrierung von nur einem Spannungskanal kann die Masse am Zykliergerät stecken gelassen werden. Werden mehrere Kanäle kalibriert muss die Masse um eine Buchse nach links gesteckt werden.

Abbildung 5.5.: Displayhandler*CALLPULL*

7. Die Schritte 4 - 7 werden nun solange wiederholt, bis alle 20 Messungen durchgeführt wurden. Nach der 20. Messung springt das Zykliergerät nach Abziehen der Kabel vom *config*-Menü zurück in das Hauptmenü und das LC-Display zurück auf den Startbildschirm. Auf der SD-Karte werden die digitalisierten Messwerte für den Kanal 1 in der *adc.txt* gespeichert, wie im Anhang B.3 zu sehen.

8. Durch Einlesen der *adc.txt* in das MATLAB-Skript *QuadratischeKalibrierung.m* wird die Berechnung der Korrekturkoeffizienten ausgeführt. Die Daten werden wie in Abb. 5.6 dargestellt ausgegeben. Diese Ausgabe ist in die *config.txt* zu kopieren, wie es in dem Beispiel einer Config-Datei B.3 zu sehen ist.Beim nächsten Start des Zykliergerätes mit der veränderten *config.txt* ist die Kalibrierung abgeschlossen. Die Auswertung einer Kalibrierung ist in Kapitel 6.1 dargestellt.

```
Export     :   3.278074e+06,    7.814156e+07,    2.686942e+07
Hex        :         3204FA,         4A85875,         199FEA9


//Datum: 29-May-2015
coefficient A 01 3204FA
coefficient B 01 4A85875
coefficient C 01 199FEA9
```

Abbildung 5.6.: Die Kalibrierkoeffizienten in Hex-Format für die *config.txt* des Zykliergerätes. Erstellt durch das MATLAB-Skript *QuadratischeKalibrierung.m* (siehe AnhangD).

Sowohl das Zykliergerät R als auch das Zykliergerät W besitzt ihre eigene *QuadratischeKalibrierung.m* D, da die Kalibrierung vom jeweiligen Zykliergerät und dem Setup abhängig ist. Mit Setup ist die externe Beschaltung gemeint, wie z. B. Spannungsquellen und Lasten. Bei der Erprobung der Geräte stellte sich heraus, dass unterschiedliche Setups einen Einfluss auf die Spannungsmessung nehmen. Diese lassen sich durch eine Kalibrierung ausgleichen. Bei jeder Setup-Änderung wurde in dieser Arbeit eine neue Kalibrierung durchgeführt. Dokumentiert sind die Kalibrier-Koeffizienten in der jeweiligen *QuadratischeKalibrierung.m*, die im Anhang D abgebildet ist.

5.1.2. Implementierung der 1-Wire Temperaturmessung

Für die Erneuerung der Temperaturmessung, durch den Einsatz eines 1-Wire Messsensors, in das Zykliergerät konnte ein Teil des Sourcecode (die *onewire.c*) aus der Arbeit von Felix Groth [15] entnommen und angepasst werden. In der bereits bestehenden *temperature.c* des Zykliergerätes mussten nur Anpassungen durchgeführt werden. Die GPIO-Pins [38] PD5 und PD6 wurden so umprogrammiert, dass die neue 1-Wire Kommunikation TxD und RxD über eine UART-Schnittstelle laufen kann. Die Veränderungen der Hardware sind in Abschnitt 5.2.2 erläutert.

5.1.3. Erhöhung der Taktrate für die Lastrelais-Umschaltung sowie der Strom- und Spannungsmessung

Da die *Timer2IntHandler(void)* zuvor nur im Minutentakt aufgerufen wurde und somit auch nur mit Minuten arbeiten konnte, wie in Abschnitt 4.5.4 beschrieben, musste die gesamte Funktion auf Sekunden umgestellt werden. Für die Implementierung der schnelleren $1\,\mathrm{s}$ Umschaltzeiten der Lastrelais wurde in der *clocktimer.c* der *TIMER2* so angepasst, dass

der Interrupt Handler *Timer2IntHandler(void)* für die Relaisumschaltung jede 1 s aufgerufen wird.

Die Umstellung des Messintervalls der Strom- und Spannungsmessung auf 500 ms erfolgt über den *TIMER0*, der regelmäßig, in Abhängigkeit von der Messfrequenz, den *Timer0IntHandler(void)* aufruft. Da die Messfrequenz über die *config.txt* mit dem Befehl *'cylestep'* eingestellt wird, wurde die *config.c* so angepasst, dass eine *'cyclestep'* Eingabe durch die Konfigurationsdatei *config.txt* in einem Bereich von 500 bis 60 000 ms verarbeitet werden kann.

5.1.4. Implementierung von Loops und Erweiterung der Konfigurationsdatei

Für die Implementierung von Loops in der *config.txt* wurden zu den fünf schon bekannten Befehlen wie sie in der Anleitung von Johannes Röhn und Martin Kusche [35] beschrieben sind noch zwei weitere Befehle programmiert worden. Dabei wurde darauf geachtet, dass der Befehlsaufbau den der bereits vorhandenen fünf Befehlen entsprach. Zur Übersicht sind die fünf Befehle nachfolgend kurz beschrieben. Mit X bzw. Y sind die jeweiligen Übergabeparameter des Befehls gemeint.

- *WAIT X Y* ist ein Warte Befehl und gibt die Wartezeit X in Sekunden und das für die Ueit geschaltete Lastrelais Y an.
- *CHARGE_CELLV X Y* ist ein Befehl für die Ladung der Zellen über das Relais Y bis zum Erreichen der maximalen Zellspannung X.
- *DISCHARGE_CELLV X Y* ist ein Befehl für die Entladung der Zellen über das Relais Y bis zum Erreichen der minimalen Zellspannung X.
- *CHARGE_BATV X Y* ist ein Befehl für die Ladung der Batterie über das Relais Y bis zum Erreichen der maximalen Batteriespannung X.
- *DISCHARGE_BATV X Y* ist ein Befehl für die Entladung der Batterie über das Relais Y bis zum Erreichen der minimalen Batteriespannung X.

Die Werte wurden nacheinander abgearbeitet. Um bestimmte Warte-, Lade- bzw. Entladevorgänge mehrfach hintereinander auszuführen, wurden ein Schleifen Befehl erstellt, die eine Verschachtelungstiefe von bis zu drei Ebenen ermöglicht.

- *LOOP_START X Y* definiert den Beginn einer Schleife. Mit X wird die Anzahl der Durchläufe angegeben. Der Parameter wird nicht benötigt und daher auf 0 gesetzt.

- *LOOP_END X Y* markiert das Ende einer Schleife. Beide Übergabeparameter X und Y werden auf 0 gesetzt, da diese nicht benötigt werden. Die Verwendung eines *LOOP_START* Befehls impliziert die Verwendung eines *LOOP_END* Befehls.

In B.3 ist ein Beispiel einer *config.txt* abgebildet, bei der eine Verschachtelungstiefe von drei genutzt wurde. Durch die Anpassung des Buffers, der für die Speicherung der Konfigurationsbefehle im Zykliergerät zuständig ist, wurde die Anzahl der Maximalen Befehle von 100 auf 150 erweitert. Für eine weitere Erweiterung ist ein größerer konzeptioneller Umbau der Software des Zykliergerätes nötig.

5.1.5. Die Anpassungen für den Externen ADC

Wie nachfolgend unter Abschnitt 5.2.1 beschrieben, auf den beiden Zykliergeräten zwei unterschiedliche ADC eingesetzt werden, wird durch eine bedingte Kompilierung mit einer Präprozessor #ifndef-Direktive in der *config.h* zwischen den beiden ADC gewählt. Dazu muss der Programmierer vor der Kompilierung der Software, den jeweiligen ADC auswählen. Die bedingte Kompilierung ermöglicht es, trotz unterschiedlicher Hardware der Zykliergeräte, mit den selben Sourcecode zu arbeiten. In der *myadc.c* wurde über #if-Direktiven zwischen den beiden ADC's unterschieden. Dies war an den Stellen nötig an denen beim Schreiben oder beim Lesen zwischen einem $16\,\mathrm{bit}$ Register für den AD7798 bzw. einem $24\,\mathrm{bit}$ Register für den AD7799 unterschieden werden musste. Die im Zykliergerät verwendete Auflösung wurde auf $16\,\mathrm{bit}$ beschränkt. Des Weiteren wurde die *myadc.c* und *myadc.h* neu strukturiert.

5.2. Hardwareanpassungen im Zykliergerät

Zusätzlich zu den Softwareanpassungen wurde auch die Hardware der Zykliergeräte angepasst, ergänzt und erneuert.

5.2.1. Neuaufbau des Zykliergerätes

An einem Punkt der Fehlersuche (wie im Anhang B.6 dargestellt) am Zykliergerät R wurde entschieden, dass ein Neuaufbau der Hauptplatine des Zykliergerätes und ein Rückbau der von Martin Kusche [26] geänderten ADC-Schaltung notwendig ist. Für den Neuaufbau wurde die von Matthias Schneider modifizierte Platine V2.1 verwendet.

Beim Aufbau der Platine V2.1 wurden nicht alle Baugruppen des Zykliergerät verwendet:

- Nur die Kanäle eins bis sechs der Spannungsmessung und Temperaturmessung wurden bestückt, da das Zykliergerät wie unter 4.5.4 beschrieben nur im Einzellen Betrieb verwendet werden soll. Eine gegebenenfalls notwendige Erweiterung auf zwölf Zellen ist möglich.
- Es wurde der baugleiche aber hochauflösende ADC desselben Herstellers [2] verwendet. Der AD7799 ist ein $24\,\mathrm{bit}$ und der AD7798 ein $16\,\mathrm{bit}$ ADC. Die zusätlichen $8\,\mathrm{bit}$ des AD7799 werden in dem Zykliergerät nicht genutzt 5.1.5.

Des Weiteren wurde die gesamte Last und Messverkabelung überprüft und neu erstellt. Dies war notwendig, um Übergangswiderständen und Wackelkontakte vorzubeugen, da die Kabel-Crimpungen bzw. Lötungen nicht zuverlässig waren. und diese Fehlerquelle sich bei der Fehlersuche häufig als Ursache herausgestellt hat. Es wurde bei der Verlegung des Kabelbaums darauf geachtet, das Leistungs- und Messverkabelung getrennt voneinander verlegt wurden. Es zeigte sich, dass durch das Schalten des großen $500\,\mathrm{A}$ Lastrelais, die Mess- und Anzeigeelektronik beeinflusst wurde. Dies äußert sich durch ein Flackern im Anzeigedisplay und in Messausfällen.

5.2.2. Ergänzung einer Schaltung für die 1-Wire-Temperaturmessung

Für die Temperaturmessung wurde eine Aufsteckplatine entwickelt, die auf die Buchsenleiste (siehe Abb. 5.7 und Abb. C.24) der analogen NTC Temperaturmessung passt. Auf dieser Platine wurde eine MOSFET-Schaltung implementiert, die über einen weiteren Stecker mit einer Spannung von $3{,}3\,\mathrm{V}$ versorgt wird. Diese MOSFET-Schaltung ermöglicht es, den 1-Wire Bus über eine UART-Schnittstelle zu betreiben. Um dies zu bewerkstelligen, wird zusätzlich zu der TxD und RxD Pins noch ein weiterer Ausgang des Mikrocontrollers benötigt, der einen MOSFET, der als Pull-Up dient, steuert. Dieser Pull-Up-MOSFET ist notwendig, um die Sensoren mit Spannung zu versorgen. Dadurch, dass der 1-Wire Bus ein zweiadriger Bus ist, wird die Datenleitung sowohl für die Spannungsversorgung der Sensoren als auch für die Datenübermittlung genutzt. Der Schaltplan ist in Abb. 5.8 zu sehen.

Auf der Hauptplatine des Zykliergerätes wurden die Widerstände und die Kondensatoren, die ein Bestandteil des analogen NTC-Spannungsteilers, entfernt.

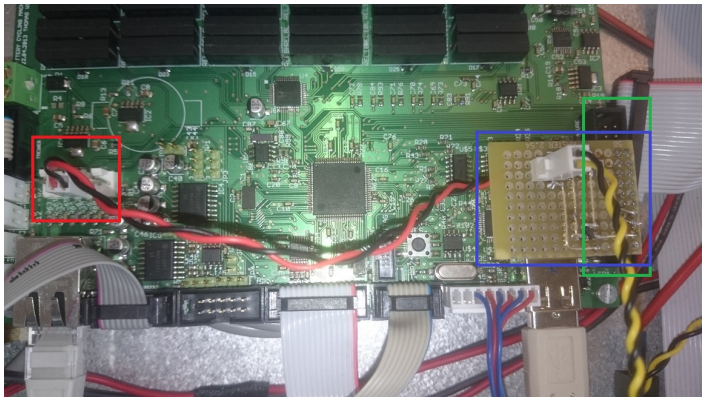


Abbildung 5.7.: Hier ist die 1-Wire Platine in blau, der Abgriff der 3,3 V Spannungsversorgung Haupt-Platine in rot und die Buchsenleiste für die Kontaktierung (in Abb. C.25 vergrößert) in grün gezeigt.

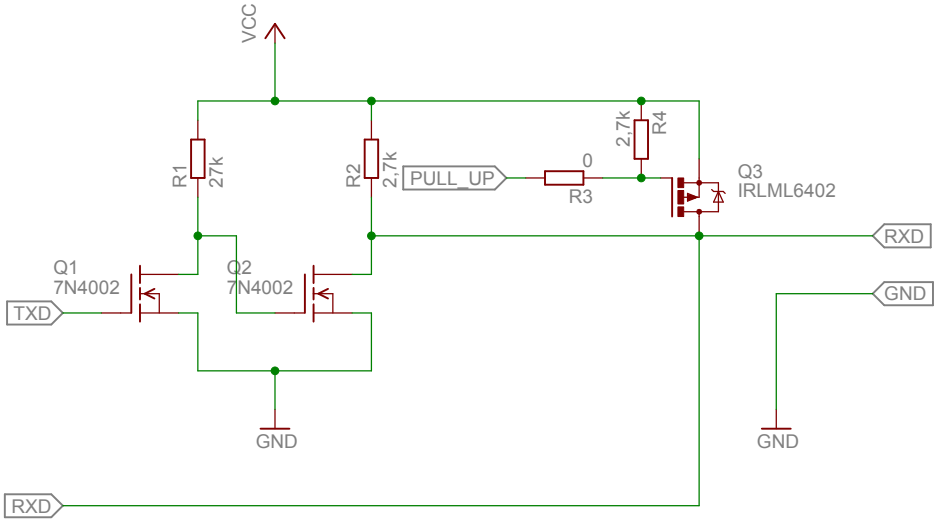


Abbildung 5.8.: Schaltplan der Mosfet-Schaltung [27]

Die 1-Wire Temperatursensoren, wurden nach der Implementierung getestet. Sie wurden in dem Temperaturschrank 4.2.1 bei verschiedenen Temperaturen als auch einem Test bei negative Temperaturen wie in Abb. 5.9 zu sehen, getestet und überprüft. Die Implementierung der 1-Wire Temperaturmessung fand sowohl an dem Zykliergerät R als auch an dem Zykliergerät W statt.

Abbildung 5.9.: Test für den negativen Temperaturbereich der 1-Wire Temperatursensoren

5.3. Mechanischer Aufbau des Prüfstandes

Der Aufbau des Zykliergerätes ist unter 5.3.1 beschrieben. Für den sicheren Transport der $LiFePO_4$ Batterien wurden Batterieboxen für die Zellen gebaut, die unter 5.3.2 beschrieben werden.

5.3.1. Beschreibung des Zyklierprüfstandes

Beide Zykliergeräte wurden in ein $19''$ Rack eingebaut, welches neben dem Temperaturschrank 4.2.1 gestellt wurde. Zusätzlich zu den Zykliergeräten wurden 4 Spannungsquellen und 2 elektronische Lasten in das Rack eingebaut. Ein Computer diente mit dem Programm HTerm als Terminal. Der Computer kann über eine eingerichtete Remotedesktopverbindung von anderen Computern ferngesteuert werden. Dies ermöglicht auch die Fernbedienung der Zykliergeräte. Eine Fernüberwachung des gesamten Systems war durch die angeschlossene Webcam möglich.

Für die Zyklierung der Batterien über die Zykliergeräte gibt es drei verschiedene Setups. Setups eins und zwei sind sich vom Aufbau ähnlich. Diese Setups ermöglichten den parallelen Betrieb der beiden Zykliergeräte mit je zwei Spannungsquellen und einer elektronischen Last. Diese Setups ermöglichen die Aufnahme von Batteriekennlinien bei Konstantstrombelastungen. Mit dem dritten Setup werden die Elektro-Busprofile simuliert. Für den Test der Buszyklen wurden die Anschlüsse des Zykliergerätes so verwendet, dass die Lastpfade eins und zwei für Entladung, drei und vier für die Ladung der Batterien genutzt wurden. Mit einer zweiten elektronischen Last können die zu testenden $LiFePO_4$ Batterien mit konstanten

Strömen von 0 bis $25\,\mathrm{A}$ belastet werden. Durch den Einsatz der großen Spannungsquellen sind konstante Ladeströme von 0 bis $20\,\mathrm{A}$ möglich.

Für die Zyklierung der Batterien mit Strömen $>16\,\mathrm{A}$ wurden Kabel benötigt, die für solch große Ströme ausgelegt sind. Die zuletzt verwendeten Laborleitungen ($1\,\mathrm{mm}^2$ mit Bananenstecker) sind für einen maximalen Strom von $16\,\mathrm{A}$ ausgelegt. Für Ströme $>16\,\mathrm{A}$ werden Kabel mit einem Querschnitt von $16\,\mathrm{mm}^2$ und Ringschuhe für M8-Gewinde (Batterieanschluss) und Steckschuhe für M6 Gewinde (Spannungsquellen und Lastanschluss) anstelle der normalen Laborleitung angefertigt und verwendet.

5.3.2. Bau und Anpassung der Batterieboxen

Damit die $LiFePO_4$ Zellen nicht lose in den Temperaturschrank gelegt werden, hat es sich im BATSEN-Projekt etabliert, Zellkoffer für die Zellen zu verwenden. In einen Zellkoffer passen je nach Größe der Batterien und Größe der Metallkoffer zwischen zwei bis vier Zellen.

In der Arbeit von Martin Kusche wurde ein Koffer für vier LiNANO-Zellen [22] gebaut. Jede Zelle lässt sich über zwei Bananenbuchsen einzeln anschließen. Dieser Koffer wurde zu Beginn dieser Arbeit ersetzt, da durch Defekte an den Befestigungsmechanismen, die Isolierung einzelner Zellen beschädigt wurden. Der Isolationswiderstand war so gering, dass sich die Zellen über das Gehäuse entladen haben. Bei dem Neuaufbau des Koffers, wurde von jeder Zelle die Isolation erneuert und es wurde darauf geachtet, dass die Befestigungsmechanismen ausreichend Abstand zu ihren Nachbarzellen haben. In Abb. 5.10 ist der neu aufgebaute Zellkoffer für die LiNANO-Zellen abgebildet.

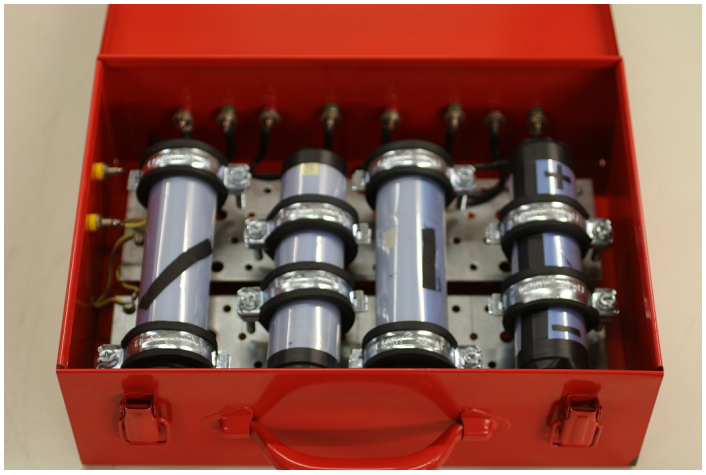


Abbildung 5.10.: Neu aufgebauter Zellkoffer für die LiNANO-Zellen

Für die zwei größeren ECC-Zellen [14] wurde ein neuer Koffer (siehe Abb. 5.11) gebaut. Dieser zeichnet sich durch die großen isolierten M8-Anschlussgewinde, aus. Durch die großen Anschlüsse lassen sich Kabel mit einem Querschnitt von bis zu $16\,\mathrm{mm}^2$ an die Zellen anschließen. Des Weiten wurde in den Koffer direkt je Zelle ein 1-Wire Temperatursensor eingebaut. Dieser wurde auf die Batterie geklebt.

Abbildung 5.11.: Neuer Zellkoffer für die ECC-Zellen

6. Probebetrieb des Prüfstandes und Durchführungen von Batterieuntersuchungen

Bei den Batterieuntersuchungen kamen die unter 2.1.4 dargestellten Batterien zum Einsatz. Durch die Begrenzung der Lade- und Entladeströme durch die verwendeten Spannungs- bzw. Stromquellen und der elektronischen Lasten, konnten nicht alle Batterien bis an ihre Grenzen belastet und getestet werden. Dies betraf vor allem die großen ECC-Batteriezellen, da die maximal mögliche $20\,\mathrm{A}$ Ladung bzw. $25\,\mathrm{A}$ Entladung nur $0{,}5\,\mathrm{C}$ der $42\,\mathrm{Ah}$ Zelle ausmacht. Die ECC-Zelle ist durch ihre Kapazität, im Vergleich zu den anderen beiden Zellen die Zelle, die den typischen Elektrobuszellen am nächsten kommt. Bei der EDDA-Bus Zelle handelt es beispielsweise um eine $46\,\mathrm{Ah}$ Zellen, wie in 2.2.2 dargestellt.

Bei der Untersuchung der Batterieeigenschaften wurde versucht, die Ruhespannungskennlinie der verschiedenen Zellen aufzunehmen. Hierzu wurde die Zelle stückweise entladen bzw. geladen und nach jeder Ladung für mehrere Stunden ruhen gelassen, damit sich die Ruhespannung einstellt. Bei den kleineren A123-Zellen wurde eine Ruhezeit von $4\,\mathrm{h}$ und bei den größeren ECC- und LiNANO-Zellen von $6\,\mathrm{h}$ angenommen.

Bevor die erstellten Buszyklen getestet wurden, wurden die Batterien mit unterschiedlichen Konstantströmen geladen und entladen. Dies diente der Untersuchung der Batterieeigenschaften und deren Verhalten bei kleinen und großen Strömen. In dem Abschnitt 6.2 wird auf die Untersuchungen der einzelnen Zellen näher eingegangen.

Des Weiteren dienten diese Untersuchungen auch dem Probebetrieb des Prüfstandes und den neuen Funktionen der Zykliergeräte. Die Funktion der Spannungskalibrierung vereinfachte den Betrieb, da bei jedem Umbau des Setups die Zykliergeräte neu kalibriert werden mussten, um weiterhin genaue Messdaten zu erfassen. In Abschnitt 6.1 ist die Auswertung einer Kalibrierung der Zykliergerätes R beschrieben.

In einem Probebetrieb wurde zuerst der Zyklus-I (siehe Kapitel 3.2.1) getestet. Hierzu wurde zwischen verschiedenen Strömen, verschiedene Zyklenlängen und dem Einbinden von Zwischenladungen variiert. In dieser Versuchsreihe sollten die Zellen mit möglichst hohen

Strömen betrieben werden. Aus diesem Grund wurde der Zyklus auch nicht an der ECC-Zelle getestet, da ein Entladen mit $0{,}5\,\mathrm{C}$ nicht bustypisch und somit nicht relevant ist. Der Zyklus wurde an der LiNANO-Zelle mit Strömen bis zu $1{,}5\,\mathrm{C}$ und später auch mit den kleinen A123-Zelle mit Strömen bis zu $10\,\mathrm{C}$ erprobt.

Für den Zyklus-E wurden die Ströme, wie unter 3.2.2 beschrieben, durch die ermittelten C-Raten-Stufen festgelegt. Da die A123-Zelle die einzige Zelle ist, mit denen $2\,\mathrm{C}$ in dem derzeitigen Prüfstand getestet werden können, wurde der Zyklus nur an dieser Zelle getestet. Ergebnisse zu den Zyklierungen der Zelle sind in Abschnitt 6.3 dargestellt.

Alle Messungen und Tests, die in dem Prüfstand durchgeführt wurden, wurden in dem Temperaturschrank bei einer Temperatur von $20\,^{\circ}\mathrm{C}$ durchgeführt.

Die Messungen ergaben, dass vor allem kleinere Ströme nicht exakt mit dem Hall-Effekt-Sensor erfasst werden können. Dies liegt an den großen Messbereichen von Kanal 1 im Bereich -75 bis $75\,\mathrm{A}$ bzw. am 2. Kanal von -500 bis $500\,\mathrm{A}$ und dem Sensitive Error des Hall-Sensors. Dieser wird laut Datenblatt für den ersten Kanal mit $\pm 0{,}5\,\%$ angegeben. Daraus können sich Messabweichungen von $375\,\mathrm{mA}$ ergeben. Des Weiteren wurde beobachtet, dass diese Abweichungen nicht konstant sind und bedingt durch die Stromrichtung voneinander abweichen.

6.1. Kalibrierung des Zykliergerätes

Bevor mit dem Prüfstand Messungen an Zellen durchgeführt werden können, muss das jeweilige Zykliergerät mit dem entsprechenden Setup kalibriert werden. Stellvertretend wird die Kalibrierung des Zykliergerätes R vom 18.03.2015 an dieser Stelle näher beschrieben. Für diese Kalibrierung wurde die unter 5.1.1 entwickelte Kalibrierfunktion in einem Spannungsbereich von 1 bis $4{,}3\,\mathrm{V}$ verwendet. Nach dem Laden der 'adc.txt' in die 'QuadratischeKalibrierungR.m' werden durch das Ausführen des Skripts die drei Koeffizienten durch die MATLAB-Funktion 'polyfit' für den Spannungskanal 1 bestimmt. Dazu werden die ADC-Digits des Zykliergerätes und die von dem Tektronix Voltmeter erfassten Spannungen, wie in Tabelle 6.1 dargestellt, verwendet.

Tabelle 6.1.: Aufgenommende ADC-Digits des Zykliergerätes bei der jeweiligen vom Tektronix (DMM4020) gemessenen Spannung für den 1. Spannungskanal

ADC-Digits	Spannung in V
12509	1.00074
14753	1.17447
16998	1.35004
19241	1.52546
21481	1.70071
23718	1.87570
25936	2.0491
28199	2.2259
30506	2.4068
32597	2.5699
34886	2.7492
37055	2.9193
39349	3.1018
41623	3.2771
43862	3.4524
46078	3.6259
48307	3.8018
50554	3.9778
52825	4.1549
54986	4.3246

Die aus der Kalibrierung (18.03.2015) bestimmten quadratischen Koeffizienten für den Spannungskanal 01 lauten:

- Koeffizient A_{01} 3.952590e-12
- Koeffizient B_{01} 7.801922e-05
- Koeffizient C_{01} 2.306038e-02

Über die quadratische Funktion (6.1) lässt sich durch Einsetzen der ADC-Digits in die variable $x_{\mathrm{ADC_Digit}}$ die jeweilige Spannung (hier für den Kanal 1) U_{ADC} berechnen.

$$U_{\mathrm{ADC}_{01}} = A_{01} \cdot x^2_{\mathrm{ADC_Digit}_{01}} + B_{01} \cdot x_{\mathrm{ADC_Digit}_{01}} + C_{01} \tag{6.1}$$

In der Abb. 6.1 wurden zum Vergleich die bei der Kalibrierung aufgenommenen Spannungen und ADC-Digits und die daraus berechnete und auf den Koeffizienten basierende quadratische Funktion geplottet.

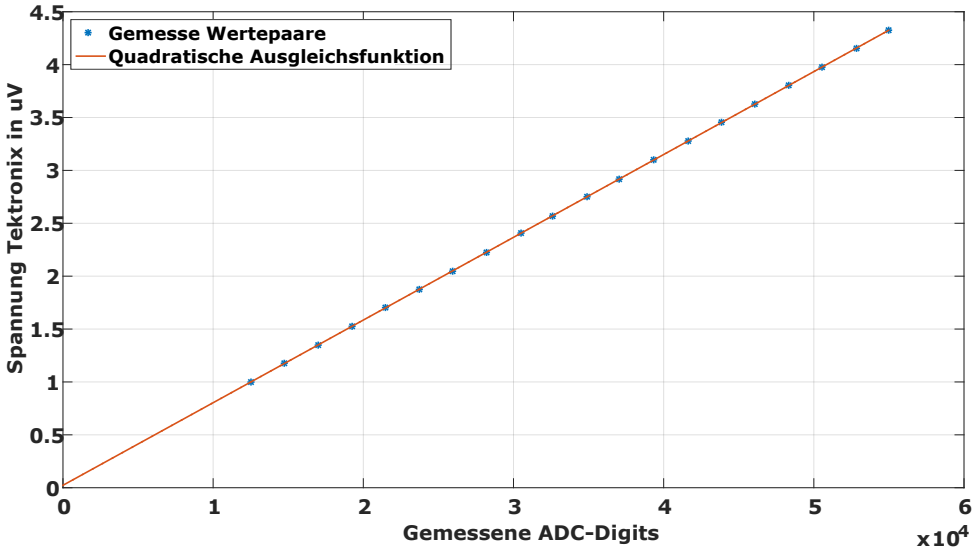


Abbildung 6.1.: Die bei der Kalibrierung mit Tektronix gemessenen Spannungen wurden über die, mit dem Zykliergerät R aufgenommenen, ADC-Digits in blau geplottet. Des Weiteren wurde die, aus den Werten interpolierte, quadratische Funktion in rot über die ADC-Digits geplottet.

Die Koeffizienten wurden nach einer visuellen Überprüfung in die 'config.txt' des Zykliergerätes R kopiert. Das Zykliergerät R wurde anschließend einem weiteren Spannungstest unterzogen. Dazu wurden 21 Spannungen von 0 bis $5\,\mathrm{V}$ in $0{,}25\,\mathrm{V}$ Schritten an das Zykliergerät angelegt und die Spannung des Zykliergerätes mit der Spannungsmessung des Tektronix abgeglichen. Das Ergebnis bzw. die Differenz zwischen den beiden Messungen ist in Abb. 6.2 dargestellt. Deutlich zu erkennen ist, dass in dem kalibrierten Bereich von 1 bis $4{,}3\,\mathrm{V}$ der maximale Messfehler $\pm 1\,\mathrm{mV}$ beträgt. Die entspricht einem relativen Fehler von $\frac{1\,\mathrm{mV}}{1\,\mathrm{V}} = 0{,}1\,\%$. Im Vergleich dazu beträgt die Genauigkeit des Tektronix Voltmeters ca. $0{,}03\,\%$ in Abhängigkeit der Messwerte und des Messbereichs.

Bei jeder Setup Änderung wurde eine Kalibrierung durchgeführt. Dies geschah analog auch mit dem Zykliergerät W, wenn dieses eingesetzt wurde.

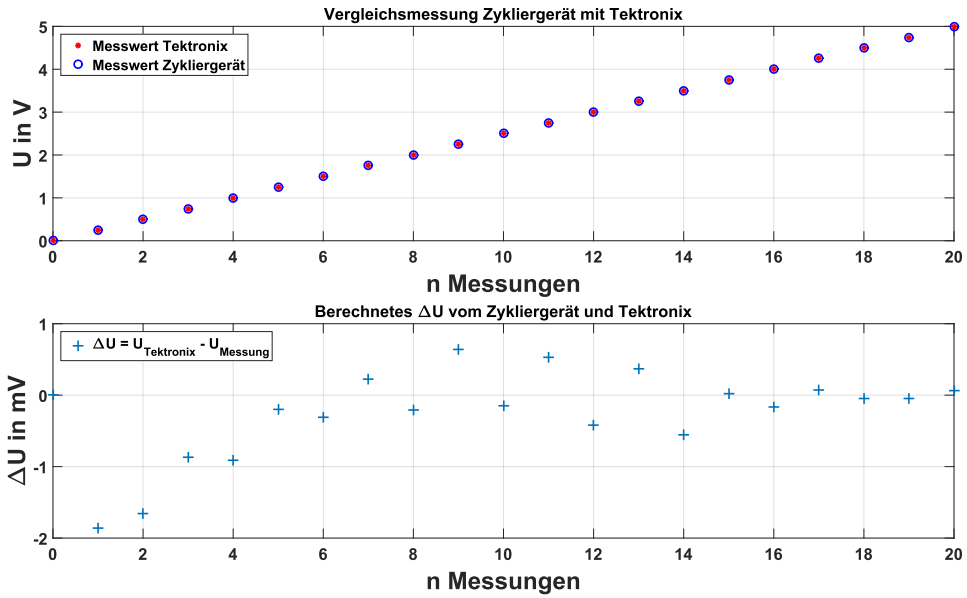


Abbildung 6.2.: Überprüfung der Spannungsmessung nach der Kalibrierung. Oben ist die Spannungsmessung des Tektronix und die des Zykliergerät R im direkten Vergleich, im unteren Plot ist die Differenz zwischen den beiden Messungen dargestellt.

6.2. Untersuchung der Batteriezellen

Für die exakte Bestimmung des Ladezustands einer Batterie ist es wichtig zu wissen, wann eine Lithium-Ionen-Batterie voll geladen ist. Eine geladene Batterie wird als voll angesehen, wenn durch eine Konstantstromladung die Ladeschlussspannung $U_{\text{charge}} = 3{,}7\,\text{V}$ erreicht ist und bei einer weiteren Ladung über eine Konstantspannungsladung, die auf $U_{\text{charge}} = 3{,}7\,\text{V}$ eingestellt ist, bis der Strom gegen $0\,\text{A}$ geht. Dies kommt einer CC-CV Ladung am nächsten. Die Entladeschlussspannung wurde auf $U_{\text{discharge}} = 2{,}8\,\text{V}$ festgelegt. Die Ladung der Batterie wird in den folgenden Abbildungen als Ladungsverlauf bezeichnet und definiert sich wie folgt:

- Ein Zyklus beginnt immer mit einer voll geladenen Batterie.
- Beim Start eines Tests (Zyklisierung) wird die Ladung der Batterie auf $0\,\text{Ah}$ gesetzt und ist somit der Ausgangspunkt des Ladeprofils.
- Im Laufe einer Entladung kann diese bis zum negativen Wert der Nennkapazität sinken.

Bei den Aufnahmen der Ruhekennlinien und der Konstantstromkennlinien wurden das Zykliergerät R und das Zykliergerät W parallel in dem Setup 1 & 2 betrieben.

6.2.1. Untersuchung der LiNANO-Batteriezelle

6.2.1.1. Ermittlung der Ruhespannung

Wie in Abschnitt 2.1.2 beschrieben, soll sich die Ruhespannung nach ca. 4 h einstellen. Eine Überprüfung erfolgte durch eine Messung mit der LiNANO-Zelle. Dazu wurde nach einer zweistündigen Konstantstromladung mit 1 A die Zelle für 6 h ruhen gelassen. Vor der Ladung hatte die Batterie einen ungefähren SOC von 60 %. In der Abb. 6.3 ist die Annäherung der Zellspannung an die Ruhespannung zu sehen. Da die Spannungsmessung verrauscht war, wurde ein Tiefpassfilter[1] eingesetzt. In grün ist der in der Theorie beschriebene Zeitbereich mit 4 h gekennzeichnet. Es zeigt sich, dass sich nach den 4 h eine weitere Veränderung ergibt und die Ruhespannung erst nach 5 h erreicht ist.

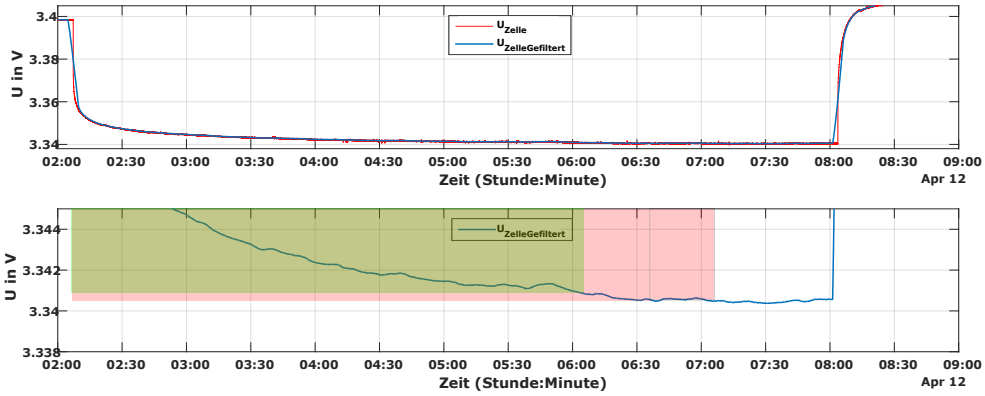


Abbildung 6.3.: Im oberen Plot ist die gemessene Spannung der LiNANO-Zelle, nach einem 2 h Konstantstromladevorgang mit 1 A, abgebildet (blau). In der folgenden 6 h Ruhephase ist die Annäherung der Leerlaufspannung an die Ruhespannung deutlich zu erkennen. Im unteren Plot ist die Annäherung an die Ruhespannung vergrößert (veränderter Spannungsausschnitt) dargestellt.

6.2.1.2. Ruhekennlinie der LiNANO-Batteriezelle

Bei der Aufnahme der Ruhekennlinie Abb. 6.4 wurde die Zelle mit 1 A über 2 h pro Stufe ent- bzw. geladen. Zwischen den Stufen wurde eine sechsstündige Ruhephase eingelegt.

[1] Gleitender Mitttelwert-Filter über ein Fenster von 600 Werten. Dies entspricht einer Fensterbreite von 5 min.

In dieser Ruhekennlinie ist ein leichter Hysterese-Effekt zu erkennen. Deutlich ist in dieser Kennlinie das frühzeitige Erreichen der maximalen Sicherheitsspannung des Zykliergerätes von $U_{MAX} = 3{,}9\,\mathrm{V}$ zu sehen, obwohl die Zelle noch nicht ganz geladen ist. Das Überschreiten der Sicherheitsspannung hatte zur Folge, dass die Zyklierung abgebrochen wurde. Die Ruhespannung sinkt trotz der erreichten $3{,}9\,\mathrm{V}$ sehr tief ab. Dies ist dadurch zu begründen, dass die Zelle, wie in dem grünen Bereich kenntlich gemacht, noch nicht voll geladen ist. Während der Messung gab es noch weitere Probleme. Wie in braun markiert, ist während der Messung die Spannung eingebrochen. Bei der Fehlersuche stellte sich heraus, dass eine Lötstelle an dem Messeingang im Zykliergerät einen Wackelkontakt hatte. Durch mechanische Erschütterungen wurden Spannungseinbrüche verursacht. In dem roten Bereich, kam es sogar zu einem Absturz des Mikrocontrollers im Zykliergerät. Die Messung wurde neu gestartet und die Messdaten aneinander geschnitten.

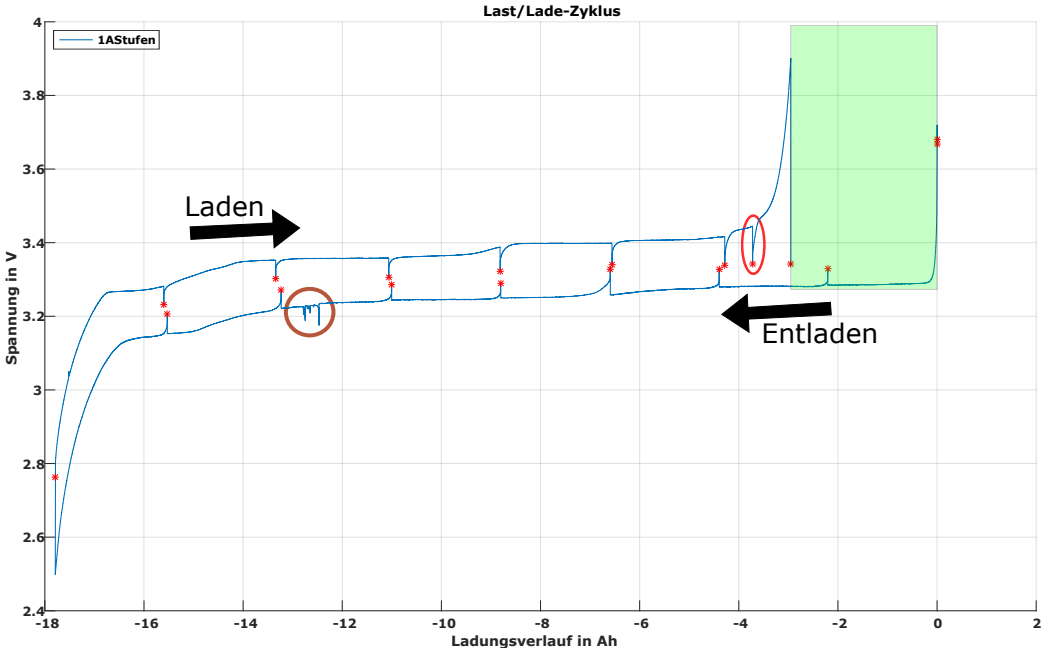


Abbildung 6.4.: Ermittlung der Ruhespannungskennlinie der $15\,\mathrm{Ah}$ LiNANO-Zelle mit einem Konstantstrom von $1\,\mathrm{A}$ über eine Stufenlänge von $2\,\mathrm{h}$. Die roten Sterne geben die erreichte Ruhespannung nach jeweils $6\,\mathrm{h}$ an.

6.2.1.3. Konstantstromkennlinien der LiNANO-Batteriezelle

Aus den vielen Messungen/Zyklierungen mit der LiNANO-Zelle werden für Abb. 6.5 einige Kennlinien ausgesucht. Die aufgenommenen Ruhespannungswerte aus Abschnitt 6.2.1.2 wurden zusätzlich dargestellt. Erkennbar ist, dass sich die Ruhespannungskennlinie in der

Mitte der Konstantstromkennlinien befindet. Außerdem wird deutlich, dass die Kennlinien sehr gut mit den theoretischen Kennlinien in Abb. 2.5 und 2.6 übereinstimmen.
Je höher der Konstantstrom ist, desto größer ist der Spannungsabfall über den Innenwiderstand der Zelle, der sich im Spannungshub, bezogen auf die Ruhespannung, deutlich zeigt. Weiter ist den Kennlinien zu entnehmen, dass bei großen Entladeströmen weniger Ladung und damit auch weniger Energie entnommen werden kann (siehe rote Markierung). Der grau markierte Bereich zeigt die spannungsorierntierte Nachladung der Zelle.

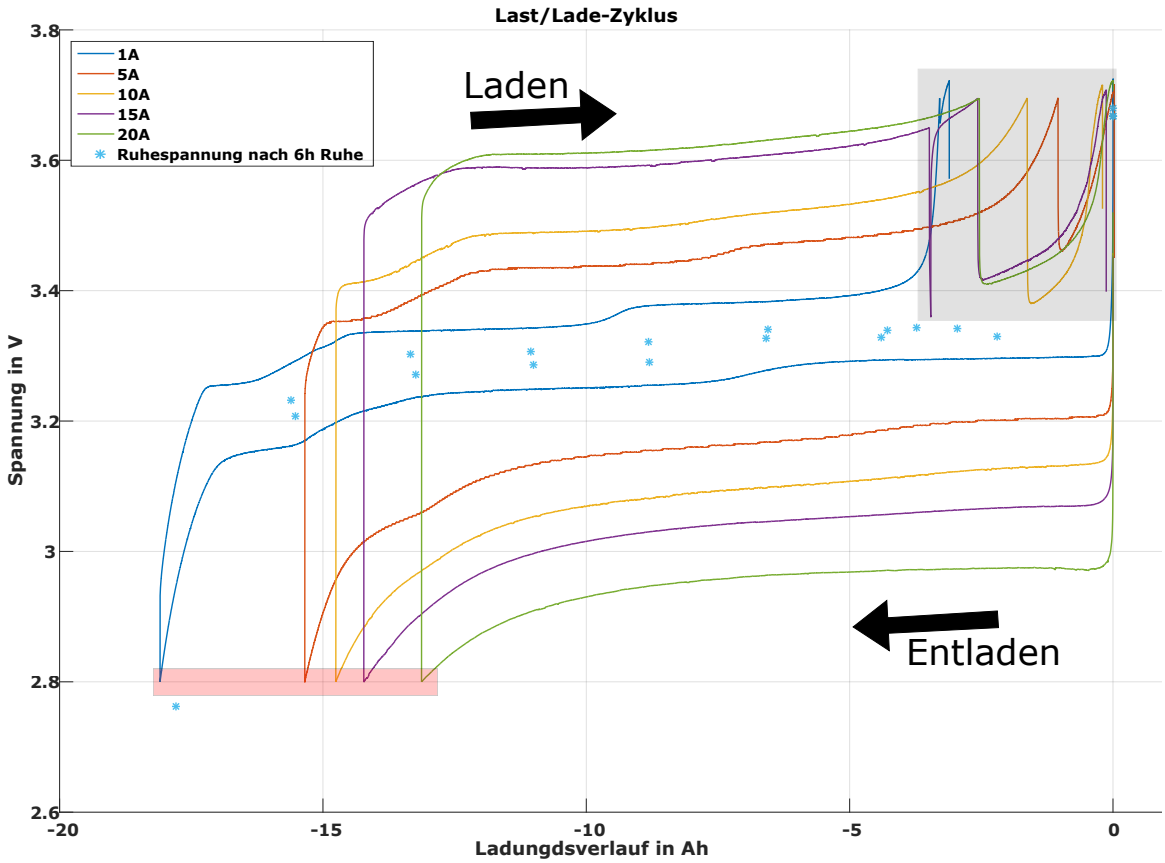


Abbildung 6.5.: Konstantstromkennlinien der 15 Ah LiNANO-Batteriezelle mit verschiedenen Konstantströmen. Im Vergleich dazu wurden die ermittelten Ruhespannungspunkte geplottet.

6.2.2. Untersuchung der A123-Batteriezelle

6.2.2.1. Ruhekennlinie der A123-Batteriezelle

Um die Ruhespannung der A123-Batterie zu ermitteln, wurde die Zelle jeweils für 1 h mit 0,25 A entladen bzw. geladen und danach für 4 h ruhen gelassen bevor eine neue 1 h Entladung bzw. Ladung stattfand. Die Ströme wurden mit einem Tektronix Messgerät zusätzlich gemessen und durch die Strombegrenzung der Spannungsquelle und der Stromregelung der elektronischen Last begrenzt. Dieses schrittweise Entladen und das darauf folgende schrittweise Laden führte zu der blauen Kennlinie in Abb. 6.6. Die sich nach 4 h einstellende Ruhespannungen sind in rot dargestellt.
Zu erkennen ist bei der Kennlinie, dass sich die Ruhespannungen des Entladepfads und die Ruhespannungen des Ladepfads annähern. Dies ist besonderes deutlich in dem fast linearen Bereich der Kennlinie zusehen. Trotz der Annäherung, ist eine leichte Ruhespannungshysterese zu erkennen.

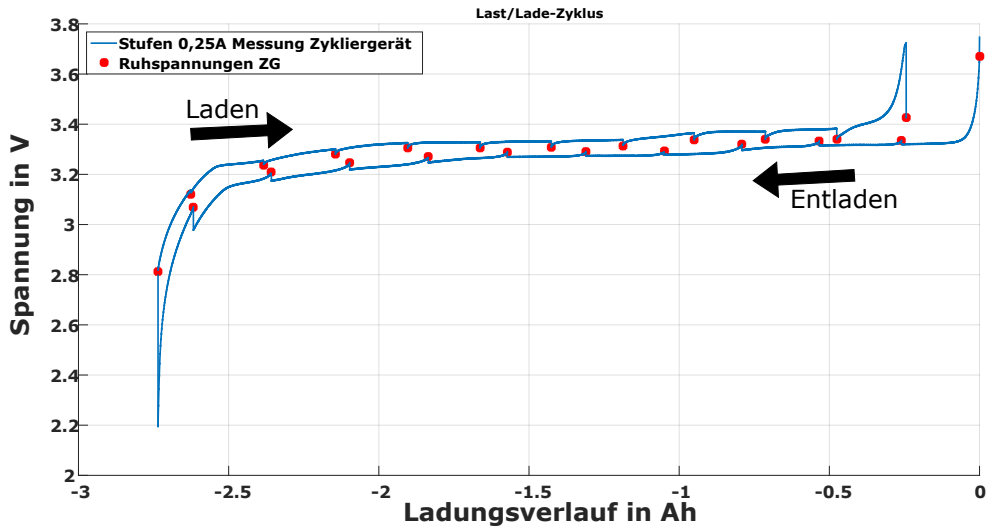


Abbildung 6.6.: Ermittlung der Ruhekennlinie der 15 Ah A123-Zelle mit einer jeweiligen Ruhezeit von 4 h (blau). Die roten Sternchen zeigt die jeweilige erreichte Ruhespannung an.

6.2.2.2. Hochstromtest an der A123-Batteriezelle

Für die Zyklierung der A123-Zelle mit hohen C-Raten wurde das Temperaturverhalten der A123-Zelle bei großen Strömen ermittelt, um in einem Test die Batterie mit höheren C-Raten, als in der Datenblattangabe, zu betreiben. In dem Datenblatt der Zelle wird der maximale

Ladestrom mit 4 C (10 A) und Entladestrom mit 20 C (50 A) angegeben. In dem Test sollen die Zelle mit den maximalen Strömen, die mit dem Prüfstand möglich sind, getestet werden. Dies bedeutet eine Ladung mit 8 C und eine Entladung mit 10 C. Die Zelle wurde gezielt, außerhalb ihrer Kennwerte ge- bzw. entladen. Die Sicherheitsabschaltung des Zykliergerätes wurde auf 35 °C eingestellt. Durch den hohen Entladestrom erreichte die Zelle bereits nach ca. 3 min die Entladeschlussspannung $U_{\text{discharge}} = 2{,}8\,\text{V}$. In der Abb. 6.7 ist deutlich zu sehen, dass ein Temperaturanstieg von 7 °C, der an der Zelle gemessen wurde. Bei der Ladephase wurde die Ladeschlussspannung $U_{\text{charge}} = 3{,}7\,\text{V}$ nach ca. 4 min erreicht. Auch hier ist der Anstieg der Temperatur deutlich zu erkennen.

In Abb. 6.8 ist die Spannung in Abhängigkeit des Ladezustands der Zelle abgebildet, wobei die 0 Ah (grüner Bereich) für eine volle Zelle und der rote Bereich für eine entladene Zelle steht. Eine Entladung bedeutet an dieser Stelle, dass die Entladeschlussspannung erreicht wurde. Beim Erreichen der Ladeschlussspannung wurde die Zyklierung beendet. Der Versatz im grünen Bereich ist durch den Messfehler des Hall-Sensors und zusätzlich noch durch die nicht erfolgte weitere Ladung mit einem geringen Strom zu begründen. Nach diesem Test wurde der Zyklus-I mit ähnlich hohen Strömen mit dieser Zelle erfolgreich erprobt, wie in Kapitel 6.3.1.5 beschrieben.

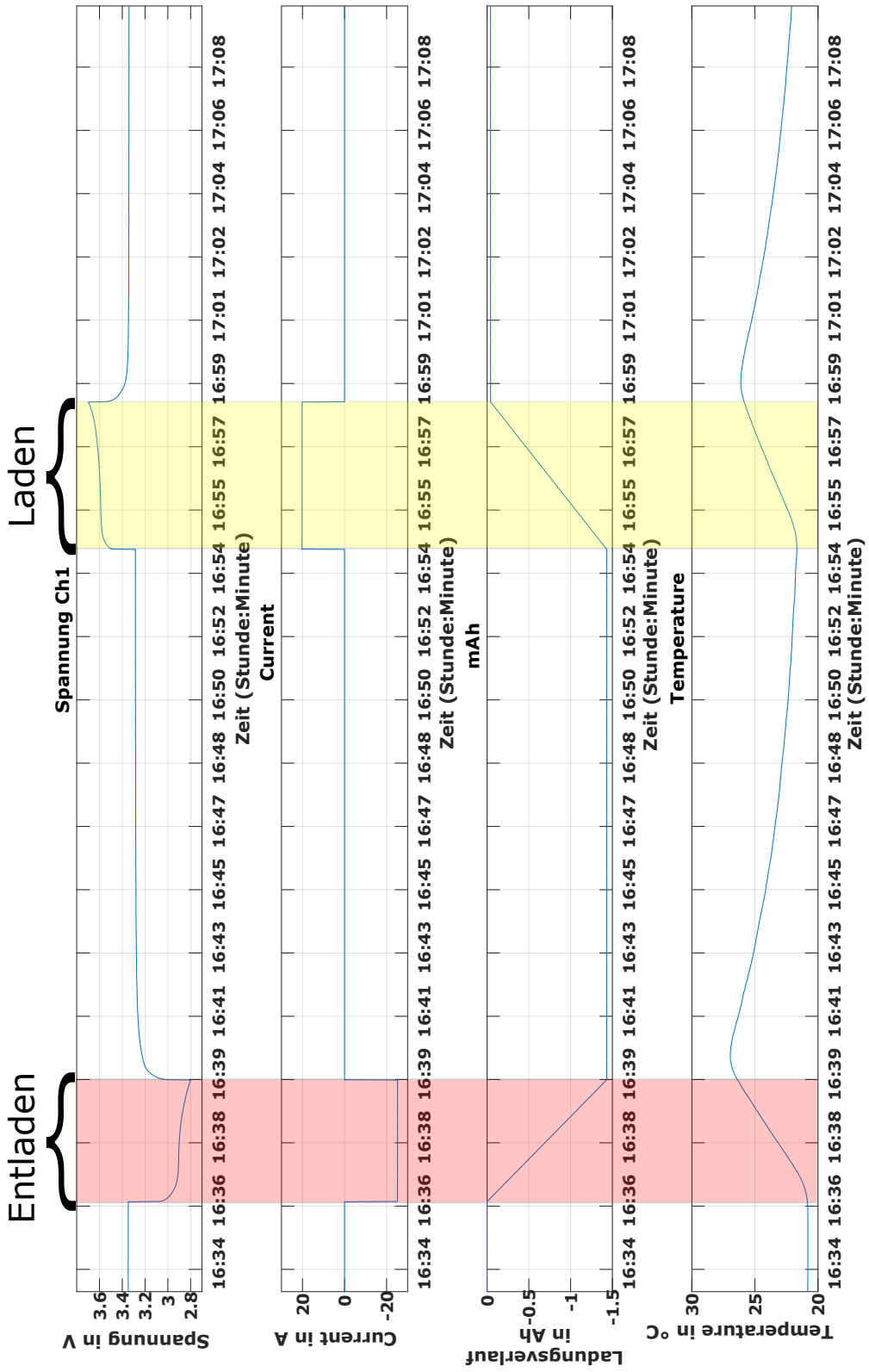


Abbildung 6.7.: Zeitverlauf einer Konstantstrom-Entladung Ladung von 8 C und von 10 C der A123-Zelle.

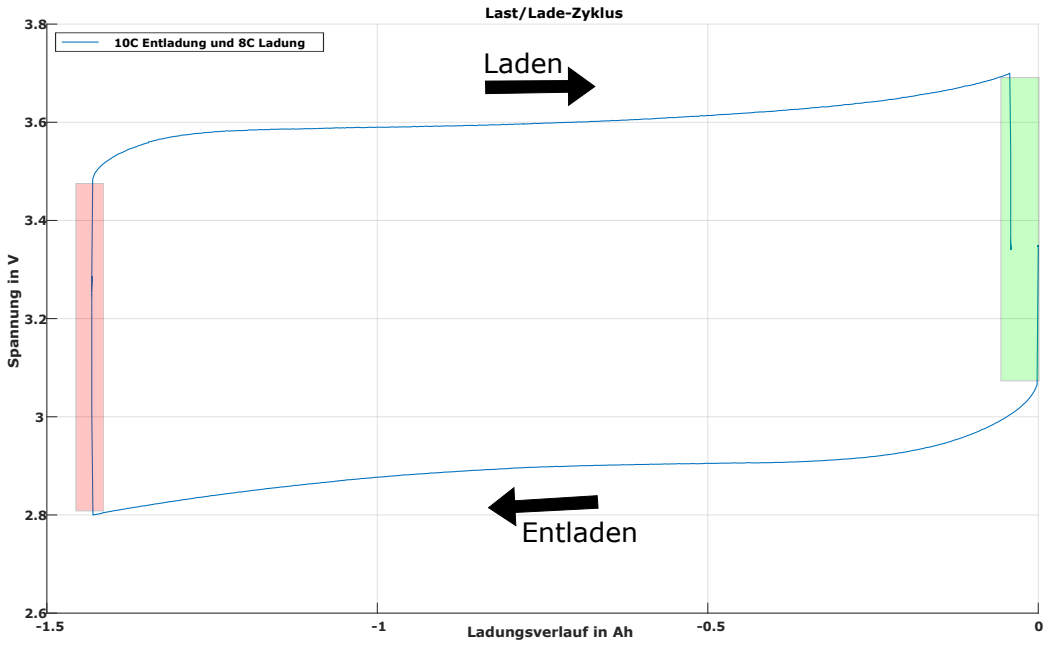


Abbildung 6.8.: Spannung über SOC-Kennline, des unter Abb. 6.7 dargestellten Zeitverlaufes.

6.2.3. Untersuchung der ECC-Batteriezelle

Bei der ECC-Zelle wurde, bedingt durch die Größe der Zelle, die Ruhespannungskennlinie mit nur einer Stützstelle ermittelt, diese ist im Anhang Abb. C.26 dargestellt.
Da der Prüfstand die ECC-Zelle mit maximal $20\,\mathrm{A}$ laden bzw. $25\,\mathrm{A}$ entladen kann und dies ungefähr einem halben C entspricht, dauert ein Entlade/Lade-Zyklus ca. $4\,\mathrm{h}$. Eine hohe Beanspruchung der Zelle mit hohen C-Raten war somit nicht gegeben und es sind daher nur wenige Tests mit der Zelle durchgeführt worden.

6.2.3.1. Konstantstromkennlinien der ECC-Batteriezelle

Ähnlich wie bei der LiNANO-Zelle wurde auch die ECC-Zelle bei verschiedenen Strömen untersucht. In Abb. 6.9 sind die Konstantstromkennlinien der ECC-Zelle zu sehen. Auch hier ist der Spannungshub, der durch den Spannungsabfall am Innenwiderstand der Zelle entsteht, deutlich zu sehen.

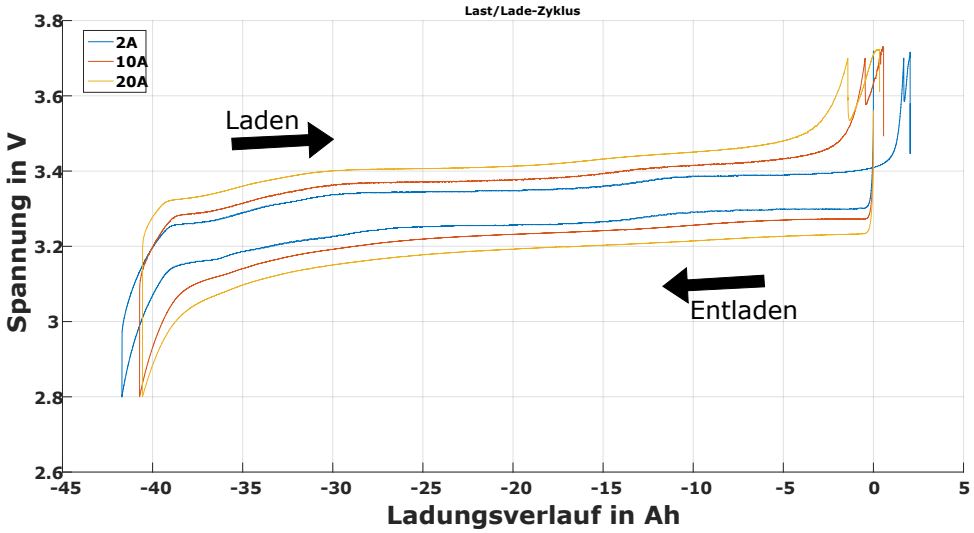


Abbildung 6.9.: Konstantstromkennlinien der 42 Ah ECC-Batteriezelle bei verschiedenen Konstantströmen.

6.2.3.2. Zyklierung der ECC-Batteriezelle im linearen Bereich

Wie in 2.1.3 beschrieben, kann einer bereits voll geladenen Batterie keine weitere Ladung mehr zugefügt werden. Diese Begrenzung kann bei dem Betrieb von Elektrobussen problematisch sein, da es einem vollgeladenen Elektrobus nicht möglich ist, bei einem Bremsvorgang die rekuperierte Energie in die Batterie zu speichern. Daraus folgt, dass der maximale Ladezustand einer Busbatterie, um eine Rekuperation zu ermöglichen, nur einer bestimmte obere Grenze, z. B. 75 % SOC erreichen darf.

In der Abb. 6.10 ist der Verlauf einer Zyklierung, mit einem Konstantstrom von 20 A dargestellt. Eine geladene Zelle wurde auf 75 % SOC entladen und danach achtmal in einem Bereich von 25 % bis 75 % SOC zykliert (roter Bereich). Dies entspricht ungefähr dem linearen Bereich der Zelle. Nach den acht Zyklen wurde die Zelle vollständig bis zum Erreichen der Entladespannung $U_{\text{discharge}} = 2{,}8\,\text{V}$ entladen und anschließend wieder vollständig geladen. Zusätzlich wurden die Ruhespannung bei einem SOC von 25 % bis 75 % mit einer Ruhezeit von 6 h ermittelt und eingezeichnet (grüne Punkte).

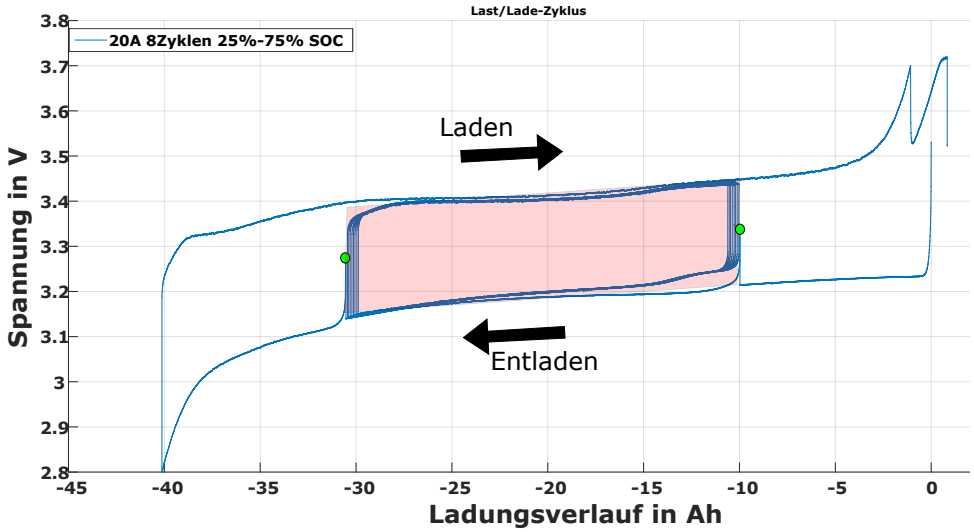


Abbildung 6.10.: Zyklierung der ECC-Batterie mit einem Konstantstrom von 20 A in einem ansatzweise linearen Bereich von 25 % bis 75 % SOC.

6.3. Probebetrieb des Zyklus-I und Zyklus-E auf dem Prüfstand mit den $LiFePO_4$ Batterien

In diesem Abschnitt werden die Buszyklen, die in Kapitel 3 entwickelt wurden, mit der A123- und der LiNANO-Batterie getestet. Von der Vielzahl der durchgeführten Untersuchungen wird nachfolgend ein Auszug der gemessenen Daten dargestellt. Für den Probebetrieb der Buszyklen wurde das dritte Setup, welches unter Kapitel 5.3.1 beschrieben wurde, gewählt. Durch die Durchführung des Probebetriebes im Temperaturschrank, erfolgten die Tests bei einer konstanten Umgebungstemperatur von $20\,^{\circ}\mathrm{C}$.

6.3.1. Probebetrieb des Zyklus-I

6.3.1.1. Einmaliger Durchlauf des Zyklus-I mit der LiNANO-Zelle

Folgende Ströme der Zyklen wurden, wie bereits in der Tabelle 3.4 beschrieben, für den Probebetrieb gewählt:

- ein Entladestrom von $24\,\mathrm{A}$ ($1{,}6\,\mathrm{C}$) über das Relais 1,
- ein Entladestrom von $12\,\mathrm{A}$ ($0{,}93\,\mathrm{C}$)über das Relais 2,
- ein Ladestrom von $8\,\mathrm{A}$ ($0{,}53\,\mathrm{C}$) über das Relais 3 und
- ein Ladestrom von $16\,\mathrm{A}$ ($1{,}06\,\mathrm{C}$) über das Relais 4.

Der Zyklus-I wurde mit der geladenen LiNANO-Zelle durchgeführt und in Abb. 6.11 dargestellt. Deutlich zu sehen ist, dass über den 27-minütigen Verlauf des Batteriezyklus die Zelle um ca. $1\,\mathrm{Ah}$ entladen wurde.

Durch den direkten Vergleich der gemessenen Beschleunigungsdaten aus Kapitel 3.1.2, den daraus ermittelten Stromstufen (Zyklus-I) in Kapitel 3.2.1.2 und den, durch das Zykliergerät, halbsekündlich gemessenen Strömen bei dem Probebetrieb des Zyklus-I in Abb. 6.12 lässt sich zeigen, dass der Elektrobuszyklus erfolgreich auf die Batterie projiziert wurde.

Anhand dieses Zyklus wurden weitere Messungen bzw. Tests durchgeführt. Im Folgenden werden Ergebnisse des mehrfach wiederholten Zyklus-I vorgestellt.

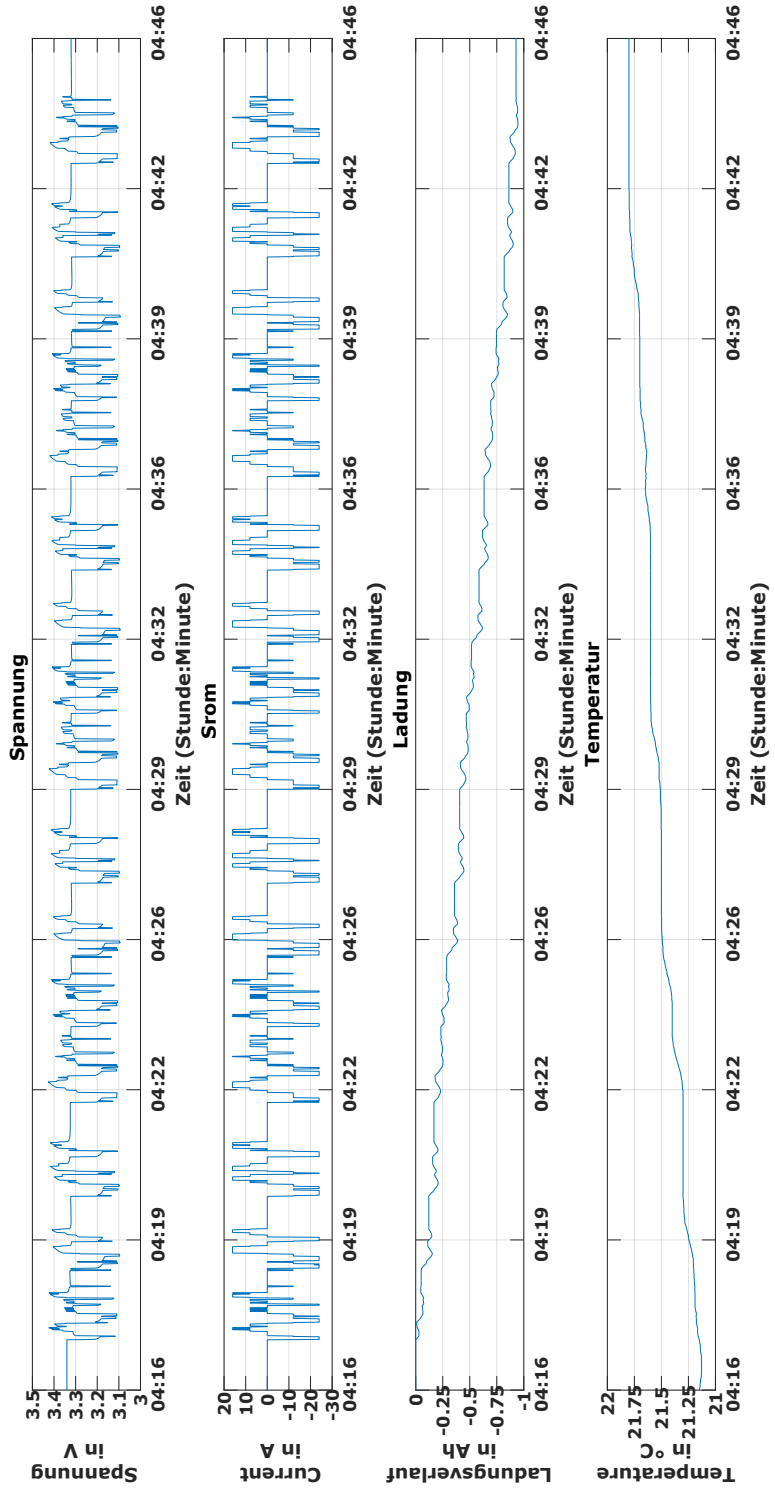


Abbildung 6.11.: Probebetrieb des Zyklus-I mit der LiNANO-Zelle.

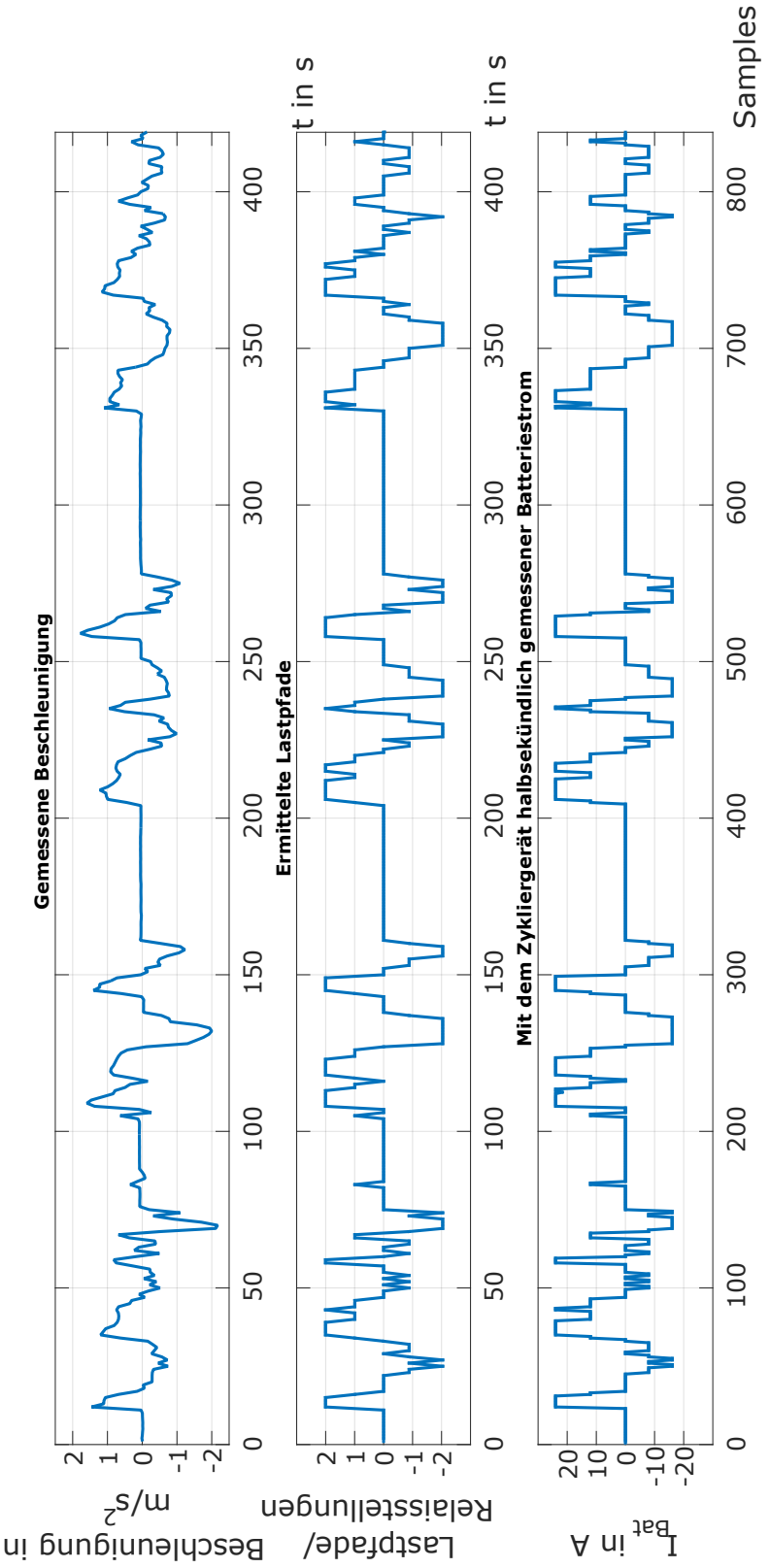


Abbildung 6.12.: Vergleich der Beschleunigungsdaten des Volvo Articulated Hybrid (oben), den daraus ermittelten Lastpfade/Relaisstellungen des Zyklus-I (Mitte) und den im Probebetrieb gemessenen Strömen der LiNANO-Batterie (unten).

6.3.1.2. Mehrfache Durchläufe des Zyklus-I mit der LiNANO-Zelle

Durch die 14-malige Aneinanderreihung des Zyklus-I entstanden die Verläufe Abb. 6.13. Diese Mehrfachzyklisierung wurde mit der LiNANO-Zelle $15\,\mathrm{Ah}$ durchgeführt.

Durch diesen Betrieb wurde eine Entladung über fast den gesamten Ladungsbereich $\frac{14\,\mathrm{Ah}}{15\,\mathrm{Ah}} = 93{,}3\,\%$ der Zelle erreicht. Nach der Beendigung des 14. Zyklus wurde die Zelle $30\,\mathrm{min}$ ruhen gelassen und anschließend mit einem Konstantstrom von $8\,\mathrm{A}$ bis zum Erreichen der Ladeschlussspannung $U_{\mathrm{charge}} = 3{,}7\,\mathrm{V}$ (gelb), geladen.

Bis zu dem 10. Zyklus liegt die Ruhespannung zwischen den Zyklen relativ konstant bei $3{,}3\,\mathrm{V}$. Ab dem 11. bis 13. Zyklus bricht die Spannung leicht ein. Deutlicher zu erkennen ist der Einbruch der Spannung beim 14. Zyklus (pink). Dies deutet an, dass sich die Zelle außerhalb ihres linearen Bereichs befindet und somit fast entladen ist.
Über die Zyklendauer ist ein leichter Temperaturanstieg zu vermerken. Dieser geringe Anstieg ist darauf zurückzuführen, dass die Zelle immer nur kurzzeitig und mit relativ kleinen Strömen ($\leq 1{,}66\,\mathrm{C}$) belastet bzw. geladen wurde.

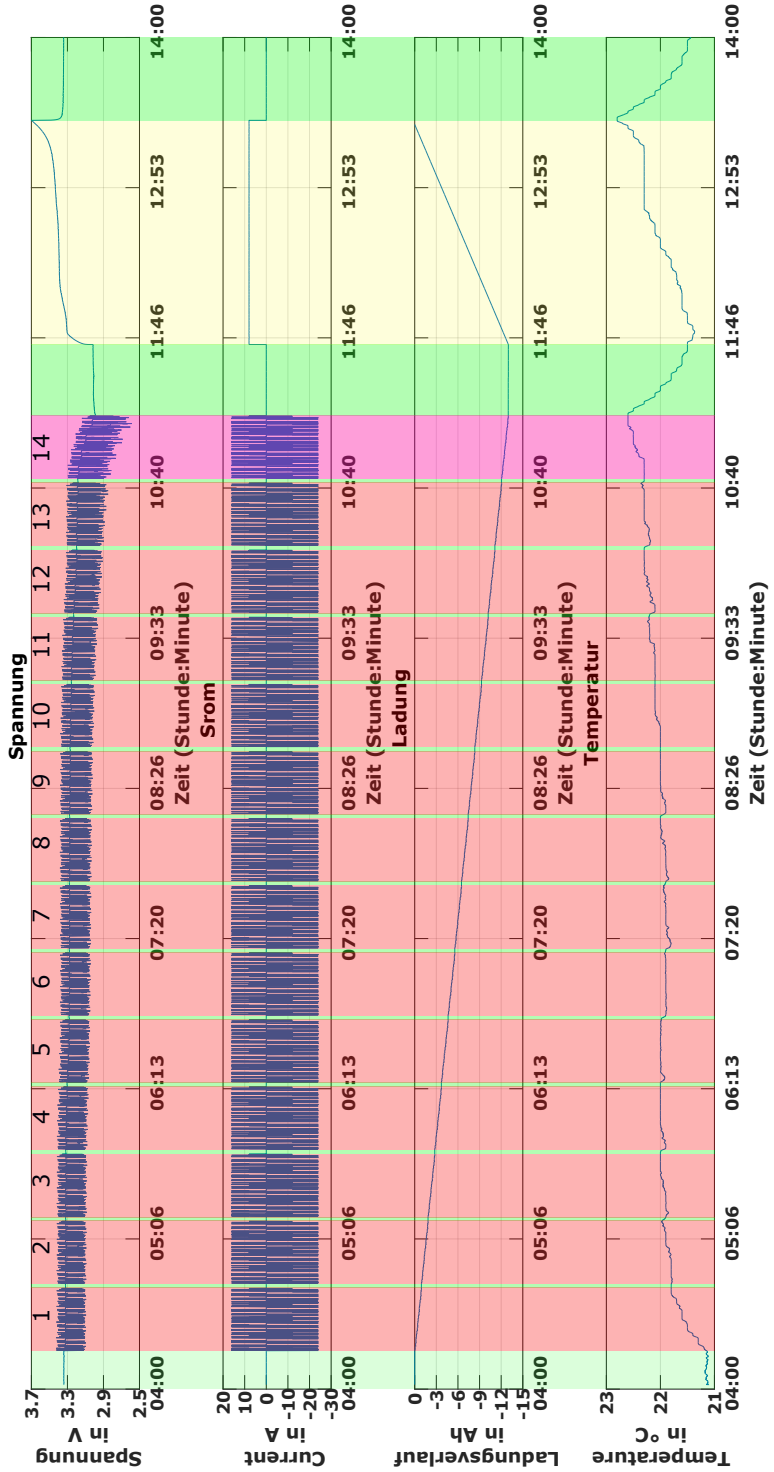


Abbildung 6.13.: 14-malige Wiederholung des Zyklus-I mit der LiNANO-Zelle.

6.3.1.3. Mehrfache Durchläufe des Zyklus-I mit jeweiligen Zwischenladungen der LiNANO-Zelle

Bei einer weiteren Zyklierung wurde durch siebenminütige Zwischenladungen versucht, die Ladephasen an den Endhaltestellen der Innovationsline 109, wie in dem Kapitel 2.2.1.1 beschrieben, zu simulieren. Durch die Aneinanderreihung von 10 Zyklen (rot) mit ihren nachfolgenden Ladephasen (gelb), sind die Verläufe in Abb. 6.14 entstanden.

In der dunkelgrünen Ruhephase ist zu erkennen, dass die Ladungsdifferenz zwischen der Nennkapazität und der entnommenen Ladung $0\,\mathrm{Ah}$ beträgt. Dies bedeutet, dass die über den jeweiligen Zyklus (rot) entzogene Leistung, durch die in gelb gekennzeichneten Zwischenladungen mit $8\,\mathrm{A}$ wieder vollständig geladen wurde.

Bei einem Vergleich einer vollständigen Entladung der Batterie (dunkelrot) und Ladung (grau) wird deutlich, dass durch die Zyklierung des Zyklus-I nur ca. $6{,}6\,\%$ des SOC genutzt wurde. Dies und die Zwischenladung mit ca. $1\,\mathrm{C}$ entspricht nicht den Erwartungen eines Busbetriebes. Durch die Rahmenbedingungen des Prüfstandes ist eine stärkere Belastung der LiNANO-Zelle nicht möglich. Aus diesem Grund wurde ein weiterer Versuch mit einer kleineren Zelle (siehe Abschnitt 6.3.1.4) durchgeführt.

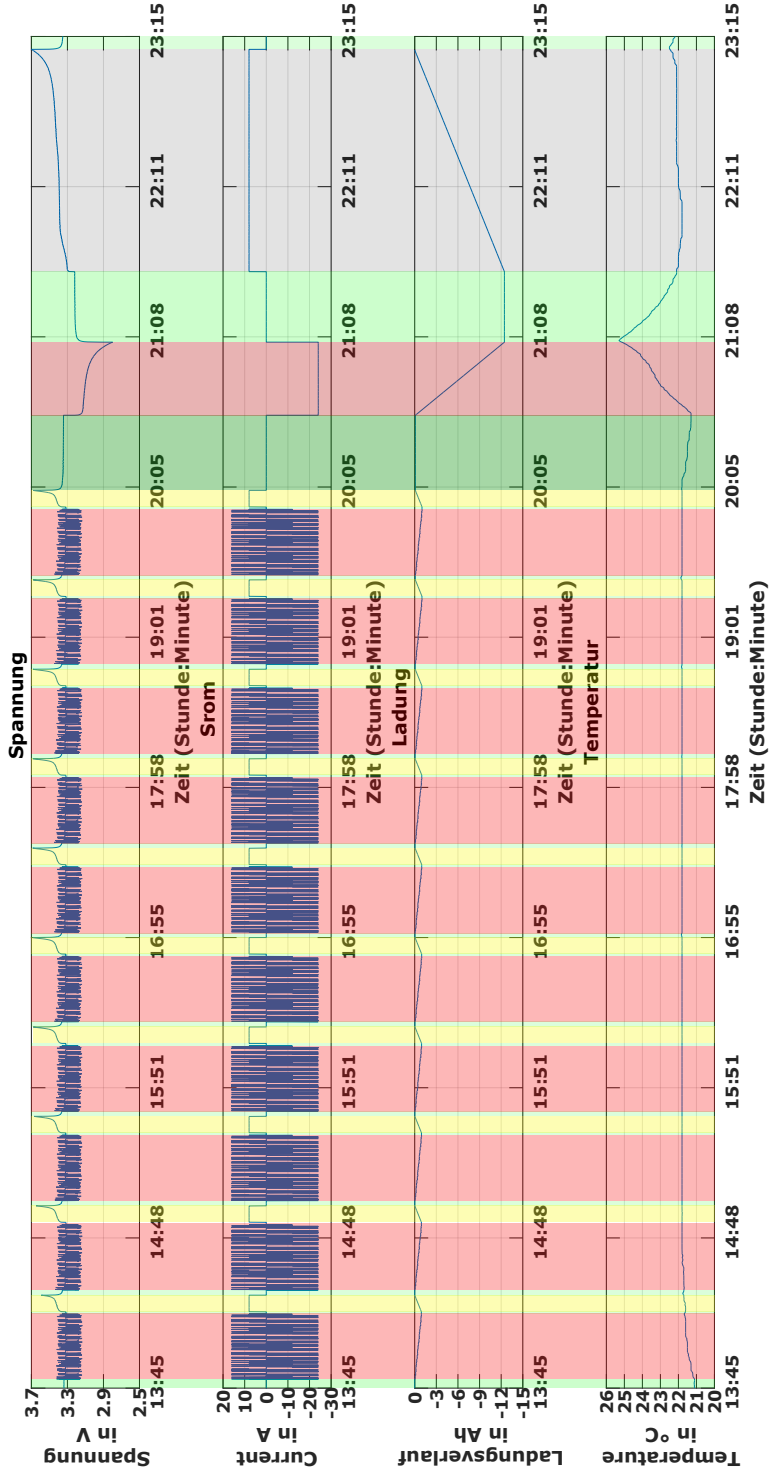


Abbildung 6.14.: Zehnmalige Wiederholung des Zyklus-I mit jeweiligen Zwischenladungen der LiNANO-Zelle.

6.3.1.4. Einmaliger Durchlauf des Zyklus-I mit der A123-Zelle

Bei den Untersuchungen des Zyklus-I an der LiNANO-Zelle wurde deutlich, dass durch die Begrenzung der Ströme in den 28 min nicht ausreichend entladen wird (siehe 6.3.1.3). Aus diesem Grund wurde der Zyklus-I an der kleineren A123-Batterie weiter erprobt. Folgende Ströme wurden, wie bereits in der Tabelle 3.4 beschrieben für den Probebetrieb gewählt:

- ein Entladestrom von 25 A (10 C) über das Relais 1,
- ein Entladestrom von 15 A (5 C) über das Relais 2,
- ein Ladestrom von 7,5 A (3 C) über das Relais 3 und
- ein Ladestrom von 15 A (5 C) über das Relais 4.

Des Weiteren wurde die Dauer des Zyklus-I von 28 min auf 49 min verlängert (sieben Loops). Der erprobte Zyklus-I an der A123-Batterie ist in Abb. 6.15 dargestellt.
In der Abbildung ist zu erkennen, dass durch die Wahl von 49 min die Batterie um 2,1 Ah entladen wird und die Spannung nicht unter 2,6 V sinkt. Eine Sicherheitsabschaltung durch das Zykliergerät würde bei einer dreimaligen aufeinanderfolgenden Unterschreitung der Spannungsgrenze von 2,5 V auslösen.
Eine Entladung der Zelle um ca. 2 Ah entspricht einer Nutzung von 80 % SOC.
Über den Zeitraum des Tests ist ein deutlicher Temperaturanstieg von ca. 3,5 °C zu vermerken. Dies ist auf die hohen C-Raten bei diesem Test zurückzuführen. Der Anstieg der Temperatur ist im Vergleich zu dem vorher durchgeführten Konstantstromtest (siehe Abschnitt 6.2.2.2) nur halb so groß.

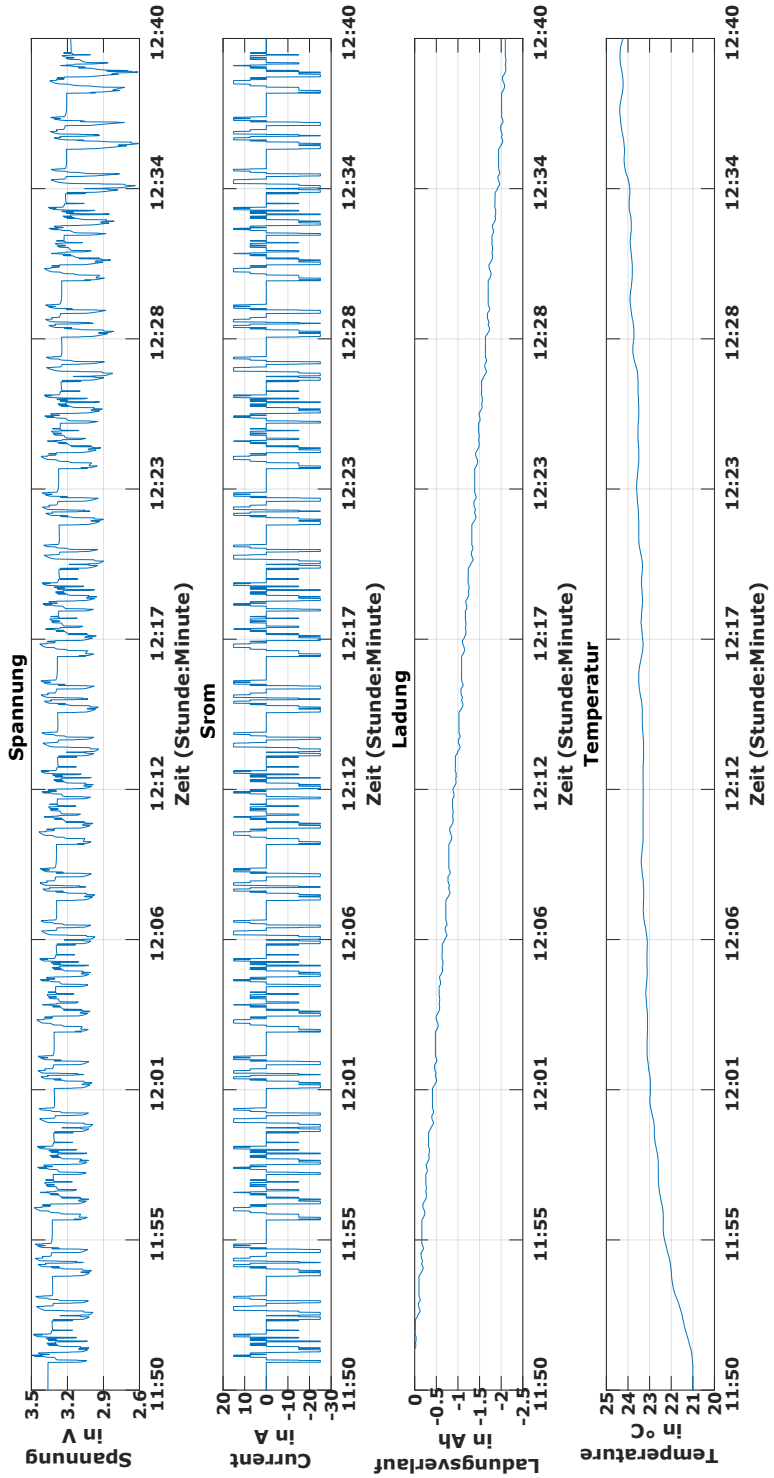


Abbildung 6.15.: Probebetrieb des Zyklus-I mit der A123-Zelle.

6.3.1.5. Mehrfache Durchläufe des Zyklus-I mit jeweiligen Zwischenladungen der A123-Zelle

Vergleichbar mit den Zwischenladungen der LiNANO-Zelle wurde die A123 Zelle mit den Strömen aus der Tabelle 3.4 und einer mehrfachen Zyklierung (8 Zyklen in rot) mit Zwischenlandungen getestet . Die Zwischenladung wurde durch eine 17-minütige Konstantstromladung mit $7{,}5\,\mathrm{A}$ (gelb) durchgeführt. Das Ergebnis der Zyklierung ist in Abb. 6.16 dargestellt.

Bei dieser Zyklierung ist zu sehen, dass die Zelle $80\,\%$ des SOC Bereichs ausnutzt. Dies im Vergleich zu dem Zyklus-I mit der LiNANO-Zelle (siehe 6.3.1.3), die nur $6{,}6\,\%$ SOC nutzt, eine deutliche Steigerung. Zu berücksichtigen ist, dass der mit der A123-Zelle erprobte Zyklus-I $21\,\mathrm{min}$ länger ist als der Zyklus-I an der LiNANO-Zelle.
Des Weiteren ist in der Abbildung zu erkennen, dass vereinzelte Werte die Grenze für die Sicherheitsabschaltung unterschreiten. Bei der näheren Auswertung ergab sich, dass beispielsweise im pink markierten Bereich nur ein Wert unter die Spannung von $2{,}5\,\mathrm{V}$ fiel, und somit keine Sicherheitsabschaltung notwendig war.
Deutlich wird auch wie die Temperatur der Zelle von der Ladung der Batterie abhängt. In dem Bereich in dem die Spannung einbricht, ist ein deutlicher Temperaturanstieg zu verzeichnen. Von Zyklenbeginn bis zur ersten Zwischenladung steigt die Temperatur der Zelle um $3{,}5\,^{\circ}\mathrm{C}$.

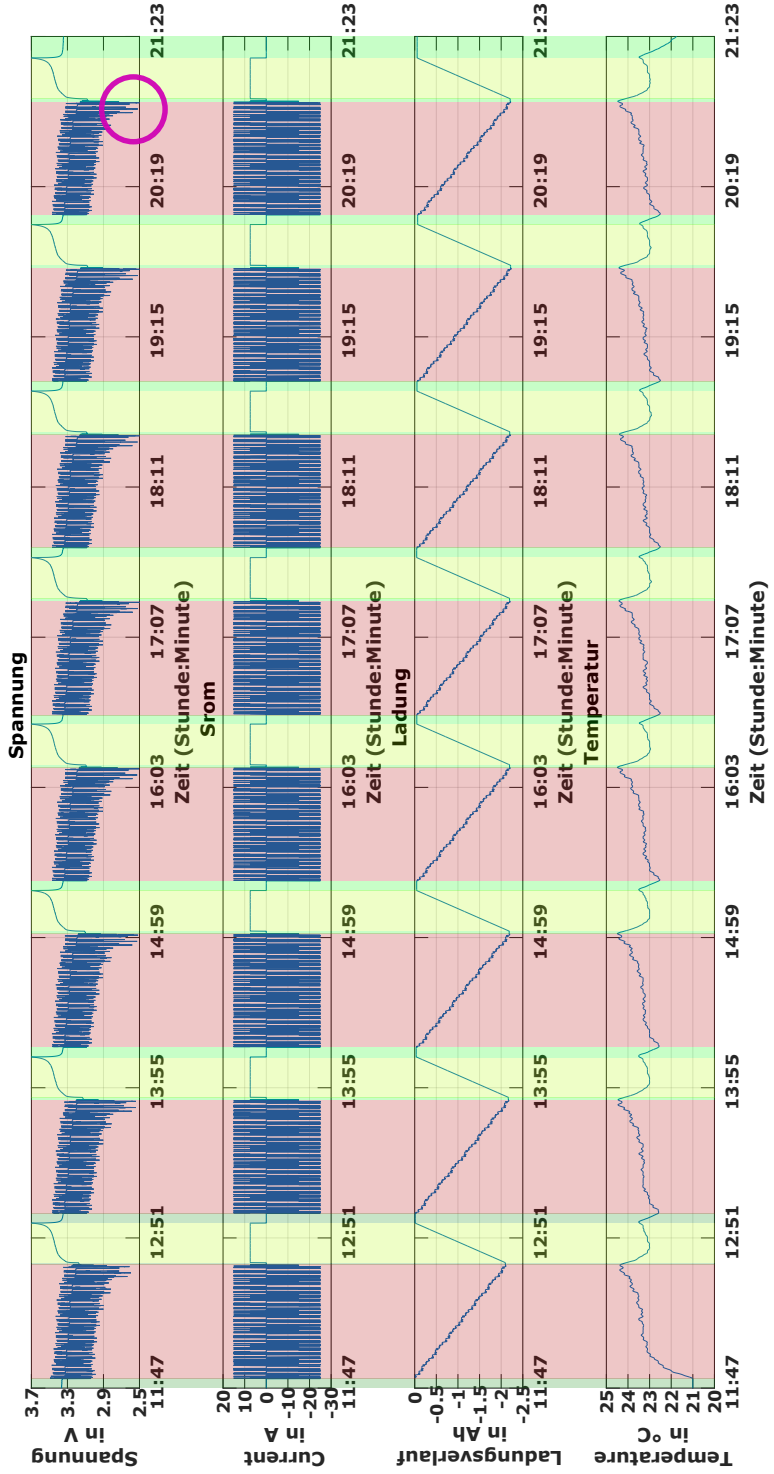


Abbildung 6.16.: Achtmalige Wiederholung des Zyklus-I mit jeweiligen Zwischenladungen der A123-Zelle.

6.3.2. Probebetrieb des Zyklus-E

Aus der Zyklierung des unter Kapitel 3.2.2 entstandenen siebenminütigen Zyklus-E und der in Tabelle 3.5 festgelegten Ströme, wurden die unter Abb. 6.17 dargestellten Messergebnisse mit dem Zykliergerät aufgezeichnet. Die voll geladene A123-Batterie wurde mit dem Zyklus-E für $42\,\mathrm{min}$ zykliert (rot) und danach über eine spannungsorientierte Ladung bis zu einer Ladeschlussspannung von $U_{\mathrm{charge}} = 3{,}7\,\mathrm{V}$ voll geladen (gelb). Über den gesamten Zyklus wurde der Batterie eine Ladung von $0{,}75\,\mathrm{Ah}$ entzogen. Dies entspricht, bezogen auf die Nennkapazität der Zelle ($2{,}5\,\mathrm{Ah}$), einer Entladung von $SOC_{\mathrm{Zyklus}} = 30\,\%$.

Bei der weiteren Untersuchung der EDDA-Messdaten, wurde mit dem in Formel (3.9) hergeleiteten Batteriestrom I_{Bat}, die Ladung bzw. die Entladung der Batterie über das sogenannte Coulomb counting (Integration der Stromwerte über die Zeit) berechnet. In Abb. 6.18 ist der Ladezustand der Batterie von Beginn bis zum Ende der Fahrt dargestellt. Die Nennkapazität der EDDA-Bus Zelle beträgt $46\,\mathrm{Ah}$ (siehe Tabelle 2.4). Daraus ergibt sich, dass die Zelle um $SOC_{\mathrm{Messdaten}} = 28\,\%$ entladen wurde. Dieser Wert stimmt fast mit dem aus der Zyklierung bestimmten $SOC_{\mathrm{Zyklus}} = 30\,\%$ überein.

Für den direkten Vergleich der EDDA-Stromdaten aus dem zyklierten Zyklus-E, wurden die Ströme I_{Bat} des EDDA Busses und die Ströme, die bei der Zyklierung durch das Zykliergerät gemessen wurden, übereinander gelegt. Dazu wurde der repräsentative $7\,\mathrm{min}$ Abschnitt der EDDA-Stromdaten und die ersten $7\,\mathrm{min}$ der Zykliergerät Messdaten in Abb. 6.19 zusammen dargestellt. Beim Vergleich der beiden Profile ist, obwohl das Zykliergerät nur eine fünfstufige Lastauflösung besitzt, eine gute Übereinstimmung zu erkennen. Daraus lässt sich schlussfolgern, dass eine Simulation des Fahrverhaltens des EDDA-Busses erfolgreich auf dem Prüfstand getestet wurde.

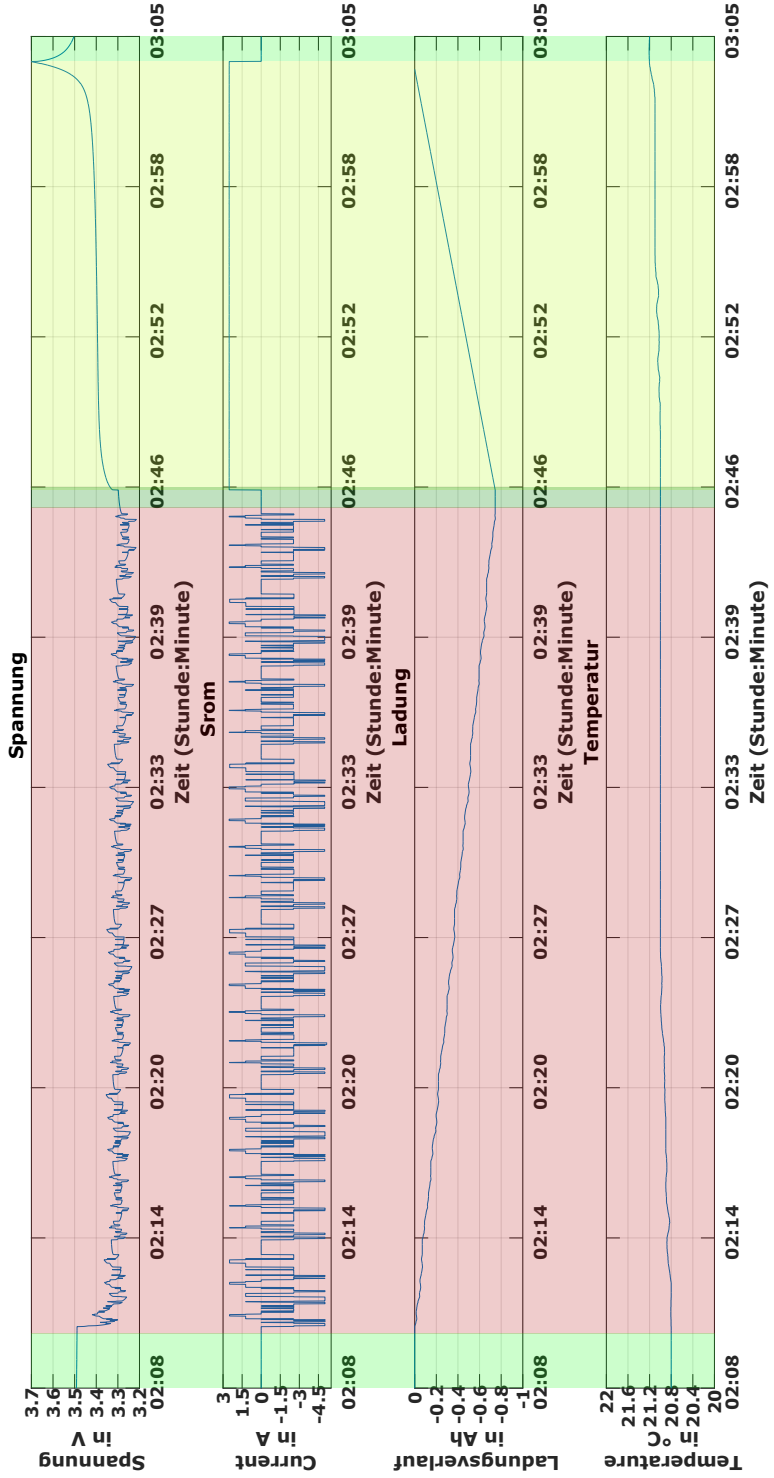


Abbildung 6.17.: Darstellung des einmalig zyklierten Zyklus-E.

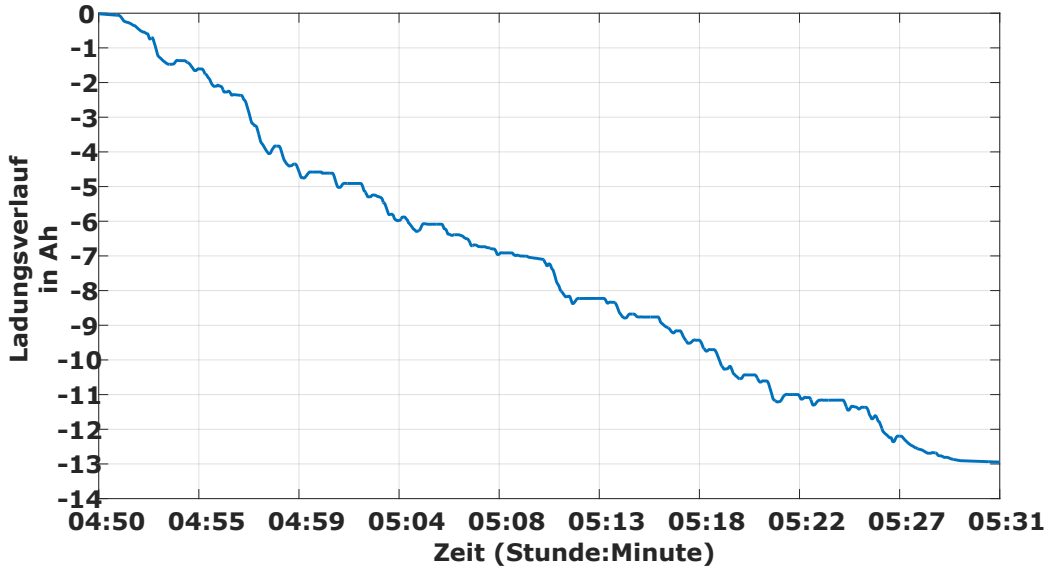


Abbildung 6.18.: Bestimmung der Ladung der Batterie aus dem Strom I_{Bat}.

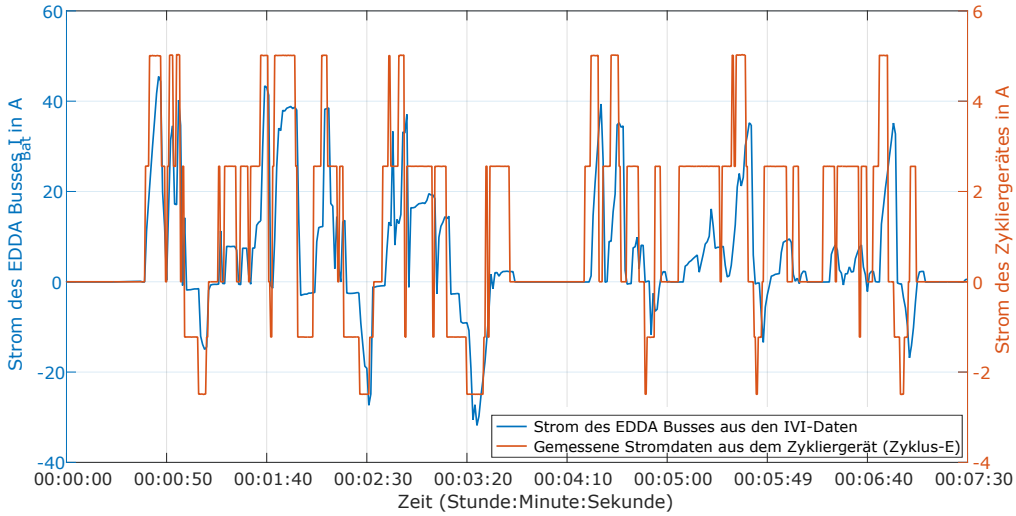


Abbildung 6.19.: Vergleich der, mit dem Zykliergerät gemessenen, Stromdaten bei der Zyklierung des siebenminütigen Zyklus-E mit dem originalen Auszug der EDDA-Stromdaten.

7. Fazit

7.1. Zusammenfassung

Gegenstand dieser Arbeit war es einen Batterieprüfstand anzupassen und mit diesem ein typisches Lastprofil von Elektrobussen (Elektrobuszyklus) auf $LiFePO_4$-Batterien anzuwenden.

Hierfür wurde zunächst aus den Fahrprofilen eines Elektrobusses mit Hilfe von ausgewählten Datenloggern Beschleunigungsdaten an Elektro-Hybrid-Bussen der Hamburger Innovationslinie 109 erhoben und bewertet (Kapitel 3.1). Des Weiteren wurden die Messdaten eines Elektrobuses (EDDA), die von dem Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme zur Verfügung gestellt wurden, analysiert und ausgewertet. Anhand dieser Daten wurde, durch die Herleitung eines Modells (Kapitel 2.3), ein direkter Bezug zwischen dem Fahrverhaltens des EDDA-Busses und der in die Bus-Batterie fließenden Ströme des Busses entwickelt.

Für die Tests von Batterien verwendet die Forschungsgruppe BATSEN einen Batteriezellen-Zyklierprüfstand. Kern dieses Prüfstandes bildet ein Zykliergerät, das die Aufgabe der überwachten Ladung und Entladung der Batterien übernimmt. Um diese Aufgabe ausführen zu können, verfügt das Zykliergerät über vier Lastrelais über, die mit externen Geräten wie Spannungs- bzw. Stromquellen und Lasten eine stufenförmige Belastung der Batterien ermöglicht wird.

Es galt diesen Prüfstand so anzupassen, dass dieser einen typischen Elektrobuszyklus nachbilden und auf einzelne ausgewählte $LiFePO_4$ Batteriezellen projizieren kann. Für die Anpassungen wurde der bestehende Prüfstand analysiert und die neuen Hard- und Softwareanforderungen formuliert (Kapitel 4) und umgesetzt (Kapitel 5).

Anhand der Messdaten zum Fahrverhalten der Elektrobusse und die durch den Prüfstand gegebenen Rahmenbedingungen, wurden zwei Buszyklen erstellt (Kapitel 3.2).

- Der erste Elektrobuszyklus (Zyklus-I) basiert auf aufgenommenen Beschleunigungsdaten der Innovationslinie und hatte zum Ziel, die Lithium-Ionen-Batterien mit einer hohen dynamischen Belastung zu zyklieren.

- Der zweite Elektrobuszyklus (Zyklus-E) wurde aus den vom Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme gemessenen Stromdaten entworfen und über die C-Raten der Batterien definiert. Die Definition über die C-Raten ermöglichte es, verschieden große Batterien zu zyklieren und zu vergleichen.

Aufgrund der gegebenen Anforderungen wurde der Prüfstand bzw. das Zykliergerät ergänzt und erweitert (Kapitel 5). Es wurden folgende Arbeiten durchgeführt:

- Die Ergänzung und Erweiterungen von Kalibrierfunktionen im Zykliergerät.
- Implementierung einer neuen Temperaturmessung (über 1-Wire) im Zykliergerät.
- Erhöhung der Taktrate der Lastrelaisumschaltung, sowie der Strom- und Spannungsmessung am Zykliergerät, aufgrund der Anforderungen der Elektrobuszyklen.
- Anpassungen an der Konfigurationsdatei des Zykliergerätes zur Abarbeitung längerer Zyklen.
- Neuaufbau der Hardware des Zykliergerätes.
- Aufbau eines Setups für die Zyklierung der Elektrobuszyklen (Zyklus-I und Zyklus-E).
- Aufbau und Anpassung von Batterieboxen für den sicheren Betrieb der $LiFePO_4$-Batterien.
- Anpassung der Hard- und Software an einem 2. Zykliergerät.

Anhand von Testläufen wurde der angepasste Prüfstand getestet. In dieser Phase wurden verschiedene Untersuchungen an unterschiedlich großen $LiFePO_4$ Batterien durchgeführt (Kapitel 6) und weitere Anpassungen an dem Zykliergerät vorgenommen.
Nachdem ein stabiler Stand des Messbetriebs erreicht war, wurden die entwickelten Elektrobuszyklen (Zyklus-I und ZE) getestet. Auf dem Prüfstand sind folgende Untersuchungen durchgeführt worden:

- Ermittlung der Ruhespannungskennlinien.
- Der Aufnahme von verschiedenen Konstantstromkennlinien bei der Ladung und der Entladung der Batterien.
- Des Überstromverhalten an einer Batterie und der daraus entstehende Temperatureinfluss.
- Der Erprobung des Zyklus-I.
- Der Erprobung des Zyklus-E.

Die erfolgreich auf dem Prüfstand erprobten Elektrobuszyklen ergeben erste Ansätze für eine optimierte Planung des Fahrverhaltens von Elektrobussen wie dem unter Kaptiel 2.2.1.1 beschriebenen Volvo Electric Hybrid Busses der Hamburger Hochbahn.

7.2. Ausblick

Die Entwicklung der beiden Elektrobuszyklen erfolgte auf dem bereits im BATSEN-Projekt bestehenden und überarbeiteten Prüfstands. Die entwickelten und implementierten Prüfstandsanpassungen ermöglichten eine fünfstufige (vier Lastpfade und ein Ruhepfad) Simulation des Fahrverhaltens eines Busses mit einer Umschaltgeschwindigkeit der Lastpfade von einer Stufe pro Sekunde. Unter diesen Bedingungen kann ein echter Buszyklus nur angenähert werden. Des Weiteren sind die begrenzten Ströme der Spannungsquellen (max. $20\,\mathrm{A}$) und der Lasten (max. $25\,\mathrm{A}$) für den Test von größeren Lithium-Ionen-Batterien wie die ECC-Zelle ($42\,\mathrm{Ah}$) nicht ausreichend. Mit dem Prüfstand kann nur eine Zyklierung mit $0{,}5\,\mathrm{C}$ erfolgen. Für weitere Tests von Elektrobuszyklen wird daher empfohlen, einen neuen Prüfstand zu entwickeln, der folgende Kriterien erfüllt:

- Eine Mindestumschaltzeit der Lastpfade sollte mit $>10\,\mathrm{Takte/s}$ erfolgen, da wie unter Kapitel 3.1.2.2 beschrieben, das Fahrverhalten eines Elektrobusses hauptsächlich in einem Frequenzbereich von 0 bis $5\,\mathrm{Hz}$ liegt.
- Ein stufenlos einstellbarer Strom ist vorzuziehen. Dies könnte durch eine PWM gesteuerte MOSFET-Schaltung in Kombination mit Lastwiderständen realisiert werden. Bei einer Umschaltung der Lade- bzw. Entladeströme in Stufen, sollten mehr als vier Lastpfade vorgesehen werden.
- Da die Messung einer einzelnen Batterie nicht repräsentativ ist, ist ein Mehrkanal-Prüfstand notwendig.
- Es sollte möglich sein, große Batterien in diesem Prüfstand zu testen. Dies bedeutet in Bezug auf die maximalen Kenndaten der ECC-Zelle (siehe Tabelle 2.1), dass der Prüfstand für den Bereich von $\pm 126\,\mathrm{A}$ ($3\,\mathrm{C}$) ausgelegt sein muss. Eine Auslegung für größere Ströme ist zu empfehlen.
- Die Hinterlegung der Zyklenabläufe sollte in einem Format geschehen, das eine möglichst große Anzahl von Zyklen zulässt, keine zeitliche Begrenzung aufweist und die verschiedensten Lade- bzw. Entlademethoden ermöglicht. Dazu zählen Funktionen wie:
  - Die Ladung bis zu einer bestimmten Spannungsgrenze mit einem Konstantstrom oder einem einstellbarem Stromprofil.

  - Die Ladung über CC-CV (constant current constant voltage).
  - Die Entladung bis zu einer bestimmten Spannungsgrenze mit einem Konstantstrom oder einem einstellbaren Stromprofil.
  - Die Ladung bzw. Entladung über eine vorgegebene Zeit mit einem Konstantstrom oder einem einstellbaren Stromprofil.

- Die Spannungsmessung sollte eine Genauigkeit von $1\,\mathrm{mV}$ aufweisen, um den teilweise linearen und mit einer geringen Steigung versehenen Spannungsverlauf der Lithium-Ionen-Batterien möglichst exakt aufzunehmen.
- Bei der Strommessung muss drauf geachtet werden, dass eine Messung von kleinen Strömen auch erfolgen kann. Dies ist bei dem in den Zykliergeräten verwendeten Hall-Effekt-Sensoren nur bedingt der Fall.
- Eine Temperaturaufzeichnung der Zelle sollte möglichst über mehr als nur einen Temperatursensor erfolgen, da das Temperaturverhalten einer Batterie an ihren Anschlüssen anderes ist als an dem Zellmantel.
- Zur Betriebssicherheit des Prüfstandes sollten folgende Aspekte beachtet werden:
  - Die Sicherheitsfunktionen sollten auf einen Sicherheits-Mikrocontroller ausgelagert werden. Dies ermöglicht bei Absturz des Hauptmikrocontrollers eine kontrollierte Abschaltung und evtl. Neustart des Prüfstandes.
  - Notabschaltung über große Lastrelais, die ggf. über Hilfskontakte verfügen, über die der Zustand des Sicherheitsrelais (geschlossen/offen) abgefragt werden kann. Alternative Lösungen wären auch eine Spannungsmessung über die Lastkontakte oder eine weitere Strommessung im Lastpfad.
  - Einbindung eines Rauchmelders in das Sicherheitssystem.

Der in Kapitel 6 getestete Zyklus-I ist auf der Basis der in Fahrtrichtung aufgezeichneten Beschleunigungsdaten des Volvo 7900 Articulated Hybrid Busses entstanden. Da eine vollständige Auswertung der Daten aufgrund der fehlenden Geschwindigkeiten (siehe 3.2.1.4) nicht möglich war, sollen weitere Elektrobusdaten aufgenommen und ausgewertet werden. Für die Entwicklung von weiteren Elektrobusprofilen sollten folgende Signale gleichzeitig aufgezeichnet werden:

- Zeit und Datum
- Beschleunigung.
- Geschwindigkeit.

- Elektrische Größen wie Strom des Fahrmotors, Spannung des Zwischenkreises, Strom der Batterien und die Spannung über die Batterie.

Dies sollte mit einer Messrate von $\gg 10\,\mathrm{Samples/s}$ geschehen (siehe 3.1.2.2).
Im Rahmen des Projekts BEEDeL werden seit Mitte Mai 2015 Datenlogger in den Bussen der Innovationsline 109 eingesetzt. Diese Daten stellen eine Grundlage für weitere Analysen da. Mit dem in Kapitel 3.2.2.1 beschrieben Modell und den bekannten Kenndaten der Elektrobusse (siehe Tabelle 2.2 und 2.3) ist ein Bezug zu der Leistung und damit auch zu dem Strom der Batterie herzustellen.
Bei den Tests des Zyklus-E wurde eine Fahrt für eine Strecke bzw. Linie des EDDA Busses ausgewertet. Da die Messdaten sehr umfangreich sind und noch weitere Strecken enthalten, können sie zur Entwicklung weiterer Zyklen genutzt werden. Der Vorteil an diesen Daten ist, dass der direkte Zusammenhang zwischen der Fahrdynamik und den elektrischen Größen (Strom und Spannung) im Bus vorliegen.
Der direkte Vergleich zu bekannten Elektrobuszyklen wie z. B. dem SORT (Standardised On-Road Test cycles) ist in dieser Arbeit nicht mehr durchgeführt worden. Die Implementierung eines SORT-Zyklus auf dem Prüfstand und der direkte Vergleich zu den im Feldtest aufgenommenen Elektrobuszyklen wäre sehr interessant.

Tabellenverzeichnis

2.1. Kennwerte der Batterien 20
2.2. Kennwerte des Volvo 7900 Electric Hybrid [42] 23
2.3. Kennwerte des Volvo 7900 Articulated Hybrid [41] 24
2.4. Kennwerte des EDDA-Busses [9] 25

3.1. Kennwerte des Smartphones 33
3.2. Darstellung der Messgrößen des EDDA-Busses, die in der Strukture 'Daten' erfasst wurden 42
3.3. Die maximalen Strombelastungen der Batterien auf dem Prüfstand 46
3.4. Festlegung der Lade- und Entladeströme für den Zyklus Zyklus-I 49
3.5. Festlegung der Lade- und Entladeströme für den Zyklus Zyklus-E 59

4.1. Kennwerte des Temperaturschrankes Typ ARU-0680 der Firma: ESPEC [10] . 62
4.2. Kennwerte der passiven Lastwiderstände 62
4.3. Kennwerte der elektronischen Last Typ 3227.1 der Firma Statron 63
4.4. Kennwerte der Spannungs- bzw. Stromquellen 63
4.5. Kennwerte der Zykliergeräte 65
4.6. Die wesentlichen Kennwerte der Lastrelais mit einem Nennstrom von 500 A und einem Nennstrom von 30 A. 66
4.7. Darstellung der gemessenen An- und Abfallzeiten (Totzeiten) aus den Oszilloskopmessungen Abb. 4.2 und Abb. 4.3 68
4.8. Innenwiderstände der Zykliergeräte 70

5.1. Beispiel eines Spannungsbereis für die quadratische Kalibrierung 77

6.1. Aufgenommende ADC-Digits des Zykliergerätes bei der jeweiligen vom Tektronix (DMM4020) gemessenen Spannung für den 1. Spannungskanal 89

B.1. Geräteliste des Messaufbaues 135

Abbildungsverzeichnis

1.1. Der Volvo 7900 Electric Hybrid der Innovationslinie 109 an der Endhaltestelle Hauptbahnhof/ZOB in Hamburg beim Ladevorgang. 10

2.1. Ragone-Diagramm für elektrische Energiespeicher. [18] 12
2.2. Ragone Diagramm für verschiedene Energiespeichertechnologien [18] 13
2.3. Aufbau und Funktion einer Lithium-Ionen-Zelle [31] 14
2.4. Stark vereinfachtes lineares Ersatzschaltbild einer Batterie links und das vereinfachte Ersatzschaltbild einer Batterie in Abhängigkeit des SOC rechts. . . 17
2.5. Typisches Verhalten der Batteriespannung in Abhängigkeit vom Ladegrad und Strom [18][1]bei einer Lithium-Ionen-Batterie. 18
2.6. Abhängigkeit der Zellspannung und Kapazität vom Entladestrom für eine Lithium-Ionen-Batterie [18] 19
2.7. Schematische Darstellung des Volvo 7900 Electric Hybrid Antriebs [40] . . . 22
2.8. Exemplarische Fahrt eines Volvo 7900 Electric Hybrid Busses [40] 22
2.9. Kräfte und Drehmomente am Fahrzeug [32][2]. 26
2.10. Blockschaltbild des Antriebes und der im Antrieb entstehenden Verluste. Abbildung in Anlehnung an [29] erstellt 30

3.1. Gängiges Fahrzeug-Koordinatensystem. Die Beschleunigung wird positiv und das Bremsen negativ dargestellt [30] [3]. 32
3.2. Die gängige Definition des Android-Smartphone-Koordinatensystems (rechts) mit dem Weltkoordinatensystem (links) nach der 'SensorEvent API' [4] 34
3.3. Exemplarischer Plot, der vom Smartphone aufgezeichneten Beschleunigungsmessdaten 35
3.4. Vergrößerter Ausschnitt der vom Smartphone aufgezeichneten Messdaten. . 36
3.5. Histogramm zeigt die Differenz $\Delta t_n = t_{n+1} - t_n$ zum nächsten Zeitwert bei der Smartphone-Messung. 37
3.6. Der Beschleunigungssensor mit drei BNC-Buchsen und Stromversorgungs-Buchse 38
3.7. Frequenzgang des Beschleunigungsaufnehmers in Z-Richtung 39
3.8. Exemplarischer Plot der von dem Messsystem aufgezeichneten Beschleunigungsmessdaten. 40
3.9. Zeitlich gemitteltes Effektivwertspektrum der Messfahrt 41

3.10. Die Geschwindigkeit und die Beschleunigung des EDDA-Busses 43
3.11. Die Beschleunigung und der Strom in den Fahrmotor des EDDA-Busses . . . 44
3.12. Häufigkeitsverteilung der Beschleunigung. 47
3.13. Verteilung der Lastpfade/Relaisstellungen 48
3.14. Ausschnitt des zeitlichen Verlaufs der Beschleunigung sowie der einzelnen Klassen - Lastpfade/Relaisstellungen 49
3.15. Benötigte Zugkraft bei verschiedenen Beschleunigungen und keiner Steigung, am Beispiel des EDDA Busses. 52
3.16. Benötigte Radleistung bei verschiedenen Beschleunigungen und keiner Steigung, am Beispiel des EDDA Busses. 52
3.17. Benötigte Radleistung in Abhängigkeit der Beschleunigung bei konstanten Geschwindigkeiten und keiner Steigung. 53
3.18. Ein 7 min Auszug[4] des berechneten Leistungsprofils des EDDA Busses. Positive Leistung entspricht einem Beschleunigungsvorgang und negative Leistung einem Bremsvorgang (Rekuperation). 54
3.19. Die Tatsächliche Leistung und die errechnete Leitung des EDDA-Busses im Vergleich 55
3.20. Vergleich der elektrischen Leistung mit der berechneten Leistung 55
3.21. Ein Auszug[5] des Stromes in die Batterie des EDDA Busses. Positiver Strom (Entladung) - Beschleunigungsvorgang Negativer Strom (Ladung) - Bremsvorgang (Rekuperation) 57
3.22. Ein Auszug[6] des Stromverlaufs in die Batterie des EDDA Busses. Positiver Strom entspricht einem Beschleunigungsvorgang, ein negativer Strom einem Bremsvorgang (Rekuperation). Die Strompfade sind farbig dargestellt. 58

4.1. Prinzipschaltbild des Zykliergerätes am Beispiel von vier Zellen [44] 61
4.2. Messung der Anzugszeit des Lastrelais 1 und der Abfallzeit des Lastrelais 2 . 67
4.3. Messung der Abfallzeit des Lastrelais 1 und die Anzugszeit des Lastrelais 2 . 68
4.4. Abbildung einer Innenwiderstandsmessung an einem Lastpfad 70

5.1. Anschluss der Masse am Zykliergerät 75
5.2. Displayhandler *CALLPLUG* 76
5.3. Anschluss des positiven Spannungszweigs am Zykliergerät 77
5.4. Displayhandler *CALLMEAS* 78
5.5. Displayhandler*CALLPULL* 78
5.6. Die Kalibrierkoeffizienten in Hex-Format für die *config.txt* des Zykliergerätes. Erstellt durch das MATLAB-Skript *QuadratischeKalibrierung.m* (siehe AnhangD). 79
5.7. 1-Wire auf der Zykliergerät Platine. 83
5.8. Schaltplan der Mosfet-Schaltung [27] 83
5.9. Test für den negativen Temperaturbereich der 1-Wire Temperatursensoren . . 84

5.10. Neu aufgebauter Zellkoffer für die LiNANO-Zellen 85
5.11. Neuer Zellkoffer für die ECC-Zellen 86

6.1. Die bei der Kalibrierung mit Tektronix gemessenen Spannungen wurden über die, mit dem Zykliergerät R aufgenommenen, ADC-Digits in blau geplottet. Des Weiteren wurde die, aus den Werten interpolierte, quadratische Funktion in rot über die ADC-Digits geplottet. 90
6.2. Überprüfung der Spannungsmessung nach der Kalibrierung. Oben ist die Spannungsmessung des Tektronix und die des Zykliergerät R im direkten Vergleich, im unteren Plot ist die Differenz zwischen den beiden Messungen dargestellt. 91
6.3. Im oberen Plot ist die gemessene Spannung der LiNANO-Zelle, nach einem 2 h Konstantstromladevorgang mit 1 A, abgebildet (blau). In der folgenden 6 h Ruhephase ist die Annäherung der Leerlaufspannung an die Ruhespannung deutlich zu erkennen. Im unteren Plot ist die Annäherung an die Ruhespannung vergrößert (veränderter Spannungsausschnitt) dargestellt. 92
6.4. Ermittlung der Ruhespannungskennlinie der 15 Ah LiNANO-Zelle mit einem Konstantstrom von 1 A über eine Stufenlänge von 2 h. Die roten Sterne geben die erreichte Ruhespannung nach jeweils 6 h an. 93
6.5. Konstantstromkennlinien der 15 Ah LiNANO-Batteriezelle mit verschiedenen Konstantströmen. Im Vergleich dazu wurden die ermittelten Ruhespannungspunkte geplottet. 94
6.6. Ermittlung der Ruhekennlinie der 15 Ah A123-Zelle mit einer jeweiligen Ruhezeit von 4 h (blau). Die roten Sternchen zeigt die jeweilige erreichte Ruhespannung an. 95
6.7. Zeitverlauf einer Konstantstrom-Entladung Ladung von 8 C und von 10 C der A123-Zelle. 97
6.8. Spannung über SOC-Kennline, des unter Abb. 6.7 dargestellten Zeitverlaufes. 98
6.9. Konstantstromkennlinien der 42 Ah ECC-Batteriezelle bei verschiedenen Konstantströmen. 99
6.10. Zyklierung der ECC-Batterie mit einem Konstantstrom von 20 A in einem ansatzweise linearen Bereich von 25 % bis 75 % SOC. 100
6.11. Probebetrieb des Zyklus-I mit der LiNANO-Zelle. 102
6.12. Vergleich der Beschleunigungsdaten des Volvo Articulated Hybrid (oben), den daraus ermittelten Lastpfade/Relaisstellungen des Zyklus-I (Mitte) und den im Probebetrieb gemessenen Strömen der LiNANO-Batterie (unten). 103
6.13. 14-malige Wiederholung des Zyklus-I mit der LiNANO-Zelle. 105
6.14. Zehnmalige Wiederholung des Zyklus-I mit jeweiligen Zwischenladungen der LiNANO-Zelle. 107
6.15. Probebetrieb des Zyklus-I mit der A123-Zelle. 109

6.16. Achtmalige Wiederholung des Zyklus-I mit jeweiligen Zwischenladungen der A123-Zelle. 111
6.17. Darstellung des einmalig zyklierten Zyklus-E. 113
6.18. Bestimmung der Ladung der Batterie aus dem Strom I_{Bat}. 114
6.19. Vergleich der, mit dem Zykliergerät gemessenen, Stromdaten bei der Zyklierung des siebenminütigen Zyklus-E mit dem originalen Auszug der EDDA-Stromdaten. 114

B.1. Der Zellsimulator 134
B.2. Messaufbau zur Bestimmung des Frequenzganges der Z-Achse. 136
B.3. Zeitlich gemitteltes Effektivwertspektrum der Messfahrt für den Frequenzbereich von 0 bis 400 Hz. 136
B.4. 1Zyklus1ganz 137

C.1. Potenzialbereiche verschiedener Elektrodenmaterialien für Lithium-Ionen Zellen [32] 144
C.2. Speicherwirkungsgrad einer Lithium-Ionen-Batterie über den Spannungsunterschied bei Ladung und Entladung [18]. 145
C.3. Route der Innovationslinie 109 146
C.4. Kontakschine auf dem Bus für die Ladung der Batterie [17] 146
C.5. Ladestation an der Endhaltestelle 147
C.6. Ladeanzeige in der Konsole des Volvo 7900 Electric Hybrid 147
C.7. Der Volvo 7900 Electric Hybrid an der Endhaltestelle Hauptbahnhof/ZOB . . 148
C.8. Der Volvo 7900 articulated Hybrid (Gelenkbus) an der Endhaltestelle Hauptbahnhof/ZOB 148
C.9. Der EDDA-Bus des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme [13] 149
C.10. Der EDDA-Bus des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme an der Ladestation [19] 149
C.11. Beispielhafte Abbildung einer Schaltzeiten Messung an einem Lastpfad. . . . 150
C.12. Messung der Abfallzeit des Lastrelais 2 und die Anzugszeiten des Lastrelais 3 151
C.13. Schaltplan zur Veranschaulichung der ADC-Anbindung mit implementierter Schutzschaltung. Basiert auf dem Schaltplan V2.0 [44]. 152
C.14. Beginn einer Messung mit dem externen ADC 153
C.15. Ende einer Messung mit dem externen ADC 153
C.16. Abbildung des PULSE-Messsystem 154
C.17. Abbildung der externen Beschaltung des Beschleunigungssensors ADXL335 [23] 155
C.18. Dieses Histogramm zeigt die Differenz $\Delta t_n = t_{n+1} - t_n$ zum nächsten Zeit-Wert bei dem Zeitvektor der EDDA-Busdaten. 156

C.19.Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die aus der Geschwindigkeit abgeleitete Beschleunigung. 157
C.20.Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die aus der Geschwindigkeit abgeleitete Beschleunigung und der in den Fahrmotor fließende Strom. 157
C.21.Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die Tatsächliche Leistung und die aus dem Modell errechnete Leistung des Busses. 158
C.22.Der Strome in die Batterie des EDDA Busses. 158
C.23.Abbildung des EDDA-Stromstufen für den Zyklus. Positiver Strom einspricht einem Beschleunigungsvorgang und ein negativer Strom einem Bremsvorgang (Rekuperation) Die Stromstufen sind Farbig dargestellt. 159
C.24.MOSFET-Schaltung für den 1-Wire Bus. 159
C.25.1-Wire Buchsenverbindung zur ARM-Platine 160
C.26.Zyklierung der ECC-Battetrien bei verschiedenen Konstantströmen. 160

Literaturverzeichnis

[1] AG, EPCOS: NTC thermistors for temperature measurement, Probe assemblies, M703. (2009). – URL http://www.produktinfo.conrad.com/datenblaetter/500000-524999/500526-da-01-en-Heissleiter_Typ_M703_B57703M103G.pdf. – Zugriffsdatum: 19.05.2015

[2] ANALOG DEVICES, Inc.: AD7798/AD7799 3-Channel, Low Noise, Low Power, 16-/24-Bit Sigma-Delta ADC with On-Chip In-Amp Data Sheet (Rev. A). (2007). – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD7798_7799.pdf. – Zugriffsdatum: 27.05.2015

[3] ANALOG DEVICES, Inc.: ADXL335 (Rev. B). (2010). – URL http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf. – Zugriffsdatum: 23.05.2015

[4] ANDROID DEVELOPERS: *SensorEvent | Android Developers*. 2015. – URL http://developer.android.com/reference/android/hardware/SensorEvent.html. – Zugriffsdatum: 19.05.2015

[5] ATE: Models RB Hochlastwiderstand. (2015). – URL http://www.produktinfo.conrad.com/datenblaetter/625000-649999/645724-da-01-en-HOCHLASTWIDERSTAND_50WATT_5__0R47_20ST.pdf. – Zugriffsdatum: 19.05.2015

[6] BECKENDORF, Bastian: *HOCHBAHN - Flotte der 109*. 30.04.2015. – URL http://www.hochbahn.de/wps/portal/de/home/hochbahn/aktuelles/innovationslinie+site/0020+flotte+der+109?1dmy&current=true&urile=wcm%3apath%3a%2Fhochbahn_content%2Fhome%2Fhochbahn%2Faktuelles%2Finnovationslinie%2Bsite%2F0020%2Bflotte%2Bder%2B109. – Zugriffsdatum: 10.05.2015

[7] BRONSTEIN, Ilja N. ; SEMENDJAJEW, Konstantin A.: *Taschenbuch der Mathematik*. 7., überarb. und erg. Aufl. Frankfurt am Main : Deutsch, 2008. – ISBN 9783817120079

[8] BUNDESMINISTERIUM FÜR WIRTSCHAFT UND TECHNOLOGIE, BUNDESMINISTERIUM FÜR VERKEHR, BAU UND STADTENTWICKLUNG, BUNDESMINISTERIUM FÜR UMWELT, NATURSCHUTZ UND REAKTORSICHERHEIT, BUNDESMINISTERIUM FÜR BILDUNG UND

FORSCHUNG: Regierungsprogramm Elektromobilität. (2011). – URL http://www.bmbf.de/pubRD/programm_elektromobilitaet.pdf. – Zugriffsdatum: 16.06.2015

[9] DIPL.-ING. (FH) SVEN KLAUSNER ; OLIVER ALEXANDER SCHMIDT (Hrsg.): *Telefonat mit dem Projektleiter des EDDA-Busses, über den Aufbau der Leistungskomponenten des Busses. Telefonat und E-Mail*. 13.05.2015 - 26.05.2015

[10] ESPEC: *Environmental Stress Chamber: AR series*. 2015. – URL http://www.seneco.it/prospetti/espec/Camere%20climatiche%20serie%20AR.PDF. – Zugriffsdatum: 18.05.2015

[11] EWALTON: Product Data: IDAe Hardware Configurations for PULSE — Types 3560-B, 3560-C, 3560-D and 3560-E(bu022822). (20018). – URL http://www.xn--90abkkvlnd.eu/content/downloads/pulse/pulse_1950K_EN.pdf. – Zugriffsdatum: 23.05.2015

[12] FRANK STEINERT ; KARL RAGMAR RIEMSCHNEIDER ; GERHARD SCHENK ; PHILIPP KRÜGER: *Bewertung des Einsatzes von Elektrobussen mit dezentraler Ladeinfrastruktur in Metropolen am Beispiel der HOCHBAHN (BEEDeL): Kick Off Treffen am 20. März 2015*. 20.03.2015

[13] FRAUNHOFER IVI: *Forschungsgebiete - Fraunhofer IVI*. – URL http://www.ivi.fraunhofer.de/de/projekte_produkte.html. – Zugriffsdatum: 11.05.2015

[14] GHASSEMLOUI, Ashkan ; ECC BATTERIES GMBH (Hrsg.): *kunden-_technische_daten_ecc_batteries_gmbh*. 2015. – URL http://www.eccbatteries.com/files/kunden-_technische_daten_ecc_batteries_gmbh.pdf. – Zugriffsdatum: 19.05.2015

[15] GROTH, Felix A.: Datenlogger für Elektrobusse mit Mikrocontrollersteuerung, Inertialmesssystem sowie GPS- und GSM-Modulen: Bachelorthesis in der Informations- und Elektrotechnik an der HAW-Hamburg. (2015)

[16] HAMBURGER HOCHBAHN: BEEDeL: Bewertung des Einsatzes von Elektrobussen mit Dezentraler Ladeinfrastruktur in Metropolen am Beispiel der HOCHBAHN: Gemeinsame Vorhabenbeschreibung. (2015)

[17] HOCHBAHN: *Warum der Plug-In-Elektro-Hybridbus gut für die Zukunft ist*. – URL http://dialog.hochbahn.de/bus-in-zukunft/warum-der-plug-in-elektro-hybridbus-gut-fuer-die-zukunft-ist/. – Zugriffsdatum: 10.05.2015

[18] HOFMANN, Peter: *Hybridfahrzeuge: Ein alternatives Antriebssystem für die Zukunft*. 2. Aufl. Wien [u.a.] : Springer, 2014. – ISBN 978-3-7091-1779-8

[19] HOPPECKE BATTERIEN GMBH & CO. KG: *Hochleistungsfähige Batteriesysteme in schnellladefähigen Elektrobussen - ein möglicher Lösungsansatz, die Anforderungen des ÖPNV an Elektrobusse, erfüllen zu können*. 27.04.2015. – URL www.battery-power.eu

[20] JOSSEN, Andreas ; WEYDANZ, Wolfgang: *Moderne Akkumulatoren richtig einsetzen: 36 Tabellen*. 1. Aufl. Neusäß : Ubooks, 2006. – ISBN 3939359114

[21] JTL-SHOP3: *A123 Systems ANR26650M1B, 2500 mAh, Li-popower, LiNANO*. – URL http://shop.lipopower.de/A123-Systems-ANR26650M1B-2500-mAh. – Zugriffsdatum: 19.05.2015

[22] JTL-SHOP3: *Headway 40152SE LiNANO*. – URL http://shop.lipopower.de/LiNANOZ-15-Ah-32V-10C-Headway-40152SE. – Zugriffsdatum: 19.05.2015

[23] KAREL TAN: *Accelerometer Module Angular Sensor Schematic*. 2015. – URL http://eshinede.com/content/?518.html#.VWBMlk_tmko. – Zugriffsdatum: 23.05.2015

[24] KORTHAUER, Reiner: *Handbuch Lithium-Ionen-Batterien*. Berlin, Heidelberg : Imprint: Springer Vieweg, 2013 (SpringerLink : Bücher). – ISBN 978-3-642-30652-5

[25] LINDEN, David ; REDDY, Thomas B.: *Handbook of batteries*. 3rd ed. New York : McGraw-Hill, 2002 (McGraw-Hill handbooks). – ISBN 0-07-135978-8

[26] MARTIN KUSCHE: Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller. (2014)

[27] MAXIM INTEGRATED: *Using a UART to Implement a 1-Wire Bus Master - Tutorial - Maxim*. – URL http://www.maximintegrated.com/en/app-notes/index.mvp/id/214. – Zugriffsdatum: 28.05.2015

[28] MAXIM INTEGRATED: DS18B20: Programmable Resolution 1-Wire Digital Thermometer. (2008). – URL http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. – Zugriffsdatum: 27.04.2015

[29] MITSCHKE, Manfred ; WALLENTOWITZ, Henning: *Dynamik der Kraftfahrzeuge*. 5., überarbeitete und ergänzte Auflage. 2014. – ISBN 978-3-658-05067-2

[30] MORITZ VON ZITZEWITZ: *Android Smartphone als Fahrzeug Datenlogger*. 2012. – URL http://www.eislab.fim.uni-passau.de/files/publications/students/vonZitzewitz-Studienarbeit-small.pdf. – Zugriffsdatum: 19.05.2015

[31] PETER KLEINE-MÖLLHOFF ; HOLGER BENAD ; MARTINA KNÖLL ; FRANK BEILARD ; MOHAMMED ESMAIL: Die Batterie als Schlüsseltechnologie für die Elektromobilität der Zukunft: Herausforderungen - Potenziale - Ausblick. (April 2012). – URL http://www.esb-business-school.de/fileadmin/_research/dokumente/Diskussionsbeitraege/2012-3-Reutlinger-Diskussionsbeitraege-Mark-Mngmt-E-Mobility-Batter pdf. – Zugriffsdatum: 5.5.2015

[32] PROF. DR.-ING. JOACHIM BÖCKER: Antriebe für umweltfreundliche Fahrzeuge. (2014). – URL http://wwwlea.uni-paderborn.de/fileadmin/Elektrotechnik/AG-LEA/lehre/antriebe_fahrzeuge/vorlesung/Skript_Fahrzeugantriebe.pdf. – Zugriffsdatum: 07.05.2015

[33] PROF. DR.-ING. KARL-RAGMAR RIEMSCHNEIDER ; OLIVER ALEXANDER SCHMIDT (Hrsg.): *Nachbereitung des Meetings "Bewertung des Einsatzes von Elektrobussen mit dezentraler Ladeinfrastruktur in Metropolen am Beispiel der HOCHBAHN (BEEDeL)" Länge der Fahrzeugumläufe vs. Nachladezeiten der Batterien: Persönliches Gespräch*. 23.03.2015

[34] RÖHN, Johannes: Batteriezellen-Zyklierprüfstand mit ARM-Mikrocontroller Software sowie Mess- und Leistungsschaltung: Bachelorthesis Johannes Röhn. (2014)

[35] RÖHN, Johannes ; KUSCHE, Martin: Kurzanleitung v3 Zyklierprüfstand. (2014)

[36] SCHÜTZ, Thomas: *Hucho - Aerodynamik des Automobils: Strömungsmechanik, Wärmetechnik, Fahrdynamik, Komfort*. 6., vollständig überarbeitete und erweiterte Auflage. 2013 (ATZ/MTZ-Fachbuch). – ISBN 978-3-8348-1919-2

[37] TE CONNECTIVITY: *T92 Series Two-pole 30A PCB or Panel Mount Relay*. 2012. – URL http://www.farnell.com/datasheets/1729804.pdf. – Zugriffsdatum: 17.05.2015

[38] TEXAS INSTRUMENTS, INCORPORATED [SPMS180,M]: Stellaris LM3S9B92 Microcontroller Data Sheet (Rev. M). (2012)

[39] TYCO ELECTRONICS: *LEV200 Series Contactor With 1 Form X (SPST-NO-DM) Contacts Rated 500+ Amps, 12-900VDC*. 2015. – URL http://www.evsource.com/datasheets/tyco/LEV200_ds_9-1773439-1.pdf. – Zugriffsdatum: 17.05.2015

[40] VOLVO: Antriebsstrang-Volvo-7900-Electric-Hybrid_DE. (2014). – URL http://electrichybrid.volvobuses.com/pdf/Electric_Hybrid_Driveline_EU6_FS_DE.pdf. – Zugriffsdatum: 27.04.2015

[41] VOLVO: Volvo 7900 articulated Hybrid. (2014). – URL http://www.volvobuses.com/SiteCollectionDocuments/VBC/Downloads/Volvo-7905LAH-EU6-Data-sheet-13-03.pdf. – Zugriffsdatum: 27.04.2015

[42] VOLVO: Volvo 7900 ELECTRIC Hybrid. (2014). – URL http://www.hochbahn.de/wps/wcm/connect/de/8950c999-dbe5-4f18-8e15-e0b4be1e548a/Technische+Daten+Volvo+7900+Electric+Hybrid+DE.pdf?MOD=AJPERES&CACHEID=8950c999-dbe5-4f18-8e15-e0b4be1e548a. – Zugriffsdatum: 27.04.2015

[43] WETTERONLINE: *Wetter Rückblick Dresden - Wetterdaten weltweit - WetterOnline*. 2015. – URL http://www.wetteronline.de/wetterdaten/dresden?pcid=pc_rueckblick_data&gid=10488&pid=p_rueckblick_diagram&sid=StationHistory&iid=10488&month=12&year=2014&period=4&paraid=PDLD. – Zugriffsdatum: 30.05.2015

[44] WISNIEWSKI, Thomas: Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung. (2013)

A. Aufgabenstellung

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 23. Februar 2015

Bachelorarbeit: Oliver Schmidt

Auslegung und Erprobung eines Lithiumbatterie-Prüfstands für typische Lade- und Lastprofile von Elektrobussen

Motivation

In der Forschungsgruppe Batteriesensoren (BATSEN) an der HAW Hamburg werden Lithiumbatterien in verschiedenen Einsatzbereichen untersucht. Sie sollen auf einem Laborprüfstand mit gespeicherten Lade- und Lastverläufen zykliert werden, welche typischen Profilen der Anwendung entsprechen.
Im Falle von Elektro- und Hybridbussen besteht aktuelles Forschungsinteresse, weil noch wenig über spezifischen Betriebsprofile und die entsprechenden Batteriebelastungen bekannt ist.

Aufgabe

Herr Schmidt erhält die Aufgabe, für den Elektrobus ein typisches Betriebsprofil zu erstellen und auf Batteriezellen anzuwenden. Dabei werden an den Batterie-Zellen Lade- und Entladevorgänge mit vergleichsweise hoher Dynamik durchgeführt. Hierzu ist die Steuerung und ggfs. die Hardware des vorhandenen Prüfstandes anzupassen. Dieser ist bisher auf Langzeituntersuchungen ausgelegt. Dies trifft insbesondere auf das Zykliergerät zu, mit dem die Lade-Entladevorgänge durchgeführt und überwacht werden.
Für die Anpassungen sind zum einen die realen Anforderungen aus dem Busbetrieb zu analysieren und zum anderen ein Vorschlag für einen typischen „Buszyklus“ zu erarbeiten. Dieser Vorschlag soll zu der entstehenden Soft- und Hardwarelösung passen, die sowohl eine aussagekräftige Nachbildung der Batteriebelastungen des Busbetriebs erlaubt, als auch im Aufwand vertretbar ist. Beim Zyklierbetrieb sind Messwerte wie Zellspannung, Strom und Temperatur aufzuzeichnen. Im Focus der praktischen Untersuchungen sollen große Lithium-Eisenphosphatzellen ($LiFePO_4$) stehen. Hierbei ist der Aspekt des Temperatureinflusses auf die Batterien mit Hilfe eines gesteuerten Temperaturschrankes zu betrachten. Die praktische Nutzung ist umfassend zu erproben und zu dokumentieren.

Gliederung

Die Aufgabe der Bachelorarbeit gliedert sich wie folgt:

1) **Analyse der Rahmenbedingungen und Vorarbeiten**
   - Einarbeitung durch Recherche und Literatur in die Thematik
   - Einarbeitung in die bestehende Hard- und Software des Zykliergerätes
   - Recherche von Elektrobusdaten und Betriebsabläufen.
2) **Konzeption des Elektrobuszyklus**
   - Exemplarische Erfassung des Zeitverhalten im Busbetrieb
   - Erarbeiten von Vereinfachungen, wie reduzierte Schaltzeiten und Stromstufen

- Vergleich zu literaturbekannten Betriebszyklen aus dem konventionellen und Elektrobetrieb (z.B. NEFZ für PKW, Braunschweiger Zyklus für Dieselbusse, SORT für Hybridbusse)
- Entwurf und Erstellung eines Zyklusses für die Optimierung des Zykliergerätes bei hoher dynamischer Belastung

3) **Konzeption der Prüfstandsanpassungen**

- Inbetriebnahme des vorhandenen Zykliergerätes, Fehlerbehebung, Kalibrierung
- Analyse von Verbesserungs- und Erweiterungs-Möglichkeiten am Prüfstand
- Vergleich Schalt-Relaissteuerung und Leistungselektronik, insbesondere im Zeitverhalten
– Zeitverhalten Messrelaissteuerung zur Zellenumschaltung, Vor- und Nachteile bei Mehrzellenbetrieb
- Arbeitsplanung und Spezifikation für die Soft- und Hardware Anpassungen

4) **Softwareanpassungen**

- Ergänzungen und Erweiterungen von Kalibrierfunktionen im Zykliergerät
- Reduzierung der Mindesschaltzeiten vom Minutenbereich auf den Sekundenbereich in der Steuer-Software
- Konzept und Implementierung für Mindestschaltzeiten der Steuer-Software auf einen gesicherten/stabilen Mindestwert
- Erprobung der angepassten Software

5) **Anpassungen für die Hardware des Prüfstandes**

- Weitere Anpassungen bzw. Änderungen am Zykliergerät
- Optional: Planung und Erweiterung der externen Geräte für Einspeisung und für eine passive Last
- Erprobung der Hardwareerweiterungen

6) **Planung und Durchführung von Batterieuntersuchungen**

- Versuchs-Zielstellungen für den Elektrobuszyklus
- Messplanung der Versuchsreihe
- Aufbau einer Messumgebung (u.a. Batterieboxen im Metallkoffer)
- Zyklierung ausgewählter Zellen

7) **Auswertung und Bewertung**

- Auswertung der Ergebnisse der Messreihen
- Bewertung der Versuchsergebnisse, Diskussion mit Anwendungsbezug
- Vorschläge zur Weiterentwicklung des Zykliersystems, für weitere Betriebsprofile und Versuchsreihen

Dokumentation

Die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Die gewählte Lösungen sind gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungem und wesentlichen Entwurfsentscheidungen sollen beschrieben werden. Die Ergebnisse sind systematisch zu erfassen und anschaulich-grafisch auszuwerten.

B. Ergänzungen

B.1. Zellsimulator

Der Zellsimulator Abb. B.1 besteht aus einer Reihenschaltung von sechs Widerständen und ermöglicht es, durch Anlegen einer Spannung über die äußeren Pins, sechs Spannungen zu simulieren. Mit einer Rohde & Schwarz DC Power Supply lassen sich Spannungen zwischen $0\,\mathrm{V}$ und $4\,\mathrm{V}$ einstellen.

Über einen Frequenzgenerator kann durch eine DC-Offset behaftete Dreieck- oder Sägezahnspannung und einer Frequenzanregung im mHz Bereich, das Spannungsverhalten von Batterien ansatzweise simuliert werden. Dieser Aufbau wurde allerdings nur im 1-Zell-Messbetrieb verwendet, da die maximale Ausgangsspannung des Frequenzgenerator einen Mehrzellenaufbau nicht zulässt.

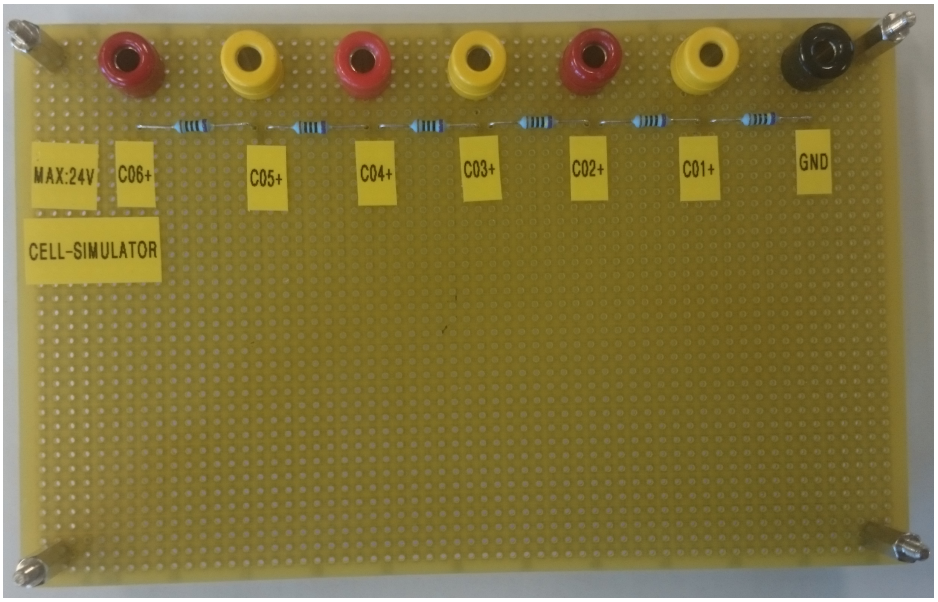


Abbildung B.1.: Der Zellsimulator

B.2. Messaufbau zur Bestimmung des Frequenzganges

An einem Notebook mit der Messsoftware (PULSE-Labshop), wurde über die Ethernet-Schnittstelle, das PULSE-Messsystem angeschlossen (gelb). An einem Ausgang des PULSE-Messystem wurde über BNC-Kabel ein Verstärker (grün) angeschlossen. Dieser wurde wiederum mit dem Shaker/Vibration Exciter (hellblau) verbunden. Über den Shaker lässt sich das DUT, in diesem Fall der gebaute ADXL335 Beschleunigungsaufnehmer, in Richtung der Z-Achse mit unterschiedlichen Frequenzen anregen. Zur Bestimmung des Frequenzganges wurde der Shaker mit einem Sinus-Sweep von 1 bis $400\,\mathrm{Hz}$ angeregt. In pink ist der Referenzbeschleunigungsaufnehmer zu sehen. Die verwendeten Geräte sind in der Tabelle B.1 aufgeführt.

Tabelle B.1.: Geräteliste des Messaufbaues

Bezeichnung	Typennummer	Hersteller	Farbe in Abb. B.2
Power Amplifier	Type 2706	Brüel & Kjær	grün
Vibration Exciter	Type 4809	Brüel & Kjær	blau
Accelerometer	Type 4397	Brüel & Kjær	pink
Schwingungs-kalibrator	Type 4294	Brüel & Kjær	n. a.
PULSE-Messystem	Type 3560-C	Brüel & Kjær	gelb
Notebook mit PULSE-Labshop	compaq nc6000	HP Windows XP	n. a.
Beschleunigungs-aufnehmer (DUT)	ADXL335 [3]	ANALOG-DEVICES	rot

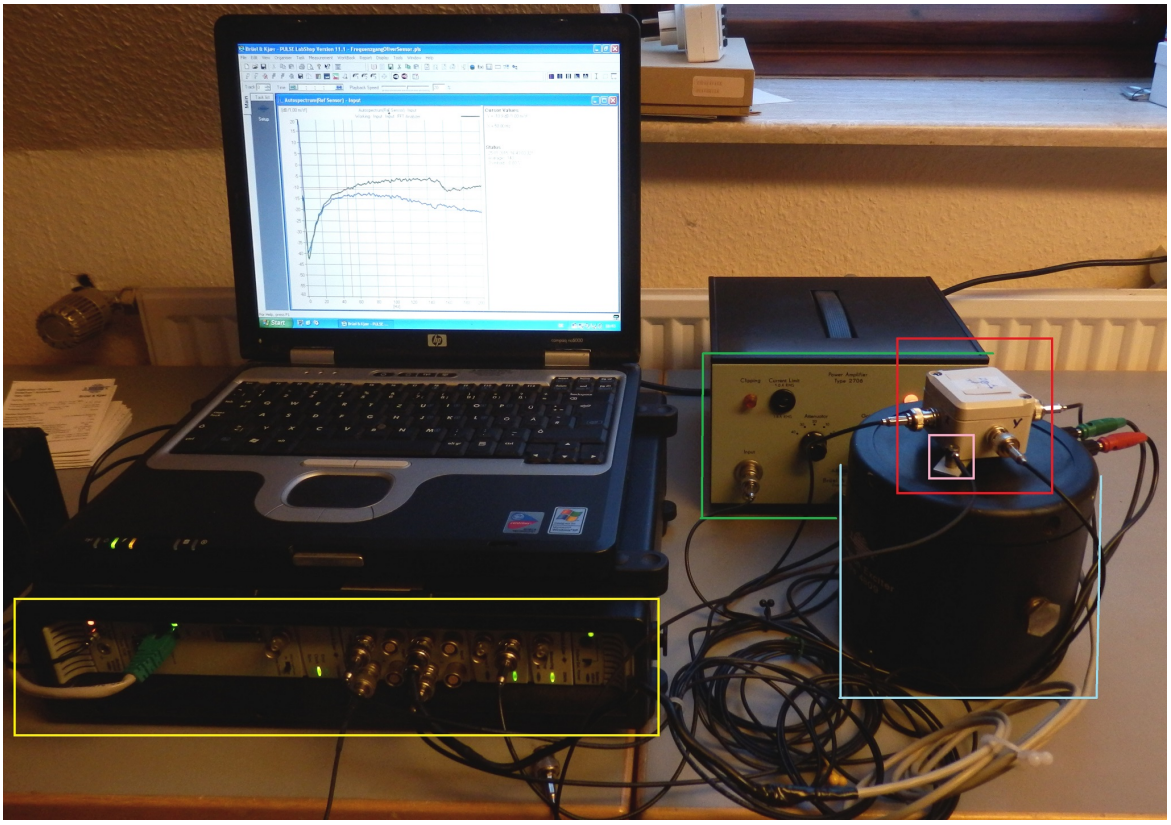


Abbildung B.2.: Messaufbau zur Bestimmung des Frequenzganges der Z-Achse.

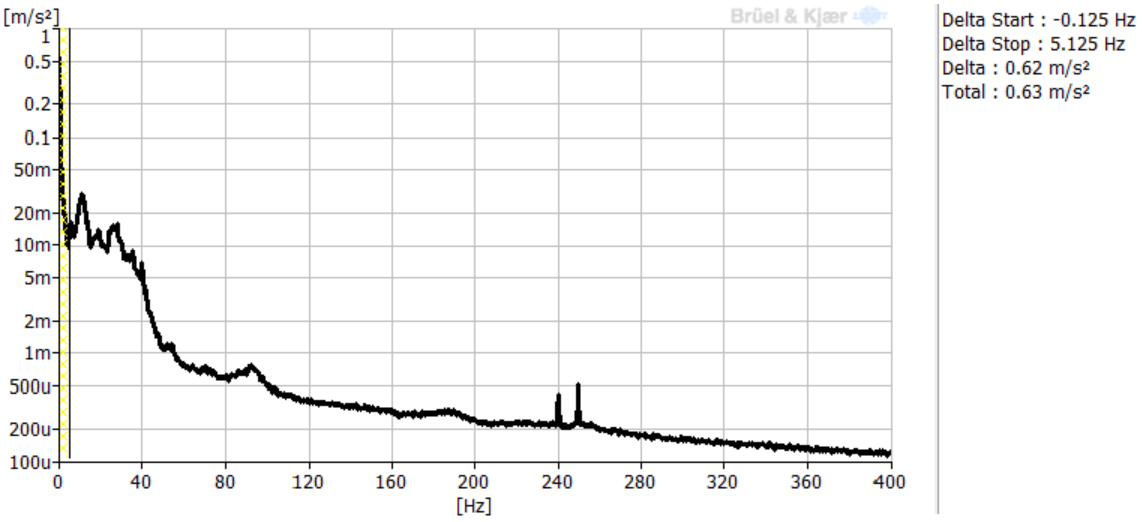


Abbildung B.3.: Zeitlich gemitteltes Effektivwertspektrum der Messfahrt für den Frequenzbereich von 0 bis 400 Hz.

B.3. Prinzipschaltbild EDDA

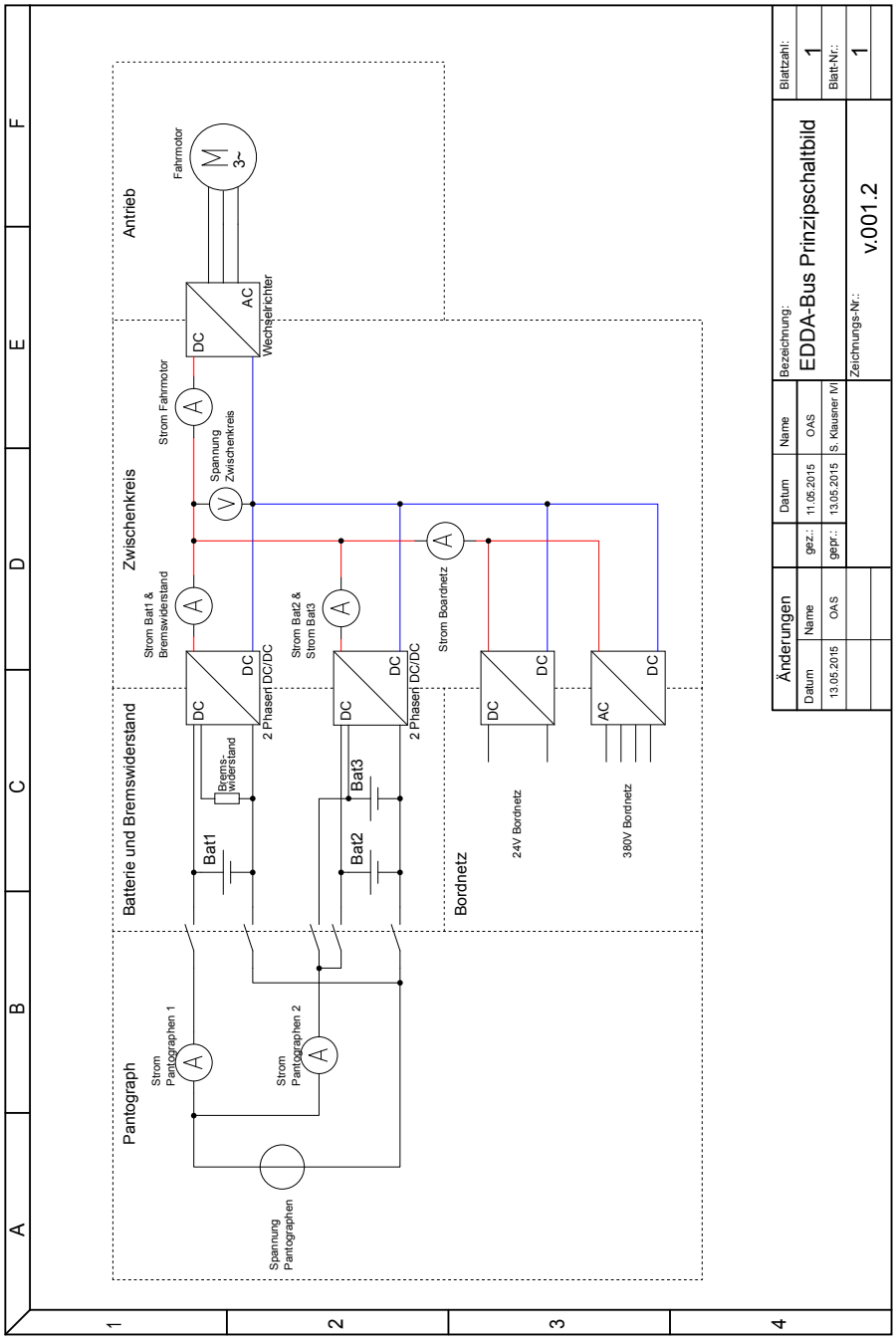
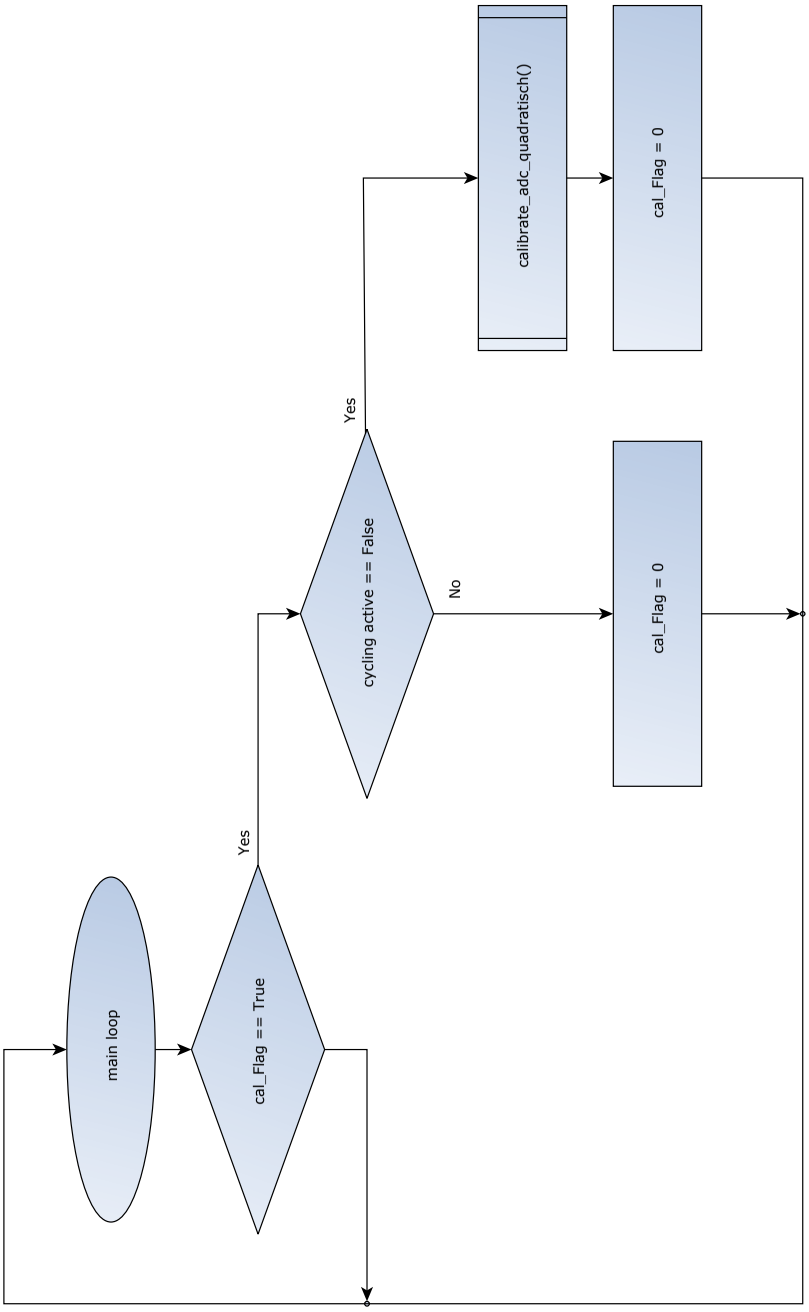
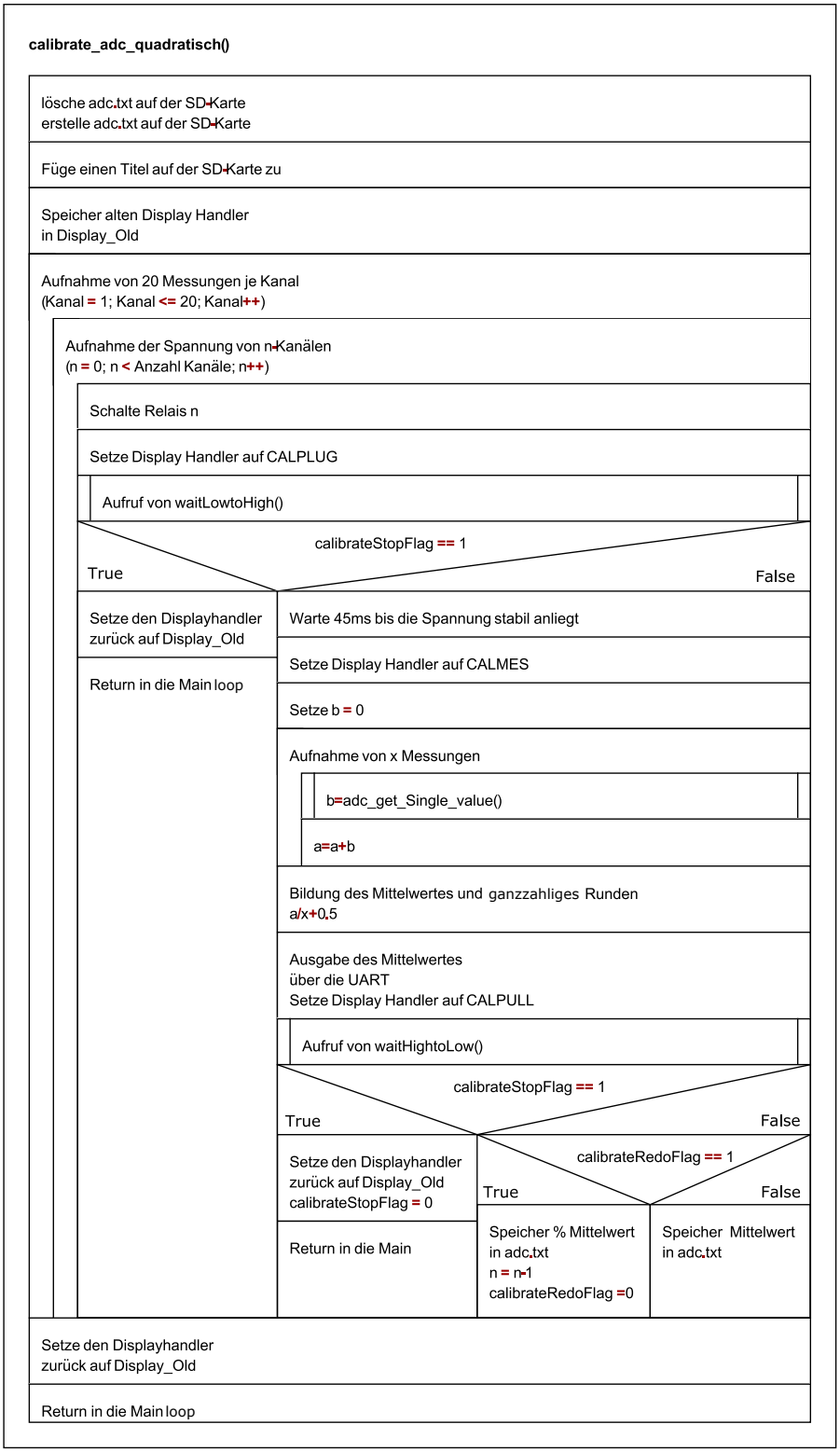


Abbildung B.4.: 1Zyklus1ganz

B.4. Flussdiagramm des Aufrufs der Kalibrierfunktion



B.5. Nassi-Shneiderman-Diagramm der Kalibrierfunktion



Beispiel einer *adc.txt* nach einer Kalibrierung des Spannungskanal 1. des Zykliergerät R

Listing B.1: adc.txt

```
%Calibration File:
%Ch1 Ch2 Ch3 Ch4 Ch5 Ch6 Ch7 Ch8 Ch9 Ch10 Ch11 Ch12
00012439
00014693
00016934
00019174
00021412
00023626
00025897
00028085
00030408
00032576
00034829
00037052
00039263
00041513
00043723
00045990
00048296
00050415
00052630
00054881
```

Beispiel einer *config.txt* mit einer Schleifen-Verschachtelungstiefe von drei.

Listing B.2: config.txt

```
//Zyklierplan fuer das ZG1 Proto3Schmidt
quantity 01
cyclestep 500

meastime 99998
maxcycletime 99998

//Datum: 30-Apr-2015
coefficient A 01 3204FA
coefficient B 01 4A85875
coefficient C 01 199FEA9
```

```
12
13 ip 192.168.000.128
14
15 //Grenzwerte zur Sicherheit
16 vlimits 2500 3900
17 maxcurr 30000
18 maxtemp 35
19 maxcharge 99999
20
21 //Zyklierablauf
22 relsel 1 wait 60 0    // 60sek Warten
23 relsel 2 loop_start 10 0 // 10 Widerholungen (10 Tage)
24 relsel 3 loop_start 24 0 // 24 Widerholungen (24 h = 1 Tag)
25 relsel 4 loop_start 30 0 // 30 Widerholungen (0,5 h)
26 relsel 6 wait 60 1    // 60sek Warten Relais 1 angezogen
27 relsel 7 loop_end 0 0
28 relsel 8 loop_start 30 0 // 30 Widerholungen (0,5 h)
29 relsel 9 wait 60 2    // 60sek Warten Relais 2 angezogen
30 relsel 10 loop_end 0 0
31 relsel 11 loop_end 0 0
32 relsel 12 loop_end 0 0
```

B.6. Fehlersuche am Zykliergerät

Während der Arbeit wurde eine umfangreiche Fehlersuche an dem Zykliergerät durchgeführt. In diesem Abschnitt werden die Fehler übersichtsartig aufgelistet.

- Kalibrierwerte werden nicht von der 'config.txt' geladen.
  - Eine Kalibrierung des Zykliergerätes über die SD-Karte ist somit nicht möglich.
  - Es stellte sich heraus, dass die Umstellung von der linearen Kalibrierung auf die quadratische Kalibrierung nicht vollständig implementiert war. Variablennamen waren unvollständig oder falsch zugeordnet.
- Über- und Unterspannungen, zu hohe Ströme bzw. zu hohe Temperaturen werden zu spät registriert.
  - Die Überschreitungen von Sicherheitsparametern werden im worst case erst nach einer Minute registriert. Dies liegt an den zu großen Mess- und Zyklierintervallen.
  - Die Messraten wurden auf $0{,}5\,\mathrm{s}$ und die Zyklierrate auf $1\,\mathrm{s}$ angepasst. Die Sicherheitsschwelle wurde so definiert, dass ein Überschreitungen von Sicherheitsparametern für $1{,}5\,\mathrm{s}$ anliegen muss, damit das Gerät die Zyklierung unterbricht.

Dies war notwendig, um beim Auftreten eines Messfehlers nicht die gesamte Zyklierung zu unterbrechen.

- Hardware-Probleme
  - Durch kalte Lötstellen und schlechte Crimpungen wurden immer wieder Wackelkontakte festgestellt, die sowohl die ADC Messungen beeinflussten, als auch zu Kontaktproblemen der Display- und Button-Verbindungen führten.
  - Es erfolgte eine Überarbeitung der Lötstellen und Crimpungen.
- ADC-Problematiken: Störungen, Messabbrüche, Über- bzw. Unterspannungen ließen das Zykliergerät unregelmäßig abstürzen. Dies führte zu einer Reihe Änderungen in Hard- und Software. Diese werden nachfolgend kurz aufgelistet.
  - Der Spannungsregler auf dem externen ADC-Board ist unterdimensioniert.
    - Die $3{,}3\,\mathrm{V}$ Versorgungsspannung des ADC's und die des SPI-Optokopplers wurde über einen Spannungsregler versorgt. Bei einer Messung mit einem Oszilloskop wurde festgestellt, das bei einer Messung des ADC diese Spannung kurzzeitig einbricht.
    - Mit einen $470\,\mu\mathrm{F}$ Kondensator wurde die Spannungsversorgung vorläufig stabilisiert.
  - Der externe Tiefpass und der Instrumentenverstärker auf dem externen ADC-Board wurden durch Jumper ausgekoppelt, um Messwertverfälschungen vorzubeugen.
  - Die Anbindung des externen ADC-Board über Flachbandleitungen an den Mikrocontroller war sehr fehleranfällig und musste mehrfach nachgelötet bzw. erneuert werden.
  - Die 'myadc.c' und 'myadc.h' wurden zugunsten einer besseren Lesbarkeit (weniger Magic-Numbers) umstrukturiert und neu implementiert. Es wurde darauf geachtet, das die Funktionen nur mit genau definierten Datentypen aufgerufen werden können. Des Weiteren wurden Funktionen auf 'static' gesetzt um eine Kapselung der Funktionen zu erreichen.
  - Fehlerhafter SPI-Clock
    - Mit Hilfe eines Logic Analyzer vom Typ. Saleae16 wurde zum Einschaltzeitpunkt des Zykliergerätes die SPI-Schnittstelle überwacht, da vermutet wurde, dass die Kommunikation mit dem externen ADC Typ: AD7799 nicht richtig funktionierte. Der ADC kommuniziert über die SPI-Schnittstelle mit einer Taktfrequenz von $2\,\mathrm{MHz}$. Zusätzlich wird über die SPI-Schnittstelle noch mit

der SD-Karte kommuniziert. Dies geschieht mit einer anderen Frequenz von $12\,\mathrm{MHz}$. Bei den Untersuchungen ergab sich, dass der ADC teilweise bei der Initialisierung mit einer Frequenz von $12\,\mathrm{MHz}$ angesprochen wird. Dies führte zu fehlerhaften Initialisierungen.

  - Durch die Erweiterung einer SSI-BUSY Abfrage in der 'myssi.c' konnte erreicht werden, dass die Initialisierung ordnungsgemäß durchgeführt werden konnte. Des Weiteren wurde die Fehler-Flags des ADC ausgewertet und Warnungen über die UART-Schnittstelle ausgegeben.

- Das externe Board für den ADC lässt sich nicht spannungsfrei schalten, da es direkt an die $230\,\mathrm{V}$ geschlossen ist.

  - Bei einem Neuaufbau des Zykliergerätes R wurde das externe ADC-Board nicht wieder verwendet.

C. Abbildungen

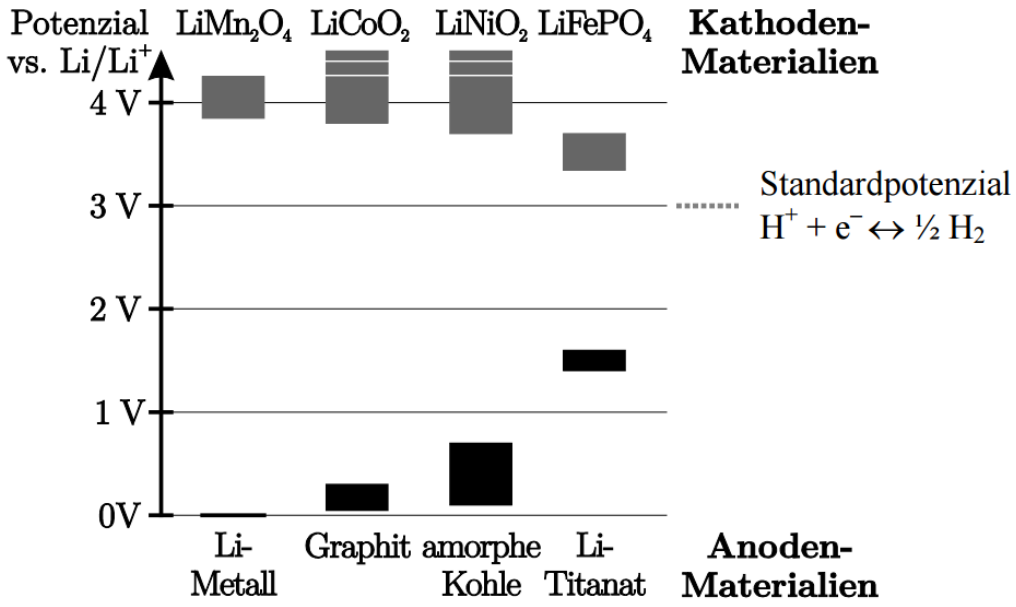


Abbildung C.1.: Potenzialbereiche verschiedener Elektrodenmaterialien für Lithium-Ionen Zellen [32]

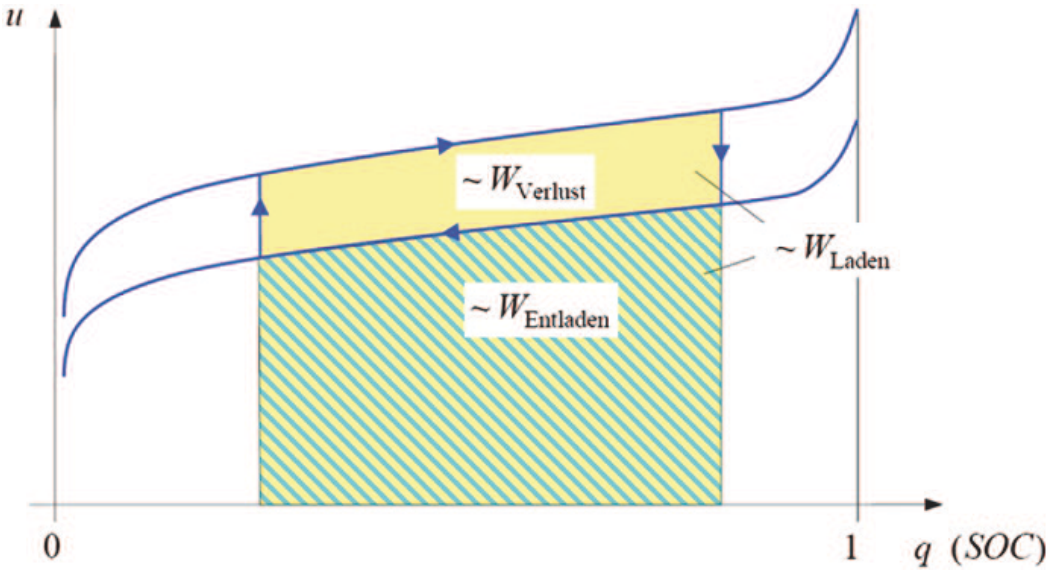


Abbildung C.2.: Speicherwirkungsgrad einer Lithium-Ionen-Batterie über den Spannungsunterschied bei Ladung und Entladung [18].

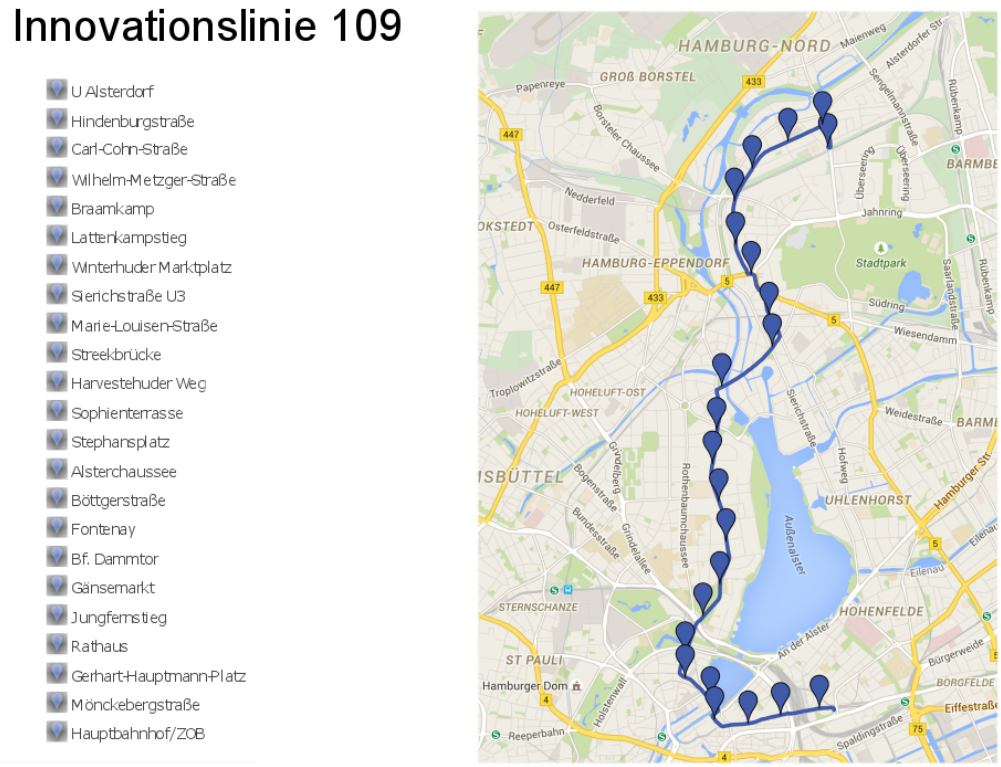


Abbildung C.3.: Route der Innovationslinie 109

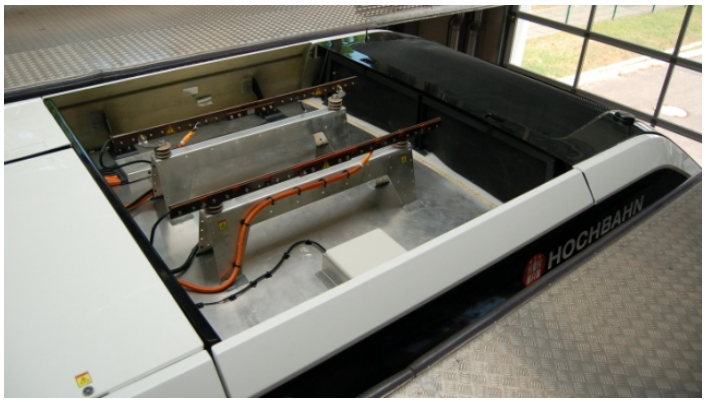


Abbildung C.4.: Kontakschine auf dem Bus für die Ladung der Batterie [17]

Abbildung C.5.: Ladestation an der Endhaltestelle

Abbildung C.6.: Ladeanzeige in der Konsole des Volvo 7900 Electric Hybrid

Abbildung C.7.: Der Volvo 7900 Electric Hybrid an der Endhaltestelle Hauptbahnhof/ZOB

Abbildung C.8.: Der Volvo 7900 articulated Hybrid (Gelenkbus) an der Endhaltestelle Hauptbahnhof/ZOB

Abbildung C.9.: Der EDDA-Bus des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme [13]

Abbildung C.10.: Der EDDA-Bus des Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme an der Ladestation [19]

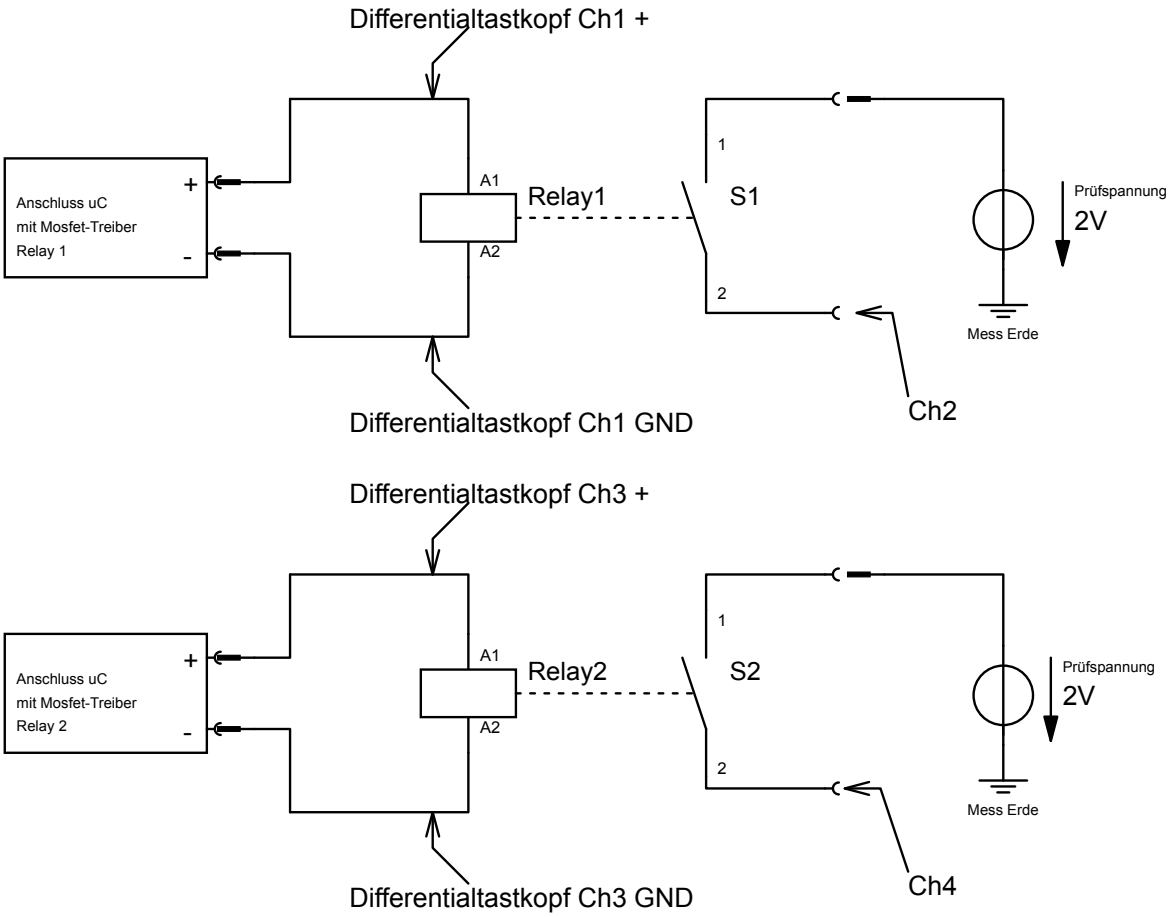


Abbildung C.11.: Beispielhafte Abbildung einer Schaltzeiten Messung an einem Lastpfad. Ch1 und Ch3 wurden mit je einem Differentialtastkopf aufgenommen, da keinen gemeinsamen Massepunkt in der Mosfet Schaltung gab. Ch2 und Ch4 wurden gegen die *Mess Erde* gemessen.

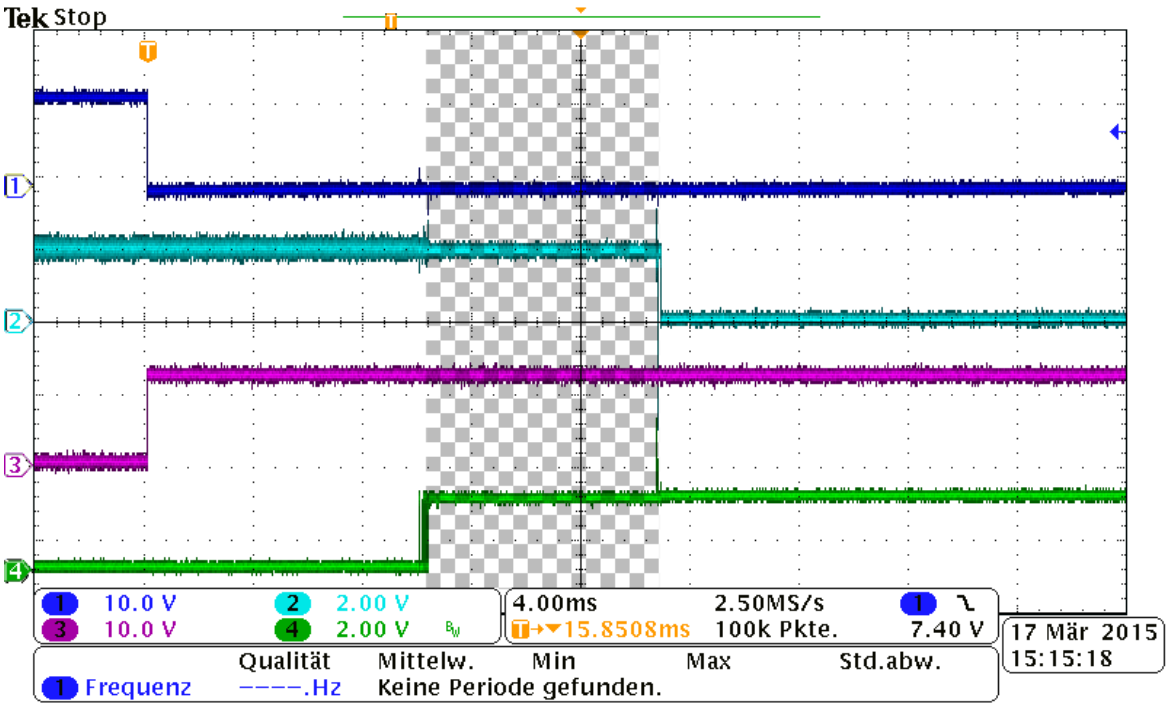


Abbildung C.12.: Messung der Abfallzeit des Relay 2 und Anzugszeit des Relay 3, wobei Ch1 die Steuerspannung des Relay 2, Ch2 die Lastspannung des Relay 2, Ch3 die Steuerspannung des Relay 3 und Ch4 die Lastspannung des Relay 3 darstellt. Der karierte Bereich stellt die Überschneidung der Leistungspfade da.

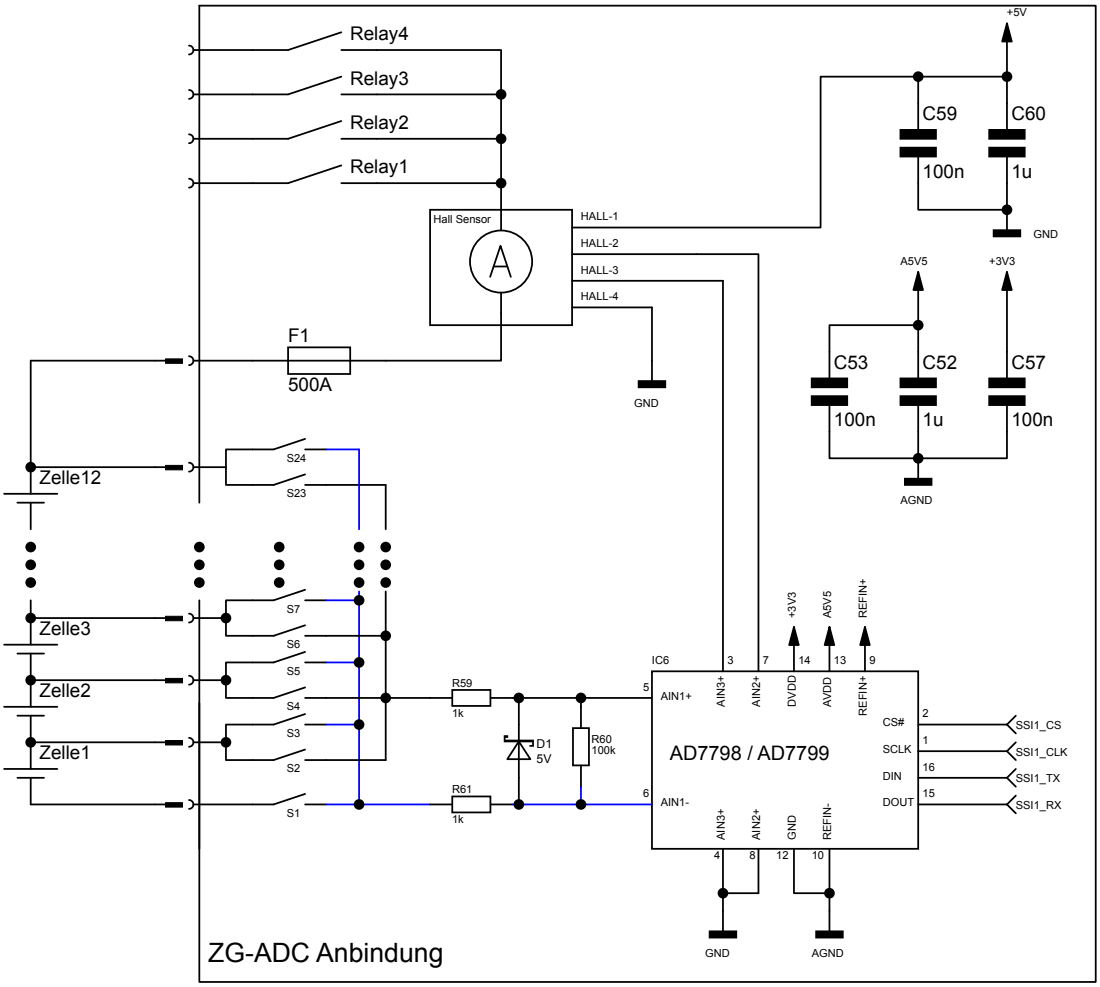


Abbildung C.13.: Schaltplan zur Veranschaulichung der ADC-Anbindung mit implementierter Schutzschaltung. Basiert auf dem Schaltplan V2.0 [44].

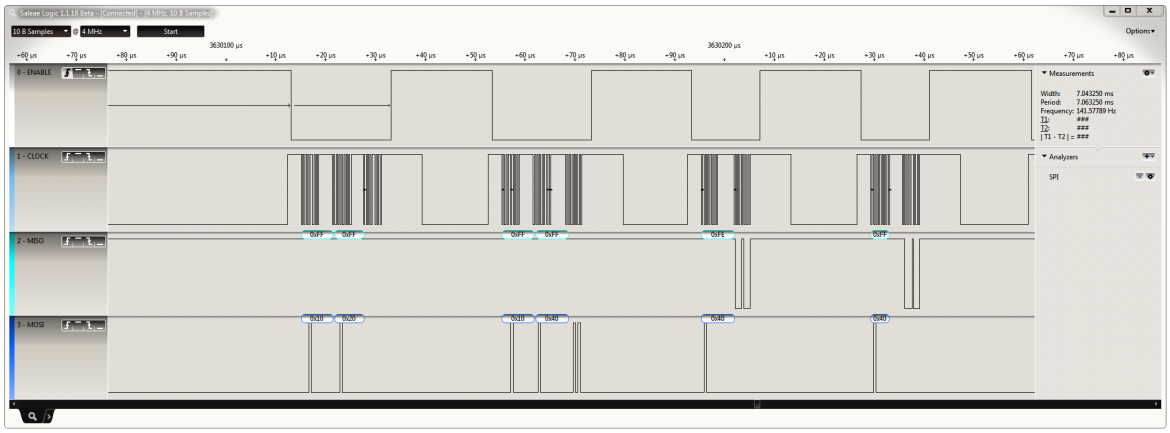


Abbildung C.14.: Beginn einer Messung mit dem externen ADC

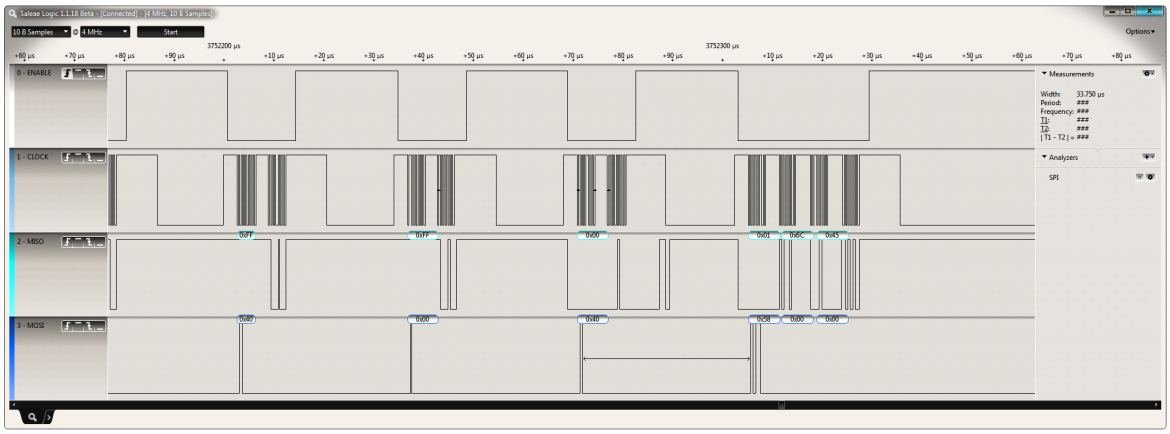


Abbildung C.15.: Ende einer Messung mit dem externen ADC

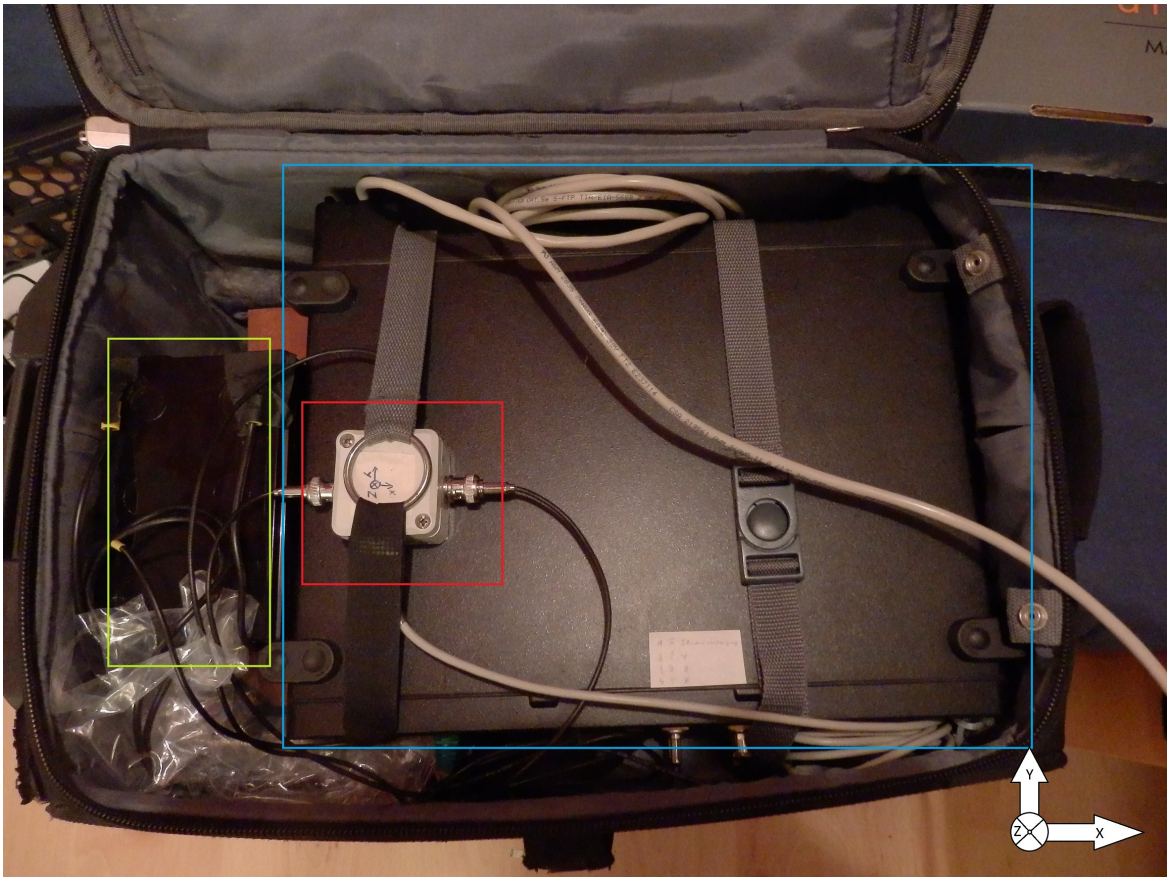


Abbildung C.16.: Abbildung des PULSE-Messsystem (blau), dem Beschleunigungssensor (rot) und einer 12V Batterie als Spannungsversorgung (grün), in einem Mess-Koffer. Über eine Ethernet-Verbindung wurde per Notebook mit dem Messsystem kommuniziert.

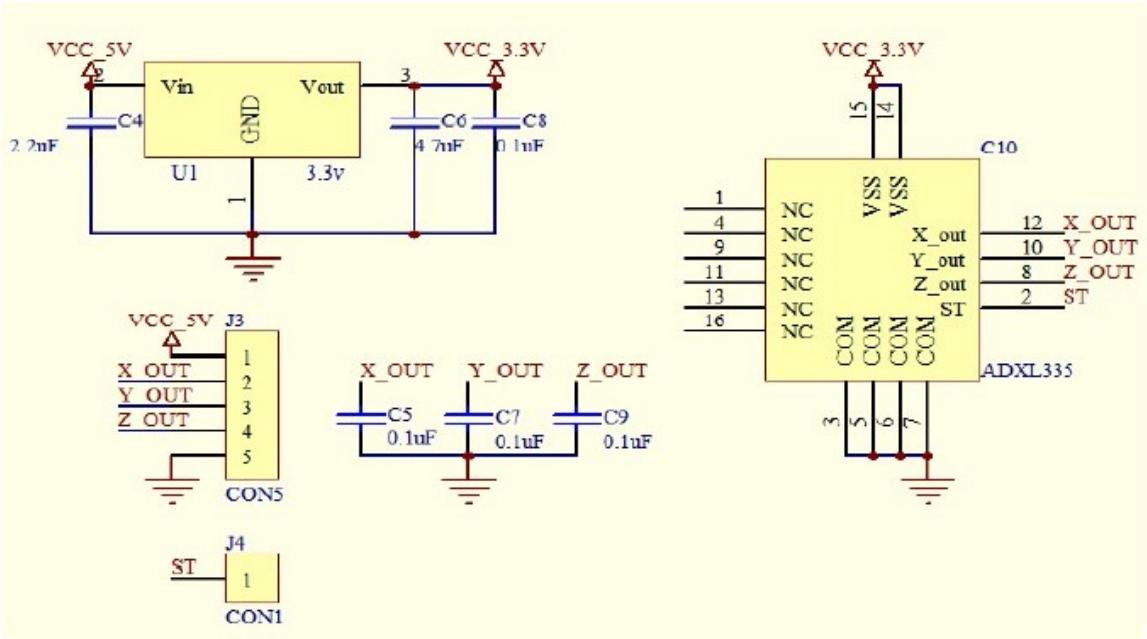


Abbildung C.17.: Abbildung der externen Beschaltung des Beschleunigungssensors ADXL335 [23]

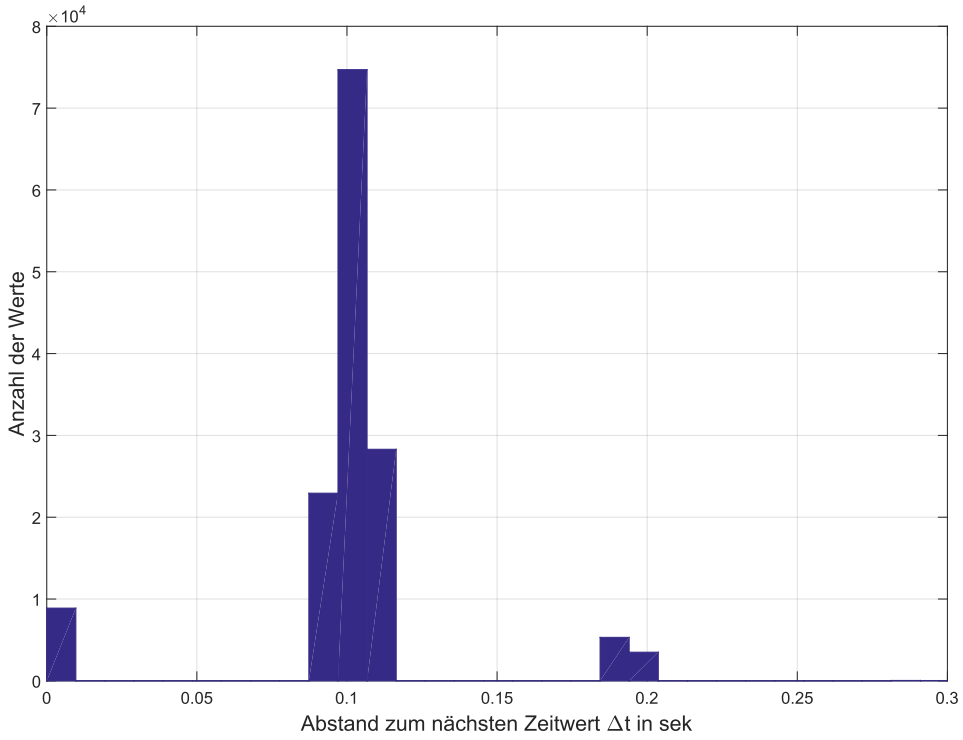


Abbildung C.18.: Dieses Histogramm zeigt die Differenz $\Delta t_n = t_{n+1} - t_n$ zum nächsten Zeit-Wert bei dem Zeitvektor der EDDA-Busdaten.

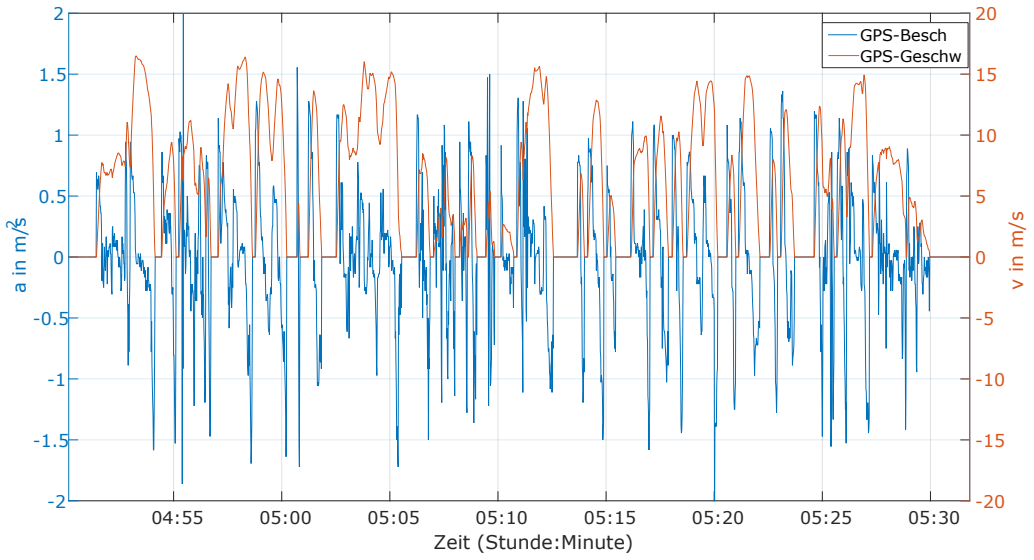


Abbildung C.19.: Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die aus der Geschwindigkeit abgeleitete Beschleunigung.

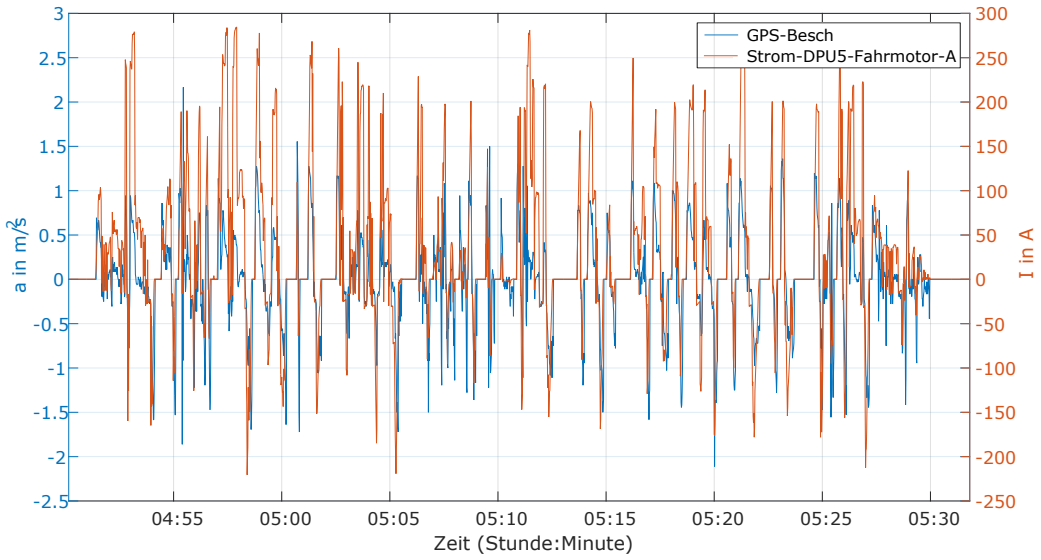


Abbildung C.20.: Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die aus der Geschwindigkeit abgeleitete Beschleunigung und der in den Fahrmotor fließende Strom.

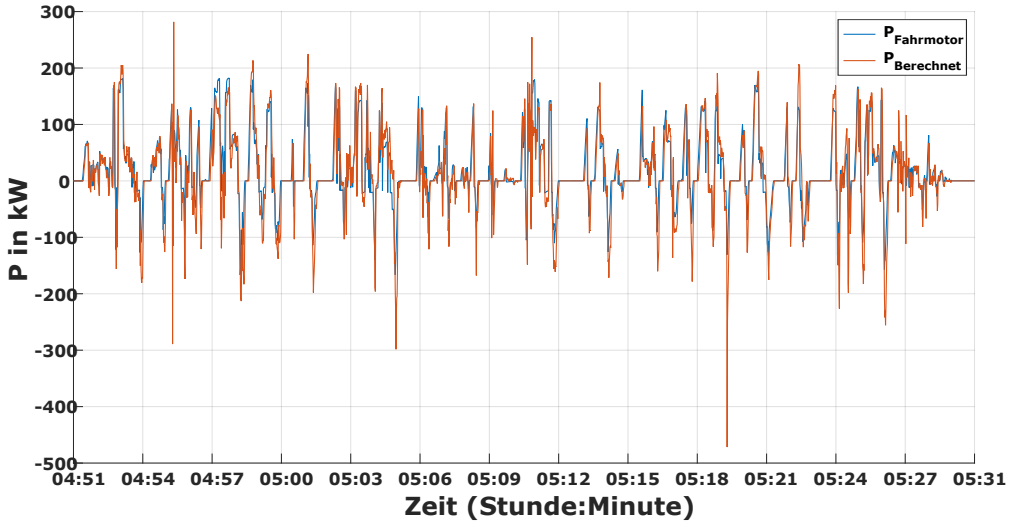


Abbildung C.21.: Die Abbildung zeigt die Fahrt des EDDA-Busses. Zu sehen ist die Tatsächliche Leistung und die aus dem Modell errechnete Leistung des Busses.

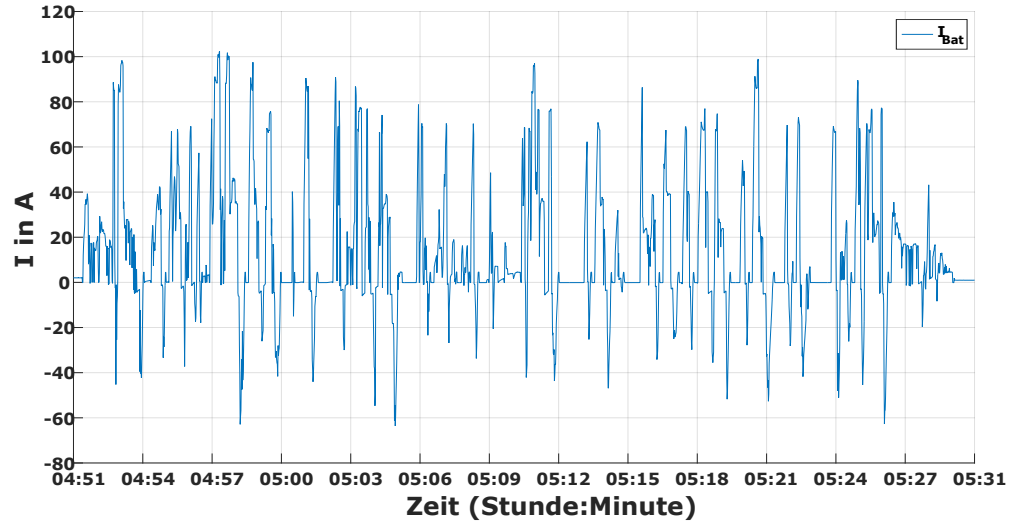


Abbildung C.22.: Der Strome in die Batterie des EDDA Busses.

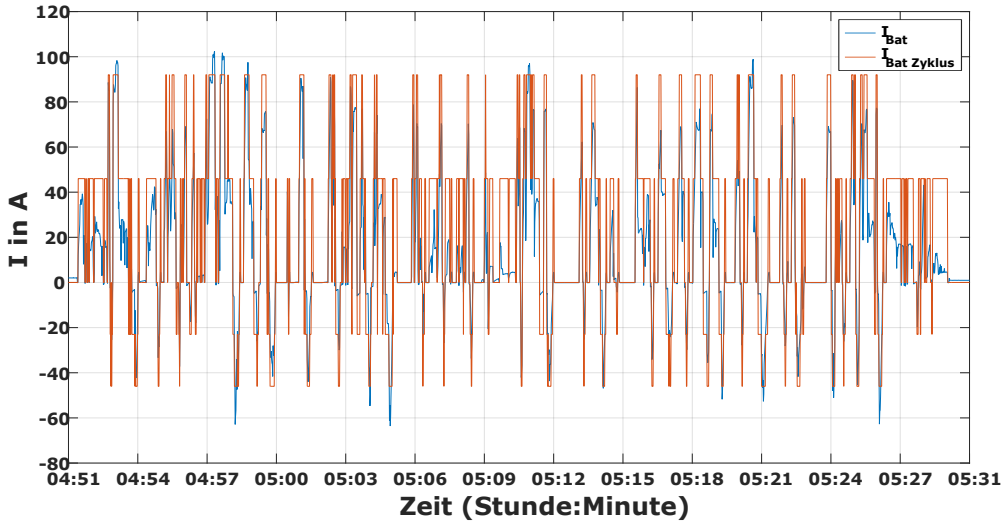


Abbildung C.23.: Abbildung des EDDA-Stromstufen für den Zyklus. Positiver Strom entspricht einem Beschleunigungsvorgang und ein negativer Strom einem Bremsvorgang (Rekuperation) Die Stromstufen sind Farbig dargestellt.

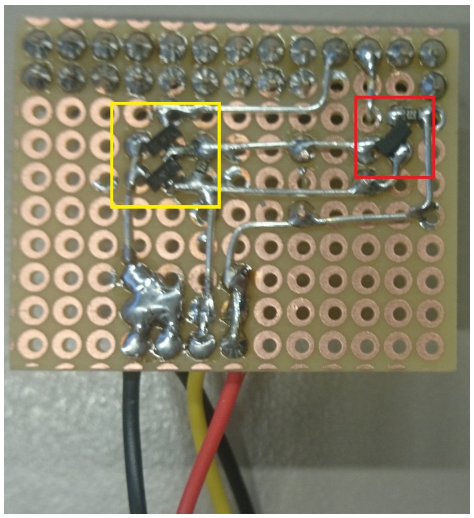


Abbildung C.24.: MOSFET-Schaltung für den 1-Wire Bus provisorisch auf Lochrasterplatine. In rot der Pull-Up-MOSFET und in gelb die Eingangsstufe (open drain) der UART-TxD

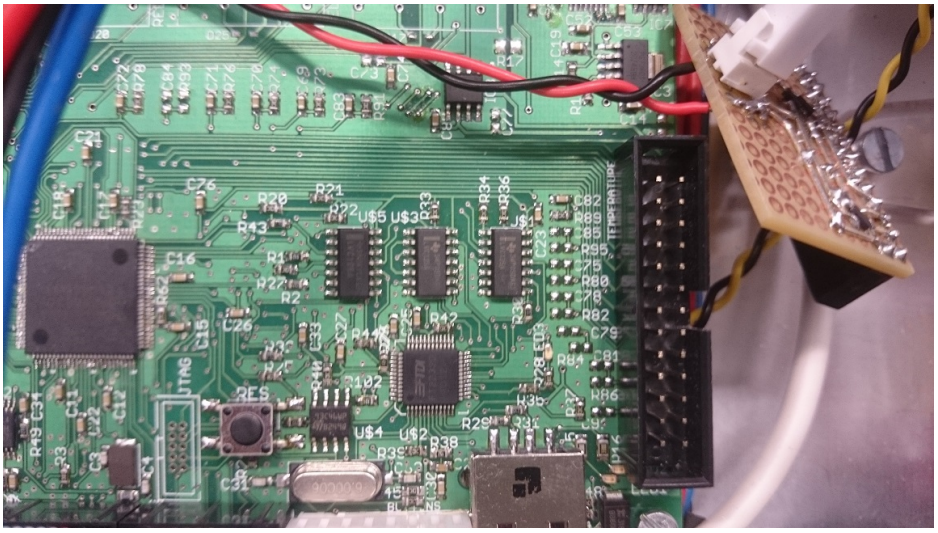


Abbildung C.25.: 1-Wire Buchsenverbindung zur ARM-Platine

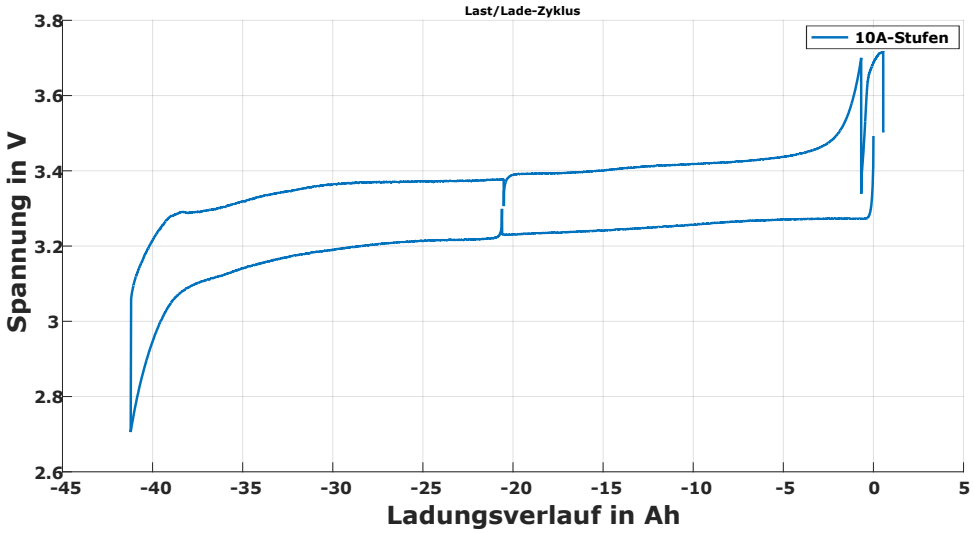


Abbildung C.26.: Zyklierung der ECC-Battetrien bei verschiedenen Konstantströmen.

D. MATLAB-Skripte

Quadratische Kalibrierung

MATLAB-Skript D.1: QuadratischeKalibrierung.m

```
% Kalibrierung eines Spannungskanals des ZG
% Author: Oliver Schmidt
%% Import
[file,path]=uigetfile('*.txt');
[Ch1] = ImportCaliFile([path,file],3, 22); %unschön!

%% Messwerte vom Tektronix händisch eingetragen

%18.03.2015 Messung kombiFileFertig
    Tektronix = [1.00074 1.17447 1.35004 1.52546 1.70071 1.87570 2.0491 2.2259 2.4068 2.5699 2.7492 2.9193 3.1018 3.2771 3.4524 3.6259
        3.8018 3.9778 4.1549 4.3246];
% original    :   3.952590e-12,    7.801922e-05,    2.306038e-02
% bearbeitet :   3.952590e+06,    7.801922e+07,    2.306038e+07
% Hex         :         3C4FCE,          4A67A90,          15FDFA0
%
%
% //Datum: 18-Mar-2015
% coefficient A 01 3C4FCE
% coefficient B 01 4A67A90
% coefficient C 01 15FDFA0

Tektronix = Tektronix';
figure(1)
plot(Ch1,Tektronix,'*')
title('Plot des 1Ch')
hleg=legend('Ch1','location','NorthEastOutside');
set(hleg);
xlabel('n Werte ADC')
ylabel('Spannung Tektronix in uV')
grid on;
%%
figure(2)
Koeff = polyfit(Ch1,Tektronix,2);
x= 0:1:Ch1(length(Tektronix));
Funk=Koeff(1)*x.*x+Koeff(2)*x+Koeff(3);
plot(Ch1,Tektronix,'*','LineWidth',2)
hold on
plot(x,Funk,'LineWidth',2)
xlabel('Gemessene ADC-digits')
ylabel('Spannung Tektronix in uV')
grid on;
legend('Gemesse Wertepaare','Quadratische Ausgleichsfunktion','location','northwest')
set(gca,'FontSize',25,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',25,'fontWeight','bold')
hold off

%Koeff in den Positiven bereich
Koeff_ADC(1)=Koeff(1)*1e18;
Koeff_ADC(2)=Koeff(2)*1e12;
Koeff_ADC(3)=Koeff(3)*1e9;

%Koeff_ADC_Hex(1)=
Koeff_ADC_1=dec2hex(round(Koeff_ADC(1)));
Koeff_ADC_2=dec2hex(round(Koeff_ADC(2)));
Koeff_ADC_3=dec2hex(round(Koeff_ADC(3)));

hex2dec(Koeff_ADC_1)

%Ausgabe der Koeffizienten zum kopieren in die config.txt
fprintf('original    :%15d, %15d, %15d\n\n',Koeff(1),Koeff(2),Koeff(3))
```

```
fprintf('Export      :%15d, %15d, %15d\n',Koeff_ADC(1),Koeff_ADC(2),Koeff_ADC(3))
fprintf('Hex         :%15s, %15s, %15s\n\n\n',Koeff_ADC_1,Koeff_ADC_2,Koeff_ADC_3)

fprintf('//Datum: %s\n',date)
fprintf('coefficient A 01 %s\n',Koeff_ADC_1)
fprintf('coefficient B 01 %s\n',Koeff_ADC_2)
fprintf('coefficient C 01 %s\n',Koeff_ADC_3)


%% Test Messung nach der Kalibrierung %18.03.2015 Messung 1V-4.3V
% Spannung Tektronix und die Spannung des ZG
Tek_M = [0 0.25209 0.50652 0.74577 0.99996 1.25460 1.50764 1.75186 2.0042 2.2496 2.5015 2.7519 3.0013 3.2497 3.5007
         3.7481 4.0000 4.2517 4.5039 4.7454 4.9943]
ZG_M  = [0 0.253953 0.508180 0.746638 1.000871 1.254797 1.507946 1.751639 2.004404 2.248958 2.501650 2.751372 3.001722 3.249332
       3.501253 3.748082 4.000166 4.251625 4.503949 4.745445 4.994239]

format short
n= 0:1:length(Tek_M)-1;

diff=0;
diff = (Tek_M-ZG_M)*1000; % umrechung in mV

figure(4)
subplot(2,1,1)
plot(n,Tek_M,'r*','LineWidth',2)
hold on
plot(n,ZG_M,'bo','LineWidth',2)
grid on
title('Vergleichsmessung Zykliergerät mit Tektronix')
xlabel('n Messungen')
ylabel('U in V')
legend('Messwert Tektronix','Messwert Zykliergerät','location','northwest')
set(gca,'FontSize',25,'fontWeight','bold')

set(findall(gcf,'type','text'),'FontSize',25,'fontWeight','bold')
hold off

subplot(2,1,2)
%plot(0:5/20:5,diff,'o')
plot(n,diff,'+','LineWidth',2)
grid on;
title('Berechnetes \DeltaU vom Zykliergerät und Tektronix')
xlabel('n Messungen')
ylabel('\DeltaU in mV')
legend('\DeltaU = U_{Tektronix} - U_{Messung}','location','northwest')


set(gca,'FontSize',25,'fontWeight','bold')

set(findall(gcf,'type','text'),'FontSize',25,'fontWeight','bold')
```

Quadratische Kalibrierung

MATLAB-Skript D.2: AuswertungMessdatenZykliergeeat.m

```
%% Skript zum einlesen und darstellen der Messdaten für einen Kanal des ZG.
% Author: Oliver Schmidt
% das datetime Format kann nur mit  >=MATLAB R2014b+ genutzt werden
%Einlesen der Daten
Menge_Daten = 0;
ZG = 2
[file,path]=uigetfile('C:\BATSEN1\Messungen\*.txt');
liste = dir([path '*.txt']);
%% n x Einlesen
x=1;
while x <= length(liste)

Menge_Daten = Menge_Daten+1;
fprintf('Starte mit %s\n',liste(x).name)
data = csvread([path,liste(x).name],130,0);


time=datetime('now');% only can use with MATLAB R2014b+
voltage=0;
temperature=0;
current=0;
relay=0;
```

```
23
24 time(1:length(data)) = datetime('now');
25 voltage(1:length(data)) = 0;
26 temperature(1:length(data)) = 0;
27 current(1:length(data)) = 0;
28 relay(1:length(data)) = 0;
29
30 % für die Prozent Ausgabe
31 P_1 = round(length(data)/100);
32 P_Count = 1;
33 fprintf('0')
34
35 % only with MATLAB R2014b+
36 for count_line = 1:1:length(data)
37     k=count_line;
38     if count_line == 1
39         sek = data(k,7);
40     elseif data(k,7) == data(k-1,7)
41         sek = data(k,7)+0.5;
42     else
43         sek = data(k,7);
44     end
45     time(k) = datetime(data(k,2),data(k,3),data(k,4),data(k,5),data(k,6),sek);
46     voltage(k) = data(k,8);
47     temperature(k) = data(k,9)/10;
48     current(k) = data(k,10);
49     relay(k) = data(k,11);
50
51     % Prozent Ausgabe
52     if (count_line == P_1 * P_Count)
53        %fprintf('%d',P_Count)
54        if P_Count == 25
55            fprintf('25')
56        elseif P_Count == 50
57            fprintf('50')
58        elseif P_Count == 75
59            fprintf('75')
60        elseif P_Count == 100
61            fprintf('100\n')
62        else
63            fprintf('-')
64        end
65        P_Count=P_Count+1;
66     end
67
68 end
69
70 % Haöö-effekt ist falsch herum eingebaut im ZG 1
71 if (ZG ==1)
72 current = -current;
73 end
74 % Ah berechnung
75 tdiffS=seconds(time-datetime(data(k,2),data(k,3),data(k,4)));
76
77 voltage = voltage/1000000;  % voltage in muV
78 current = current/1000;     % in mA
79 SOC=cumtrapz(tdiffS,current)/3600; %
80
81 count2=1;
82 ruhe=0;
83 %ermittlung der Ruhephasen
84 for count1 = 1:length(relay)-1
85     %[row(count),col(count),v(count)]=find(relay1,0);
86    if relay(count1)==0;
87        if (relay(count1+1) ~= 0)
88            count2 = count2+1;
89            ruhe(count2) = count1;
90        elseif (count1+1==length(relay))
91            count2 = count2+1;
92            ruhe(count2) = count1;
93        end
94    end
95 end
96 % Speichern der Messdaten in einem struct
97 structM(Menge_Daten) = struct('file',liste(x).name,'time',time,'timeS',tdiffS,'voltage',voltage,'temperature',temperature,'current',
       current,'relay',relay,'SOC',SOC,'ruhe',ruhe);
98 fprintf('fertig!\n')
99 x = x+1;
100 end
101 fprintf('Alle fertig!\n')
102
103
104 %% Plott der Messdaten
105 Menge_Daten = length(structM);
106 for Plot_Daten = 1:1:Menge_Daten
```

```
107
108 Fcount=5;
109 figure(Plot_Daten)
110
111 h(1)=subplot(Fcount,1,1);
112 plot(structM(Plot_Daten).time,structM(Plot_Daten).voltage)
113 title('Spannung Ch1')
114 xlabel('time')
115 ylabel('Spannung in V')
116 grid on
117
118
119 h(2)=subplot(Fcount,1,2);
120 plot(structM(Plot_Daten).time,structM(Plot_Daten).current)
121 title('Current')
122 xlabel('time')
123 ylabel('Current in A')
124 grid on
125
126
127 h(3)=subplot(Fcount,1,3);
128 plot(structM(Plot_Daten).time,structM(Plot_Daten).SOC)
129 title('Ladung')
130 xlabel('time')
131 ylabel('Ladungsverlauf in Ah')
132 grid on
133
134 h(4)=subplot(Fcount,1,4);
135 plot(structM(Plot_Daten).time,smooth(structM(Plot_Daten).temperature,100,'moving'))
136 title('Temperature')
137 xlabel('time')
138 ylabel('Temperature in °C')
139 grid on
140
141 h(5)=subplot(Fcount,1,5);
142 plot(structM(Plot_Daten).time,structM(Plot_Daten).relay)
143 title('Powerd Relay')
144 xlabel('time')
145 ylabel('Relay no.')
146 grid on
147
148 linkaxes(h,'x');
149
150 end
151 %% SOC
152
153 Fcount=1;
154 figure(Menge_Daten+1)
155 hold on
156 for Plot_Daten = 1:1:Menge_Daten
157
158     plot(structM(Plot_Daten).SOC,structM(Plot_Daten).voltage,'LineWidth',1.2)
159     title('Last/Lade-Zyklus')
160     xlabel('Ladungsverlauf in Ah')
161     ylabel('Spannung in V')
162     grid on
163 end
164
165 set(gca,'FontSize',20,'fontWeight','bold')
166
167 set(findall(gcf,'type','text'),'FontSize',20,'fontWeight','bold')
168
169 legend(structM(1).file,'location','northwest')
170
171
172
173 %% Zusammenfügen von Daten
174 Ziel   = 3;
175 Erster  = 1;
176 Zweiter = 2;
177
178 structM(Ziel).time        = [structM(Erster).time        structM(Zweiter).time];
179
180
181 structM(Ziel).voltage     = [structM(Erster).voltage     structM(Zweiter).voltage];
182 structM(Ziel).temperature = [structM(Erster).temperature structM(Zweiter).temperature];
183 structM(Ziel).current     = [structM(Erster).current     structM(Zweiter).current];
184 structM(Ziel).relay       = [structM(Erster).relay       structM(Zweiter).relay];
185 structM(Ziel).SOC         = [structM(Erster).SOC         structM(Zweiter).SOC];
186
187
188 structM(Ziel).tdiffS=seconds(structM(Ziel).time-datetime(data(k,2),data(k,3),data(k,4)));
189 structM(Ziel).SOC=cumtrapz(structM(Ziel).tdiffS,structM(Ziel).current)/3600;
190
191 Ziel = 11
```

```
count2=1;
ruhe=0;
%ermittlung der Ruhephasen
for count1 = 1:length(structM(Ziel).relay)-1

    if structM(Ziel).relay(count1)==0;
        if (structM(Ziel).relay(count1+1) ~= 0)
            count2 = count2+1;
            structM(Ziel).ruhe(count2) = count1;
            fprintf('fertig!%d\n',count1)
        elseif (count1+1==length(structM(Ziel).relay))
            count2 = count2+1;
            structM(Ziel).ruhe(count2) = count1;
            fprintf('blub!%d\n',count1)
        end
    end
end

%% Plott der Messdaten Lino Teil1
%% Plot 5  LINO
  Plot_Daten = 5;
  xmin = datenum(2015,5,16,04,0,01)
  xmax = datenum(2015,5,16,14,0,00)
   a=10;

Fcount=4;
figure(Plot_Daten)

h(1)=subplot(Fcount,1,1);
plot(structM(Plot_Daten).time,structM(Plot_Daten).voltage)

title('Spannung')
xlabel('Zeit (Stunde:Minute)')
ylabel('Spannung in V')
grid on
xlim([xmin xmax])

xData = linspace(xmin,xmax-datenum(seconds(0)),a);

ax = gca;
ax.XTick = xData;
datetick('x','HH:MM','keepticks')
yData = linspace(2.50,3.70,4);
ax.YTick = yData;
ylim([2.50 3.70])


set(gca,'FontSize',14,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',16,'fontWeight','bold')


h(2)=subplot(Fcount,1,2);
plot(structM(Plot_Daten).time,structM(Plot_Daten).current)
title('Srom')
xlabel('Zeit (Stunde:Minute)')
ylabel('Current in A')
grid on
xlim([xmin xmax])

xData = linspace(xmin,xmax-datenum(seconds(0)),a);
ax = gca;
ax.XTick = xData;
datetick('x','HH:MM','keepticks')
yData = linspace(-30,20,6);
ax.YTick = yData;

ylim([-30 20])
set(gca,'FontSize',14,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',16,'fontWeight','bold')

h(3)=subplot(Fcount,1,3);
plot(structM(Plot_Daten).time,structM(Plot_Daten).SOC)
title('Ladung')
xlabel('Zeit (Stunde:Minute)')
ylabel({'Ladungsverlauf','in Ah'})
grid on
xlim([xmin xmax])
xData = linspace(xmin,xmax-datenum(seconds(0)),a);
ax = gca;
ax.XTick = xData;
yData = linspace(-15,0,6);
ax.YTick = yData;
ylim([-15 0])

datetick('x','HH:MM','keepticks')
```

```
set(gca,'FontSize',14,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',16,'fontWeight','bold')

h(4)=subplot(Fcount,1,4);
plot(structM(Plot_Daten).time,smooth(structM(Plot_Daten).temperature,100,'moving'))
title('Temperatur')
xlabel('Zeit (Stunde:Minute)')
ylabel('Temperature in °C')
grid on
xlim([xmin xmax])
xData = linspace(xmin,xmax-datenum(seconds(0)),a);
ax = gca;
ax.XTick = xData;
datetick('x','HH:MM','keepticks')
yData = linspace(20,25,6);
ax.YTick = yData;

set(gca,'FontSize',14,'fontWeight','bold')

set(findall(gcf,'type','text'),'FontSize',16,'fontWeight','bold')

linkaxes(h,'x');
```

Fahrverhalten EDDA

MATLAB-Skript D.3: FahverhaltenEDDA.m

```
% Simulation des Fahrverhaltens
% Author: Oliver Schmidt
%% Bus EDDA

% Simulationsparameter EDDA
v = 0:0.0001:42;
h = 3.360;
b = 2.250;
l = 12.0;
f_r = 0.01;
m_f = 18000;%29000;
g = 9.81;
al = 0; %Steigung
c_w = 0.9;
A_x = h*b;
T = 8;
p = 1021;
p_L = 1.293 * 273 /(T+273) * p/1013
W = 0.9
m_Res = 1*m_f;
%%
figure(200)


F_W = c_w * A_x * p_L / 2 * (v.*v); % Luft
F_R(1:1:length(v)) = f_r * m_f * g * cos(al); % Rollwiderstand
F_H(1:1:length(v)) = m_f * g * sin(al); % Hangabtriebskraft

hold on
for a = 3 : -1 : -3
F_B(1:1:length(v)) = m_Res * a; % Bschleunigungswiderstand
F_gesWiderstand= F_R + F_W + F_H + F_B;
P_R = bsxfun(@times , F_gesWiderstand , v);
plot(v*3.6,F_gesWiderstand/1000,'LineWidth',2)
grid on
ylabel('${F \rm \ in \ kN}$','Interpreter','latex')
xlabel('$v \rm \ in \ \frac{km}{h}$','Interpreter','latex')
ylabel('F in kN')
xlabel('v in km/h')
end
xlim([0 80])
hold off
legend('Zugkraft für eine Beschleunigung von 3 m/s^2',...
    'Zugkraft für eine Beschleunigung von 2 m/s^2',...
    'Zugkraft für eine Beschleunigung von 1 m/s^2',...
    'Zugkraft für eine Beschleunigung von 0 m/s^2',...
    'Zugkraft für eine Beschleunigung von -1 m/s^2',...
    'Zugkraft für eine Beschleunigung von -2 m/s^2',...
    'Zugkraft für eine Beschleunigung von -3 m/s^2','Location','northwest')
legend('boxoff')
```

```
set(gca,'FontSize',20,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',20,'fontWeight','bold')
%% Radleistung
figure(201)
hold on
for  a = 3 : -1 :-3
F_B(1:1:length(v)) = m_Res * (a); % Bschleunigungswiderstand
F_gesWiderstand= F_R + F_W + F_H + F_B;
plot(v*3.6,(F_R.*v+F_W.*v+F_H.*v+F_B.*v)/1000,'LineWidth',2)
grid on
%title('Zugkraft Geschwindigkeitsdiagramm','FontSize',18)
%legend('Rollwiderstand*v','Aerodynamischer Widerstand*v','Hangabtriebskraft*v',...
legend('Radleistung für eine Beschleunigung von 3 m/s^2',...
    'Radleistung für eine Beschleunigung von  2 m/s^2',...
    'Radleistung für eine Beschleunigung von  1 m/s^2',...
    'Radleistung für eine Beschleunigung von  0 m/s^2',...
    'Radleistung für eine Beschleunigung von -1 m/s^2',...
    'Radleistung für eine Beschleunigung von -2 m/s^2',...
    'Radleistung für eine Beschleunigung von -3 m/s^2','Location','northwest')
legend('boxoff')
ylabel('P_{R} in kW')
xlabel('v in km/h')
%ylabel('$P \rm  \ in \ kW$','Interpreter','latex')
%xlabel('$v \rm  \ in \ \frac{km}{h}$','Interpreter','latex')
end
xlim([0 80])
ylim([-800 800])
set(gca,'FontSize',20,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',20,'fontWeight','bold')
hold off

%% Leistung über Beschleunigung
a = -3:0.0001:3;
F_W=0
F_R=0
F_H=0
F_B_Neu=0
figure(205)
hold on
for v = [0, 5 ,10 ,20 ,50 ,80]
    v=v/3.6
    F_W = c_w * A_x * p_L / 2 * (v*v); % Luft
    F_R(1:1:length(a)) = f_r * m_f * g * cos(al); % Rollwiderstand
    F_H(1:1:length(a)) = m_f * g * sin(al); % Hangabtriebskraft
    F_B_Neu(1:1:length(a)) = m_Res * (a); % Bschleunigungswiderstand
    plot(a,(F_R.*v+F_W.*v+F_H.*v+F_B_Neu.*v)/1000,'LineWidth',2)
end
grid on
legend('Radleistung für eine konstante Geschwindigkeit von von 0 km/h',...
    'Radleistung für eine konstante Geschwindigkeit von von 5 km/h',...
    'Radleistung für eine konstante Geschwindigkeit von von 10 km/h',...
    'Radleistung für eine konstante Geschwindigkeit von von 20 km/h',...
    'Radleistung für eine konstante Geschwindigkeit von von 50 km/h',...
    'Radleistung für eine konstante Geschwindigkeit von von 80 km/h','Location','northwest')
legend('boxoff')
ylabel('P_{R} in kW')
xlabel('a in m/s^2')
set(gca,'FontSize',20,'fontWeight','bold')
set(findall(gcf,'type','text'),'FontSize',20,'fontWeight','bold')
hold off
```

E. Quellcode Zykliergerät

Quellcode E.1: clocktimer.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          clocktimer.c
4
5  Auhtor:         Thomas Wisnewski
6  Credits:        Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Johannes Roehn
10          Oliver Schmidt
11          Stellaris Ware
12
13 last modified:      2015/03/19
14
15 Project Status      Under Construction
16 Status:         running
17
18 CCS:         5.5.1.00031
19 Stellarisware:       8555
20
21 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
22          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
23          NTC-Connectors, MAX3232 and Suplly Circuits
24
25 Description:       Source File for Timer
26
27 */
28
29 //*****************************************************************************
30 //
31 // Includings
32 //
33 //*****************************************************************************/
34 #include <string.h>
35 #include "driverlib/rom.h"
36 #include "third_party/fatfs/src/ff.h"
37 #include "third_party/fatfs/src/diskio.h"
38 #include "inc/hw_ints.h"
39 #include "inc/hw_memmap.h"
40 #include "inc/hw_types.h"
41 #include "driverlib/debug.h"
42 #include "driverlib/gpio.h"
43 #include "driverlib/interrupt.h"
44 #include "driverlib/pin_map.h"
45 #include "driverlib/rom.h"
46 #include "driverlib/sysctl.h"
47 #include "driverlib/timer.h"
48 #include <stdio.h>
49
50 //*****************************************************************************
51 //
52 // Own Includings
53 //
54 //*****************************************************************************/
55 #include "utils/uartstdio.h"
56 #include "header/mysdcard.h"
57 #include "header/config.h"
58 #include "header/clocktimer.h"
59 #include "header/myadc.h"
60 #include "header/temperature.h"
61 #include "header/ethernet.h"
62 #include "header/control.h"
63 #include "header/relais.h"
64 #include "header/display.h"
65
66
```

```
67  // Global variables for systemclock data
68  volatile unsigned long clock_msec = 00;
69  volatile unsigned long clock_sec = 00;
70  volatile unsigned long clock_min = 00;
71  volatile unsigned long clock_hour = 18;
72  volatile unsigned long clock_day = 06;
73  volatile unsigned long clock_month = 03;
74  volatile unsigned long clock_year = 2013;
75
76  // Global variables for loadrelay timing control
77  volatile unsigned int timer2second = 0;
78  volatile unsigned int power_relay_active = 0;
79  volatile unsigned int cycle_active = false;
80  volatile unsigned int measurement_active = false;
81
82  static unsigned int loopStart[LOOPDEPTH];
83  static unsigned int loopParam[LOOPDEPTH];
84  static unsigned char loopLevel = 0;
85
86  // Global variable for cycle control
87  volatile unsigned int current_cycle_step = 1;
88  volatile unsigned int wait_start_time = 0;
89  //Minute counter for maximum cycle time
90  volatile int max_time_counter = 0;
91  //global variable for battery charge
92  volatile int gBat_charge = 0;
93
94
95  //*********external variables**************
96
97  // TCP-Socket für die aktuelle Steuerungs-Verbindung
98  extern struct tcp_pcb* control_connection;
99
100 //Feed watchdog variable
101 extern volatile tBoolean g_bFeedWatchdog;
102
103 //defines the string variables for cycling types
104 extern const char *cycle_types[];
105
106 //Variable for the current
107 extern volatile int g_iCurrent;
108 //global variable for the current cell voltages
109 extern volatile unsigned int g_iCellVoltages[12];
110 //global variable for the current cell voltages
111 extern volatile int g_iCellTemperature[12];
112
113 //Flag for Display ReINIT
114 extern volatile int g_bDispInit;
115
116 //global variable for the calibration
117 extern volatile char g_Flag_calibrationModeOn;
118
119 //*****************************************************************************
120 //Extern variable for name of file for measurment saving
121 //*****************************************************************************
122 extern char MEAS_FILE[128];
123
124
125 //*****************************************************************************
126 // Set up and enable the timers.  Configure a timer
127 // for the measure interrupt handler.
128 // Additionally set up Timer for display refreshment at 15Hz.
129 // and minute counter.
130 //*****************************************************************************
131 void init_clock_timer(){
132
133  UARTprintf("Initializing Timer...");
134
135         //
136         // Enable the peripherals used by this example.
137         //
138   //Timer for Measurment-Interrupt
139         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
140         //Timer for LED-Display refreshment
141         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
142         //Timer for loadrelay timing control
143         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
144         //Timer for toggle cycle active LED
145         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER3);
146
147         //
148         // Configure the 32-bit periodic timers.
149         //
150         //Configure Timer0A for defined time "Cyclestep" from config
151         TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
```

```
152 //          TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() * config.cyclestep); // bei Cyclestep=1 Messung alle 1sek
153         TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()/1000 * config.cyclestep); // bei Cyclestep=1 Messung alle 0,5sek
154
155         //Configure Timer1A at 15 Hz
156         TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
157         TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/15); //hier gucken
158
159         //configure Timer2A interrupt for every minute
160         TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
161 //          TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()*60);
162         TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()); //configure Timer2A interrupt for every 1sek
163
164
165         //configure Timer3A as 32 Bit Timer for LED toggling every 500ms
166         TimerConfigure(TIMER3_BASE, TIMER_CFG_PERIODIC);
167         TimerLoadSet(TIMER3_BASE, TIMER_A, SysCtlClockGet()/2);
168
169
170
171         //
172         // Enables Trigger for internal ADCs
173         //
174         TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
175
176         //
177         // Setup the interrupt for the timer timeout.
178         //
179         IntEnable(INT_TIMER0A);
180         TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
181
182         IntEnable(INT_TIMER1A);
183         TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
184
185         IntEnable(INT_TIMER2A);
186         TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
187
188         IntEnable(INT_TIMER3A);
189         TimerIntEnable(TIMER3_BASE, TIMER_TIMA_TIMEOUT);
190
191     UARTprintf("done\n");
192 }
193
194 //*****************************************************************************
195 // Function to reinitialize the Timer0A when variable "Cyclestep" reconfigured
196 //*****************************************************************************
197 void reinit_timer0(void){
198  TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()/1000 * config.cyclestep);
199 }
200
201
202 //*****************************************************************************
203 //
204 // The interrupt handler for the periodical measurement. Called depended on
205 // configured "Cyclestep".
206 //
207 //*****************************************************************************
208 void Timer0IntHandler(void)
209 {
210
211  //
212     // Clear the timer interrupt.
213     //
214     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
215
216  // Do the measurement
217  do_measure();
218
219
220 }
221
222 //*****************************************************************************
223 //
224 // The interrupt handler for switching power relays and automatic stop of measurement.
225 // Called every second.
226 //
227 //*****************************************************************************
228 void Timer2IntHandler(void)
229 {
230  int i;
231
232
233  //Increment minute counter
234  timer2second++;
235
236     if(timer2second == config.measuretime)//Stop measurment and cycling if measuretime reached, If measuretime = 0 INFINITE MEASUREMENT
```

```
237     Cmd_stop(0,NULL);
238
239     if(cycle_active){//follow cycle steps if cycling is active
240
241      //CYCLING STATEMACHINE
242    switch(config.cycle_steps[current_cycle_step].type){
243
244    case WAIT:{
245
246     if(timer2second >= (wait_start_time + config.cycle_steps[current_cycle_step].param)){//check if relative waiting time reached
247      current_cycle_step++;//next step
248      if(current_cycle_step == (config.step_quantity + 1))//if step quantity exceeded stop cycling
249       stop_cycling();
250      else{//SWITCH POWERRELAY for the next step
251       power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
252       wait_start_time = timer2second;//set the waiting start time for the wait step
253      }
254     }
255     break;
256    }
257    case CHARGE_CELLV:{
258
259     for(i = 0; i < config.quantity_cells; i++){//check if one of the cells has reached charge end voltage
260      if(g_iCellVoltages[i] >= (config.cycle_steps[current_cycle_step].param * 1000)){//muV
261       current_cycle_step++;
262       if(current_cycle_step == (config.step_quantity + 1))//if step quantity exceeded stop cycling
263        stop_cycling();
264       else{//SWITCH POWERRELAY for the next step
265        power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
266        wait_start_time = timer2second;//set the waiting start time for the wait step
267       }
268       break;
269      }
270     }
271     break;
272    }
273    case DISCHARGE_CELLV:{
274
275     for(i = 0; i < config.quantity_cells; i++){//check if one of the cells has reached min voltage
276      if(g_iCellVoltages[i] <= (config.cycle_steps[current_cycle_step].param * 1000)){//muV
277       current_cycle_step++;
278       if(current_cycle_step == (config.step_quantity + 1))//if step quantity exceeded stop cycling
279        stop_cycling();
280       else{//SWITCH POWERRELAY for the next step
281        power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
282       wait_start_time = timer2second;//set the waiting start time for the wait step
283       }
284       break;
285      }
286     }
287     break;
288    }
289    case CHARGE_BATV:{
290
291     int bat_voltage = 0;
292     for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
293      bat_voltage += g_iCellVoltages[i];
294     //check if battery voltage has reached charge end voltage
295     if(bat_voltage >= (config.cycle_steps[current_cycle_step].param * 1000)){//muV
296      current_cycle_step++;
297      if(current_cycle_step == (config.step_quantity + 1))//if step quantity exceeded stop cycling
298       stop_cycling();
299      else{//SWITCH POWERRELAY for the next step
300       power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
301       wait_start_time = timer2second; //set the waiting start time for the wait step
302      }
303     }
304     break;
305    }
306    case DISCHARGE_BATV:{
307
308     int bat_voltage = 0;
309     for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
310      bat_voltage += g_iCellVoltages[i];
311     //check if battery voltage has reached discharge end voltage
312     if(bat_voltage <= (config.cycle_steps[current_cycle_step].param * 1000)){//muV
313      current_cycle_step++;
314      if(current_cycle_step == config.step_quantity)//if step quantity exceeded stop cycling
315       stop_cycling();
316      else{//SWITCH POWERRELAY for the next step
317       power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
318       wait_start_time = timer2second;//set the waiting start time for the wait step
319      }
320     }
321     break;
```

```
322     }
323     case LOOP_START:{
324
325      if(loopLevel >= LOOPDEPTH){
326       stop_cycling();
327       break;
328      }
329      loopParam[++loopLevel-1]=config.cycle_steps[current_cycle_step].param;
330      loopStart[loopLevel-1] = current_cycle_step++;
331
332      if(current_cycle_step >= (config.step_quantity + 1))//if step quantity exceeded stop cycling
333       stop_cycling();
334      else{//SWITCH POWERRELAY for the next step
335       power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
336       wait_start_time = timer2second;//set the waiting start time for the wait step
337      }
338      timer2second--;
339      Timer2IntHandler();
340     }break;
341
342     case LOOP_END:{
343
344      if(loopLevel == 0){
345       stop_cycling();
346       break;
347      }
348
349      if( --loopParam[loopLevel-1] > 0){
350       current_cycle_step = loopStart[loopLevel-1]+1;
351
352      }
353      else{
354       loopLevel--;
355       current_cycle_step++;
356      }
357
358      if(current_cycle_step >= (config.step_quantity + 1))//if step quantity exceeded stop cycling
359       stop_cycling();
360      else{//SWITCH POWERRELAY for the next step
361       power_relay_active = switch_power_relais(config.cycle_steps[current_cycle_step].relay);
362       wait_start_time = timer2second;//set the waiting start time for the wait step
363      }
364      timer2second--;
365      Timer2IntHandler();
366
367     }break;
368
369     default:{//default statement
370      stop_cycling();//safety turn off
371     }
372    }
373      }
374      // Clear the timer interrupt.
375      //
376      TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
377 }
378
379
380 //*****************************************************************************
381 // The interrupt handler for the LED toggling if cycling is active
382 //*****************************************************************************
383 void Timer3IntHandler(void){
384  //
385      // Clear the timer interrupt.
386      //
387      TimerIntClear(TIMER3_BASE, TIMER_TIMA_TIMEOUT);
388
389    if(cycle_active)
390    //Toggle cycle acitve LED
391    GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4, ~GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_4));
392    else
393    GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4, 0);
394 }
395
396
397
398 //*****************************************************************************
399 // Update the Systemclock and toggle cycle LED + check of maximum cycle time
400 //*****************************************************************************
401 void TimerClock(void)
402 {
403   clock_msec += 10;
404    if(clock_msec == 1000){
405     clock_msec = 0;
406     clock_sec++;
```

```
407   }
408
409       if (clock_sec == 60){
410        clock_sec = 0;
411        clock_min++;
412
413        if(cycle_active){
414         max_time_counter++;
415         if(max_time_counter == config.max_cycle_time){//If absolute maximum measurement time reached shut down relays and cycling
416          stop_cycling();
417          handlerState = ERRTIME;//Display Error Message
418          UARTprintf("\nMAXIMUM CYCLE TIME EXCEEDED!\n");
419       if(control_connection)
420        telnet_write("\nMAXIMUM CYCLE TIME EXCEEDED!\n");
421         }
422        }
423
424        //raise Flag for Display ReINIT every minute
425        g_bDispInit = 1;
426       }
427
428       if(clock_min == 60){
429        clock_hour++;
430        clock_min = 0;
431       }
432       if(clock_hour == 24){
433        clock_day++;
434        clock_hour = 0;
435       }
436
437       if((clock_month == 1) || (clock_month == 3) || (clock_month == 5) || (clock_month == 7) || (clock_month == 8) || (clock_month ==
              10) || (clock_month == 12)){
438        if(clock_day == 32) {
439         clock_month++;
440         clock_day = 1;
441        }
442       }
443       if((clock_month == 4) || (clock_month == 6) || (clock_month == 9) || (clock_month == 11)){
444        if(clock_day == 31) {
445         clock_month++;
446         clock_day = 1;
447        }
448       }
449
450       if(clock_month == 2){
451        int d_temp = 29;
452        if(clock_year%4 == 0)
453         d_temp = 30;
454        if(clock_year%4 != 0)
455         d_temp = 29;
456        if(clock_day == d_temp) {
457         clock_month++;
458         clock_day = 1;
459        }
460       }
461
462       if(clock_month == 13){
463        clock_year++;
464        clock_month = 1;
465       }
466   }
467
468   //*****************************************************************************
469   //
470   // This function implements the cycling safety switch off, measurement continues
471   //
472   //*****************************************************************************
473   void stop_cycling(void)
474   {
475    power_relay_active = switch_power_relais(0);//Deactivate power relays
476    cycle_active = false;//Deactivate cycling
477    wait_start_time = 0;//reset cycling variables
478    max_time_counter = 0;
479    current_cycle_step = 1;
480
481   }
482
483
484   //*****************************************************************************
485   //
486   // This function implements the "start" command. It starts the measurement
487   // by activating Timer 0A.
488   //
489   //*****************************************************************************
490   int
```

```
491 Cmd_start(int argc, char *argv[])
492 {
493  int i = 0;
494
495  //Start cycling only if not already running
496  if(!cycle_active && !measurement_active && !g_Flag_calibrationModeOn){
497
498   //Generate Filename
499   sprintf(MEAS_FILE, "%02d%02d%02d%02d.txt", clock_month, clock_day, clock_hour, clock_min);
500
501   //Generate header info
502   char header[64];
503   add_to_file(MEAS_FILE, "*****************************************");
504
505   sprintf(header, "\nFilename: '%s'\nDate: %04d-%02d-%02d %02d:%02d:%02d", MEAS_FILE, clock_year, clock_month, clock_day, clock_hour,
          clock_min, clock_sec);
506   add_to_file(MEAS_FILE, header);
507
508   sprintf(header, "\nQuantity Cells: %02d\nCyclestep: %05d msec", config.quantity_cells, config.cyclestep);
509   add_to_file(MEAS_FILE, header);
510
511   sprintf(header, "\nIP Address: %03d.%03d.%03d.%03d\nMeasuretime: %05d min", config.ip_a, config.ip_b, config.ip_c, config.ip_d,
          config.measuretime/60);
512   add_to_file(MEAS_FILE, header);
513
514   sprintf(header, "\nMaximum cycle time: %05d min", config.max_cycle_time);
515   add_to_file(MEAS_FILE, header);
516
517   sprintf(header, "\nCell voltage limits: %05d mV , %05dmV", (int)(config.min_voltage/1000), (int)(config.max_voltage/1000));
518   add_to_file(MEAS_FILE, header);
519
520   sprintf(header, "\nMaximum current: %05d mA", config.max_current);
521   add_to_file(MEAS_FILE, header);
522
523   sprintf(header, "\nMaximum cell temperature: %05d °C", (int)(config.max_temp/10));
524   add_to_file(MEAS_FILE, header);
525
526   sprintf(header, "\nMaximum battery charge: %05d mAh", config.max_charge/3600);
527   add_to_file(MEAS_FILE, header);
528
529   sprintf(header, "\nCELL  | ADC Gain  |  ADC Offset");
530   add_to_file(MEAS_FILE, header);
531
532   for(i = 1; i <= 12; i++){
533    sprintf(header, "\n%02d:    %03d     %08d     %08d", i, config.A[i], config.B[i], config.C[i]);
534    add_to_file(MEAS_FILE, header);
535   }
536
537   sprintf(header, "\nCycle steps:  %02d", config.step_quantity);
538   add_to_file(MEAS_FILE, header);
539
540   sprintf(header, "\nStepID     Steptype        Parameter  Relayselect");
541   add_to_file(MEAS_FILE, header);
542
543   for(i = 1; i <= 100; i++){
544    if(i <= config.step_quantity){
545     sprintf(header,"\n%02i         %-15s %05d      %01d", i, cycle_types[config.cycle_steps[i].type ], config.cycle_steps[i].param,
            config.cycle_steps[i].relay);
546     add_to_file(MEAS_FILE, header);
547    } else{
548     sprintf(header,"\n%02i         none            00000      0", i);
549     add_to_file(MEAS_FILE, header);
550    }
551   }
552
553
554   add_to_file(MEAS_FILE, "\n*****************************************");
555   add_to_file(MEAS_FILE, "\nCE,YYYY,MM,DD,HH,MM,SS,VOL(uV),TEM,CUR(mA),R");
556   add_to_file(MEAS_FILE, "\n*************************************************\n");
557
558   //Enable Timer0A and Cycling Mode
559   TimerEnable(TIMER0_BASE, TIMER_A);
560   cycle_active = true;
561   measurement_active = true;
562
563   //turn on power relay for first cycle step and turn on LED
564   current_cycle_step = 1;
565   power_relay_active = switch_power_relais(config.cycle_steps[1].relay);
566
567   // Enable Timer 3 for LED toggle every 500ms
568   TimerEnable(TIMER3_BASE, TIMER_A);
569
570   UARTprintf("\nMeasurement cycling started");
571   if(control_connection){
572    telnet_write("Measurement cycling started\n");
```

```
573     }
574
575     //Enable Timer2A for cycle control
576     TimerEnable(TIMER2_BASE, TIMER_A);
577   }
578   else{
579     UARTprintf("\n ERROR: measurement already active or calibration active! \n");
580   }
581       //
582       // Return success.
583       //
584       return(0);
585 }
586
587 //*****************************************************************************
588 //
589 // This function implements the "stop" command. It stops the measurement
590 // by deactivating Timer 0A and Timer 2A.
591 //
592 //*****************************************************************************
593 int
594 Cmd_stop(int argc, char *argv[])
595 {
596   int i = 0;
597   //Disable Timer0A and Timer2A
598       TimerDisable(TIMER0_BASE, TIMER_A);
599       TimerDisable(TIMER2_BASE, TIMER_A);
600
601       UARTprintf("\nMeasurement cycling stopped");
602   if(control_connection){
603     telnet_write("Measurement cycling stopped\n");
604   }
605
606   //Disable Cycling mode and deselect powerrelays
607   cycle_active = false;
608   measurement_active = false;
609
610   //Disable LED toggle and switch off LED
611   TimerDisable(TIMER3_BASE, TIMER_A);
612   GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);
613
614   //reset measurement and cycling values
615   wait_start_time = 0;
616   max_time_counter = 0;
617   timer2second = 0;
618   current_cycle_step = 1;
619
620   gBat_charge = 0;
621   for(i = 1; i < 13; i++){
622     g_iCellVoltages[i]= 0;
623         g_iCellTemperature[i]= 0;
624   }
625   g_iCurrent = 0;
626
627
628   //switch off power relay
629   power_relay_active = switch_power_relais(0);
630
631   //Turn off Measurement-LED
632   GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  0x00);
633
634       //
635       // Return success.
636       //
637       return(0);
638 }
```

Quellcode E.2: Anhang/c/config.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          config.c
4
5  Auhtor:         Thomas Wisnewski
6  Credits:        Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Johannes Roehn
10          Martin Kusche
11          Oliver Schmidt
12          Stellaris Ware
13
14 last modified:      2015/03/19
```

```
15
16 Project Status       Under Construction
17 Status:          running
18
19 CCS:          5.5.1.00031
20 Stellarisware:       8555
21
22 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
23          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
24          NTC-Connectors, MAX3232 and Suplly Circuits
25
26 Description:       Source Code for configure operations
27
28 */
29
30 /*****************************************************************************
31 *
32 * Includings
33 *
34 *****************************************************************************/
35 #include <stdio.h>
36 #include <stdlib.h>
37 #include <string.h>
38 #include "math.h"
39 #include "third_party/fatfs/src/ff.h"
40 #include "third_party/fatfs/src/diskio.h"
41 #include <utils/uartstdio.h>
42 #include <driverlib/ethernet.h>
43 #include <utils/lwiplib.h>
44 #include "inc/hw_memmap.h"
45 #include "driverlib/timer.h"
46 #include "header/watchdog.h"
47 /*****************************************************************************
48 *
49 * Own Includings
50 *
51 *****************************************************************************/
52 #include "header/myadc.h"
53 #include "header/config.h"
54 #include "header/mysdcard.h"
55 #include "header/mycmdline.h"
56 #include "header/ethernet.h"
57 #include "header/control.h"
58 #include "header/relais.h"
59 #include "header/myadc.h"
60 #include "header/clocktimer.h"
61 #include "header/rtc.h"
62 #include "header/display.h"
63
64 // Watchdog variable extern
65 extern volatile tBoolean g_bFeedWatchdog;
66
67
68 // Define Name of Configurationfile
69 const char* CONFIG_FILE = "config.txt";
70 //*****************************************************************************
71 //
72 // A global variable for calibration
73 //
74 //*****************************************************************************
75 volatile char g_Flag_calibrationModeOn = 0;
76 volatile short g_calibrationChannel = 0;
77 volatile short g_calibrationVoltage = 0;
78 volatile char g_Flag_calibrationRedo = 0;
79 volatile char g_Flag_calibrationStop = 0;
80
81
82 //*****************************************************************************
83 //
84 // Defines the size of the buffer that holds the input data.
85 //
86 //*****************************************************************************
87 #define CONFIG_INPUT_DATA_SIZE     64
88
89 //*****************************************************************************
90 //
91 // A temporary data buffer used for input data
92 //
93 //*****************************************************************************
94 static char g_cTmpInputData[CONFIG_INPUT_DATA_SIZE];
95
96 //*****************************************************************************
97 //
98 // A global Flag initiating the configuration from sd card read
99 //
```

```
100 //*****************************************************************************
101 char g_ConfDone = 0;
102
103 //*****************************************************************************
104 //
105 // This is the table that holds the command names, implementing functions,
106 // and brief description.
107 //
108 //*****************************************************************************
109 tCmdLineEntry g_sConfigCmdTable[] =
110 {
111     { "help",     Cmd_help,      "     : Display list of commands" },
112     { "h",        Cmd_help,   "        : alias for help" },
113     { "?",        Cmd_help,   "        : alias for help" },
114     { "print",    Cmd_print,     "     : Print current Configuration" },
115     { "quantity",    Cmd_quantity,      " : Set quantity of cells (01 - 12)" },
116     { "cyclestep",   Cmd_cyclestep,     ": Set measurement period in msec (500 - 60000)" },
117     { "coefficient",Cmd_set_coeff,      ": Set calibration coefficient (eg. 'coefficient [ABC] [01-12] value')" },
118     { "calibrate",   Cmd_calibrate,     ": Calibrate current sensor to zero" },
119     { "calVolt",   Cmd_calVolt,      ": calibrate Voltage measurement" },
120     { "ip",    Cmd_ip,       "       : Set IP address (eg. 'ip 192.168.000.128')" },
121     { "relais",   Cmd_relais,      "    : Manualy activates choosen powerrelais (e.g. 'relais 1')"},
122     { "relsel",   Cmd_relsel,      "    : Set Relayselect (eg. 'relsel 01 wait 00001 1')"},
123     { "relqua",   Cmd_relqua,      "    : Set quantity of Relay switches (01 - 999)"},
124     { "meastime",    Cmd_meastime, " : Set time of full measurement and cycling procedure in min (eg. 'meastime 1440')"},
125     { "save",     Cmd_save,       "    : Save configuration to SD-Card" },
126     { "clock",     Cmd_set_clock,  "    : Set the system clock, format: dd.mm.yyyy hh:mm (eg. 'clock 10.12.2013 13:45')" },
127     { "vlimits",     Cmd_set_limits,  "  : Set voltage min and max limits in mV (eg. 'vlimits 500 4000')" },
128     { "maxcurr",  Cmd_set_current,  "  : Set absolute maximum current limit in mA (eg. 'maxcurrent 1500')" },
129     { "maxcycletime",Cmd_max_time,     ": Set absolute maximum cycling time limit in min (eg. 'maxcycletime 1000')" },
130     { "maxtemp",  Cmd_max_temp,      "  : Set absolute maximum temperature limit in °C (eg. 'maxtemp 50')" },
131     { "maxcharge",  Cmd_max_charge,   ": Set absolute maximum charge taken through the battery in mAh (eg. 'maxcharge 500')" },
132     { "load",     Cmd_load,       "    : Load configuration from SD-Card and re-init" },
133     { "exit",     Cmd_config_exit,"    : Exit configuration operations" },
134     { 0, 0, 0 }
135 };
136
137 //holds the strings for the relsel comman0d
138 const char *cycle_types[] = {"wait","charge_cellV","discharge_cellV","charge_batV","discharge_batV","loop_start","loop_end"};
139
140
141 //*********external variables*****************
142 extern tCmdLineEntry g_sMainCmdTable;
143 extern tCmdLineEntry *g_psCmdTable;
144
145 // string for main location
146 extern const char *g_cMainLocalBuf;
147
148 // Global location buffer
149 extern char *g_cLocalBuf;
150
151 //define string for config location
152 const char *g_cConfigLocalBuf = "Config";
153
154 // String buffer for print operations to UART and Ethernet
155 extern char print_buffer[1024];
156
157 // TCP-Socket für die aktuelle Steuerungs-Verbindung
158 extern struct tcp_pcb* control_connection;
159
160 // MAC Adresse für weitere Verwendung im Programm
161 extern unsigned char pucMACArray[8];
162
163 //Flag zeigt an, ob Konfigurationsdatein eingelsen
164 extern char g_ConfDone;
165
166
167 //*****************************************************************************
168 //Extern variables for Clock
169 //*****************************************************************************
170 extern volatile unsigned long clock_msec;
171 extern volatile unsigned long clock_sec;
172 extern volatile unsigned long clock_min;
173 extern volatile unsigned long clock_hour;
174 extern volatile unsigned long clock_day;
175 extern volatile unsigned long clock_month;
176 extern volatile unsigned long clock_year;
177
178
179 /*
180  * Initialisieren der Konfiguration.
181  */
182 void init_config(void){
183
184   UARTprintf("Initializing Configuration:\n");
```

```
185
186   // set Command line pointer to the beginning of the Config command structure
187   g_psCmdTable = &g_sConfigCmdTable[0];
188   // set location buffer to Browser
189   g_cLocalBuf = (char*)g_cConfigLocalBuf;
190
191   config_read();
192
193   // set Command line pointer to the beginning of the main command structure
194   g_psCmdTable = &g_sMainCmdTable;
195   // set location buffer to main
196   g_cLocalBuf = (char*)g_cMainLocalBuf;
197
198   UARTprintf("Initializing Configuration...done\n");
199
200 }
201
202
203
204 /*
205  * Löschen der Konfiguration
206  */
207 void config_erase(void)
208 {
209  delete_file(CONFIG_FILE);
210  create_file(CONFIG_FILE);
211 }
212
213 /*
214  * Auslesen der Konfiguration.
215  */
216 void config_read(void)
217 {
218  char buffer[ CONF_BUF_SIZE ];
219  int i, err_flag = 0;
220
221  //Load default values
222  config.quantity_cells = 01;
223  config.cyclestep = 500;
224
225  config.A[0]  = 0;
226  config.A[1]  = 2.948254e-12;
227  config.A[2]  = 0;
228  config.A[3]  = 0;
229  config.A[4]  = 0;
230  config.A[5]  = 0;
231  config.A[6]  = 0;
232  config.A[7]  = 0;
233  config.A[8]  = 0;
234  config.A[9]  = 0;
235  config.A[10] = 0;
236  config.A[11] = 0;
237  config.A[12] = 0;
238
239  config.B[0]  = 0;
240  config.B[1]  = 7.807312e-05;
241  config.B[2]  = 0;
242  config.B[3]  = 0;
243  config.B[4]  = 0;
244  config.B[5]  = 0;
245  config.B[6]  = 0;
246  config.B[7]  = 0;
247  config.B[8]  = 0;
248  config.B[9]  = 0;
249  config.B[10] = 0;
250  config.B[11] = 0;
251  config.B[12] = 0;
252
253  config.C[0]  = 0;
254  config.C[1]  = 1.994334e-02;
255  config.C[2]  = 0;
256  config.C[3]  = 0;
257  config.C[4]  = 0;
258  config.C[5]  = 0;
259  config.C[6]  = 0;
260  config.C[7]  = 0;
261  config.C[8]  = 0;
262  config.C[9]  = 0;
263  config.C[10] = 0;
264  config.C[11] = 0;
265  config.C[12] = 0;
266
267  config.ip_a = 192;
268  config.ip_b = 168;
269  config.ip_c = 000;
```

```
270  config.ip_d = 128;
271
272  config.step_quantity = 1;
273
274  config.cycle_steps[0].type = 0;
275  config.cycle_steps[0].param = 0;
276     config.cycle_steps[0].relay = 0;
277     config.cycle_steps[1].type = WAIT;
278  config.cycle_steps[1].param = 0;
279  config.cycle_steps[1].relay = 0;
280  for(i = 2; i <= MAX_CYCLE_STEPS; i++){
281   config.cycle_steps[i] = (cycle_step_t) {0};
282  }
283
284  config.measuretime = 1000;
285  config.max_cycle_time = 1;
286
287  config.max_current = 10000;
288  config.min_voltage = 0;
289     config.max_voltage = 4500000;
290  config.max_temp = 400;
291  config.max_charge = 100 * 3600;
292
293  //If Error reading CONFIG_GILE, load default Values
294  if( read_into_buffer( CONFIG_FILE, &buffer[0] ) ) {
295
296   UARTprintf("\nERROR opening %s", CONFIG_FILE);
297   UARTprintf("\nLoading default Values\n\n");
298
299   g_ConfDone = 1;
300   config_print_uart();
301   return;
302  }
303
304
305  //Read config data from SD-Card
306  char *tmp1,*tmp2;
307  tmp1 = tmp2 = buffer;//set pointers to the start of the file
308
309  while(*tmp1){//parse till end of config file
310
311   if(*tmp1 == '\n'){//Line Feed reached, execute command
312
313    *tmp1 = 0;//terminate string
314
315    if(strlen(tmp2) != 0){//skip empty lines
316     if((*tmp2) == '/' && (*(tmp2 + 1)) == '/');//Ignore comment lines
317     else if(CmdLineProcess(tmp2) == CMDLINE_BAD_CMD){//process line and check for error
318      UARTprintf("\nBad command in config file line1: %s\nLoading default values!\n", tmp2);
319      err_flag = 1;
320      break;
321     }
322    }
323
324    tmp2 = (tmp1 + 1);//next element of the file
325   }
326
327   tmp1++;//next character
328
329   if((*tmp1) == 0){//end of config file reached
330    if(strlen(tmp2) != 0){//skip empty lines
331     if((*tmp2) == '/' && (*(tmp2 + 1)) == '/');//Ignore comment lines
332     else if(CmdLineProcess(tmp2) == CMDLINE_BAD_CMD){//execute last line of the file
333       UARTprintf("\nBad command in config file line2: %s\nLoading default values!\n", tmp2);
334       err_flag = 1;
335      }
336    }
337   }
338  }
339
340  if(err_flag){
341
342   handlerState = ERROR;
343
344   //Load default values
345   config.quantity_cells = 01;
346   config.cyclestep = 500;
347
348   config.A[0]  = 0;
349   config.A[1]  = 2.948254e-12;
350   config.A[2]  = 0;
351   config.A[3]  = 0;
352   config.A[4]  = 0;
353   config.A[5]  = 0;
354   config.A[6]  = 0;
```

```
355   config.A[7]  = 0;
356   config.A[8]  = 0;
357   config.A[9]  = 0;
358   config.A[10] = 0;
359   config.A[11] = 0;
360   config.A[12] = 0;
361
362   config.B[0]  = 0;
363   config.B[1]  = 7.807312e-05;
364   config.B[2]  = 0;
365   config.B[3]  = 0;
366   config.B[4]  = 0;
367   config.B[5]  = 0;
368   config.B[6]  = 0;
369   config.B[7]  = 0;
370   config.B[8]  = 0;
371   config.B[9]  = 0;
372   config.B[10] = 0;
373   config.B[11] = 0;
374   config.B[12] = 0;
375
376   config.C[0]  = 0;
377   config.C[1]  = 1.994334e-02;
378   config.C[2]  = 0;
379   config.C[3]  = 0;
380   config.C[4]  = 0;
381   config.C[5]  = 0;
382   config.C[6]  = 0;
383   config.C[7]  = 0;
384   config.C[8]  = 0;
385   config.C[9]  = 0;
386   config.C[10] = 0;
387   config.C[11] = 0;
388   config.C[12] = 0;
389
390   config.ip_a = 192;
391   config.ip_b = 168;
392   config.ip_c = 000;
393   config.ip_d = 128;
394
395   config.step_quantity = 1;
396
397   config.cycle_steps[0].type = WAIT;
398   config.cycle_steps[0].param = 0;
399   config.cycle_steps[0].relay = 0;
400   config.cycle_steps[1].type = WAIT;
401   config.cycle_steps[1].param = 0;
402   config.cycle_steps[1].relay = 0;
403   for(i = 2; i <= MAX_CYCLE_STEPS; i++){
404    config.cycle_steps[i] = (cycle_step_t) {0};
405   }
406
407
408   config.measuretime = 1000;
409   config.max_cycle_time = 1;
410
411   config.max_current = 10000;
412   config.min_voltage = 0;
413      config.max_voltage = 4500000;
414   config.max_temp = 400;
415   config.max_charge = 100 * 3600;
416  }
417
418  g_ConfDone = 1;//Configuration done
419  config_print_uart();
420
421  return;
422 }
423
424
425 /*
426  * Schreiben der aktuellen Konfiguration in die config.txt (SD-Karte).
427  */
428 int config_write(void)
429 {
430  int i = 0;
431
432  delete_file(CONFIG_FILE); //TODO: Keine Fehlerbehandlung
433  create_file(CONFIG_FILE);
434
435  char write_quantity_cells[3];
436  sprintf(write_quantity_cells, "%02i", config.quantity_cells);
437  config_write_struct("quantity ", write_quantity_cells );
438
439  char write_cyclestep[5];
```

```
440   sprintf(write_cyclestep, "%05i", config.cyclestep);
441   config_write_struct("cyclestep ", write_cyclestep );
442
443   add_to_file(CONFIG_FILE, "\n");
444
445   char write_adc_A[32];
446   for(i = 1; i <= 12; i++){
447    sprintf(write_adc_A, "%02i %08d", i, config.A[i]);
448    config_write_struct("coefficient A ", write_adc_A );
449   }
450
451   add_to_file(CONFIG_FILE, "\n");
452
453   char write_B[32];
454   for(i = 1; i <= 12; i++){
455    sprintf(write_B, "%02i %08d", i, config.B[i]);
456    config_write_struct("coefficient B ", write_B );
457   }
458
459   add_to_file(CONFIG_FILE, "\n");
460
461   char write_adc_C[32];
462   for(i = 1; i <= 12; i++){
463    sprintf(write_adc_C, "%02i %08d ", i, config.C[i]);
464    config_write_struct("coefficient C ", write_adc_C );
465   }
466
467   add_to_file(CONFIG_FILE, "\n");
468
469   char write_ip_address[48];
470   sprintf(write_ip_address, "%03d.%03d.%03d.%03d", config.ip_a, config.ip_b, config.ip_c, config.ip_d);
471   config_write_struct("ip ", write_ip_address );
472
473   add_to_file(CONFIG_FILE, "\n");
474
475   char write_measure_time[32];
476   sprintf(write_measure_time,  "%05d", config.measuretime);
477   config_write_struct("meastime ", write_measure_time);
478
479   sprintf(write_measure_time, "%d", config.max_cycle_time);
480   config_write_struct("maxcycletime ", write_measure_time);
481
482   add_to_file(CONFIG_FILE, "\n");
483
484   char write_limits[32];
485   sprintf(write_limits, "%d %d", config.min_voltage/1000, config.max_voltage/1000); //Gespeichert in muV
486   config_write_struct("vlimits ", write_limits);
487
488   sprintf(write_limits, "%d", config.max_current);
489   config_write_struct("maxcurr ", write_limits);
490
491   sprintf(write_limits, "%d", (int)(config.max_temp/10));
492   config_write_struct("maxtemp ", write_limits);
493
494   sprintf(write_limits, "%d", (int)(config.max_charge/3600));
495   config_write_struct("maxcharge ", write_limits);
496
497   char write_relay_control[32];
498   for(i = 1; i <= config.step_quantity; i++){
499     sprintf(write_relay_control, "%02i %s %05d %01d", i, cycle_types[config.cycle_steps[i].type - 1], config.cycle_steps[i].param,
            config.cycle_steps[i].relay);
500     config_write_struct("relsel ", write_relay_control);
501   }
502
503   return 1;
504 }
505
506 /*
507  * Hilfsfunktion zum strukturieren der Konfigurationsdatei
508  */
509 void config_write_struct(const char *conf_name, const char *conf_val){
510
511   add_to_file(CONFIG_FILE, conf_name);
512   add_to_file(CONFIG_FILE, conf_val);
513   add_to_file(CONFIG_FILE, "\n");
514
515 }
516
517 /*
518  * Ausgabe der Konfiguration Ueber UART und Ethernet.
519  * Gibt tabellarisch alle Werte der aktuellen Konfiguration über
520  * die UART und ggbf. vorhandene Ethernet Verbindung aus.
521  */
522 void config_print_uart(void)
523 {
```

```
524 int i = 0;
525
526    UARTprintf("\n  Quantity_cells: %d Cells", config.quantity_cells);
527    UARTprintf("\n  Cyclestep:      %d msec", config.cyclestep);
528    UARTprintf("\n  IP Address:     %03d.%03d.%03d.%03d", config.ip_a, config.ip_b, config.ip_c, config.ip_d);
529    UARTprintf("\n\n    Cell |    ADC A    |    ADC B    |    ADC C   \n");
530
531 for(i = 1; i <= config.quantity_cells; i++){
532  UARTprintf("\n     %02d:    %03d      %08d      %08d", i, config.A[i], config.B[i], config.C[i]);
533 }
534
535 UARTprintf("\n\nMeasuretime (min): %d", config.measuretime/60);
536
537 UARTprintf("\n\nMaximum cycling time (min): %d", config.max_cycle_time);
538
539 UARTprintf("\n\nVoltage limits (mV):  min:%d  max:%d", (int)(config.min_voltage/1000), (int)(config.max_voltage/1000));
540
541 UARTprintf("\n\nMaximum current (mA): %d", config.max_current);
542
543 UARTprintf("\n\nMaximum cell temperature (°C): %d", (int)(config.max_temp/10));
544
545 UARTprintf("\n\nMaximum battery charge (mAh): %d", config.max_charge/3600);
546
547 UARTprintf("\n\nCycle steps: %d", config.step_quantity);
548
549 UARTprintf("\n\nStepID      Steptype         Parameter  Relayselect");
550
551 for(i = 1; i <= config.step_quantity; i++){
552  sprintf(print_buffer,"\n%02i         %-15s %05d       %01d", i, cycle_types[config.cycle_steps[i].type ], config.cycle_steps[i].param,
        config.cycle_steps[i].relay);
553  UARTprintf(print_buffer);
554 }
555
556 UARTprintf("\n\n");
557
558
559
560    if(control_connection){
561
562     telnet_write("\n");
563     sprintf(print_buffer, "  Quantity_cells: %d Cells\n", config.quantity_cells);
564  telnet_write(print_buffer);
565     sprintf(print_buffer, "  Cyclestep:      %d msec\n", config.cyclestep);
566  telnet_write(print_buffer);
567     sprintf(print_buffer, "  IP Address:     %03d.%03d.%03d.%03d\n", config.ip_a, config.ip_b, config.ip_c, config.ip_d);
568  telnet_write(print_buffer);
569  telnet_write("\n");
570     sprintf(print_buffer, "    Cell |  ADC Gain | ADC Offset\n");
571  telnet_write(print_buffer);
572  telnet_write("\n");
573  telnet_write("\n");
574
575  sprintf(print_buffer, "");
576  char buf[64];
577  for(i = 1; i <= config.quantity_cells; i++){
578   sprintf(buf, "  %02d:    %03d      %08d      %08d\n", i, config.A[i], config.B[i], config.C[i]);
579   strcat(print_buffer, buf);
580  }
581
582  telnet_write(print_buffer);
583
584  telnet_write("\n");
585  telnet_write("\n");
586
587     sprintf(print_buffer, "  Measuretime (min): %d\n", config.measuretime);
588  telnet_write(print_buffer);
589
590  sprintf(print_buffer, "\n\nMaximum cycling time (min): %d", config.max_cycle_time);
591  telnet_write(print_buffer);
592
593  sprintf(print_buffer, "\n\nVoltage limits (mV):  min:%d  max:%d", (int)(config.min_voltage/1000), (int)(config.max_voltage/1000));
594  telnet_write(print_buffer);
595
596  sprintf(print_buffer,"\n\nMaximum current (mA): %d", config.max_current);
597  telnet_write(print_buffer);
598
599  sprintf(print_buffer,"\n\nMaximum cell temperature (°C): %d", (int)(config.max_temp/10));
600  telnet_write(print_buffer);
601
602  sprintf(print_buffer,"\n\nMaximum battery charge (mAh): %d", config.max_charge/3600);
603  telnet_write(print_buffer);
604
605  telnet_write("\n");
606  telnet_write("\n");
607
```

```
608       sprintf(print_buffer, "  Cyclesteps   : %01d\n", config.step_quantity);
609    telnet_write(print_buffer);
610
611    telnet_write("\n");
612
613
614    sprintf(print_buffer,"\n\nStepID      Steptype        Parameter   Relayselect");
615    telnet_write(print_buffer);
616
617    for(i = 1; i <= config.step_quantity; i++){
618     sprintf(print_buffer,"\n%02i          %-15s %05d       %01d", i, cycle_types[config.cycle_steps[i].type - 1], config.cycle_steps[i].
           param, config.cycle_steps[i].relay);
619     telnet_write(print_buffer);
620    }
621
622
623    telnet_write("\n");
624    telnet_write("\n");
625
626      }
627
628
629
630
631 }
632
633 //*****************************************************************************
634 //
635 // Implementation of an command line for configure operations based on stellarisware sd-card example
636 //
637 //*****************************************************************************
638 int config_cmd_line(int argc, char *argv[])
639 {
640
641     // set Command line pointer to the beginning of the Config command structure
642     g_psCmdTable = &g_sConfigCmdTable[0];
643
644     // set location buffer to Browser
645     g_cLocalBuf = (char*)g_cConfigLocalBuf;
646
647  return(0);
648 }
649
650 //*****************************************************************************
651 //
652 // Implementation of an command to exit to main command line entry
653 //
654 //*****************************************************************************
655 int Cmd_config_exit(int argc, char *argv[])
656 {
657     // set Command line pointer to the beginning of the main command structure
658     g_psCmdTable = &g_sMainCmdTable;
659
660     // set location buffer to main
661     g_cLocalBuf = (char*)g_cMainLocalBuf;
662
663  return(0);
664 }
665
666 //*****************************************************************************
667 //
668 // Implementation of an command to set the calB value
669 //
670 //*****************************************************************************
671 int Cmd_set_coeff(int argc, char *argv[])
672 {
673
674  if(argc == 4){//takes only 3 parameters
675   int cell, cal;
676   float *coeff;
677   float multiplier=1;
678   char buffer[30];
679
680   //
681   // Copy the first input data into buffer.
682   //
683   strcpy(g_cTmpInputData, argv[2]);
684
685   cell = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
686
687   switch (*argv[1]) {
688   case 'A':
689    coeff = config.A;
690    multiplier = 0.000000000001;  // 18
691    break;
```

```
692   case 'B':
693    coeff = config.B;
694    multiplier = 0.000001; //12
695    break;
696   case 'C':
697    coeff = config.C;
698    multiplier = 0.001; //9
699    break;
700   default:
701    return -1;
702   }
703
704
705   if(cell > 0 && cell < 13){//check borders
706    //
707    // Copy the second input data into buffer.
708    //
709    strcpy(g_cTmpInputData, argv[3]);
710
711    cal = (int) strtol(g_cTmpInputData, (char **)NULL, 16);
712
713    coeff[cell] = cal * multiplier;
714
715    return 0;//success
716   }
717  }
718  return -1;//Bad command
719 }
720
721 //*****************************************************************************
722 //
723 // Implementation of an command to autocalibrate hall sensor offset
724 //
725 //*****************************************************************************
726 int Cmd_calibrate(int argc, char *argv[])
727 {
728
729  //
730  // Calibrate and set offset for current
731 /* //
732  config.zerocurrentch1 = 0;
733  config.zerocurrentch2 = 0;
734
735  config.zerocurrentch1 = (32770 - adc_get_single_value_ch2());
736  config.zerocurrentch2 = (32770 - adc_get_single_value_ch3());
737 */
738 // new myadc.c
739     adc_calibrate_hall_sensor();
740  return(0);
741
742 }
743 //*****************************************************************************
744 //
745 // Implementation of an command to calibrate Voltage measurement
746 //
747 //*****************************************************************************
748 int Cmd_calVolt(int argc, char *argv[])
749 {
750  g_Flag_calibrationModeOn=1;
751 // calibrate_adc_quadratisch();
752  return(0);
753 }
754
755
756 //*****************************************************************************
757 //
758 // Implementation of an command to set the up-address
759 //
760 //*****************************************************************************
761 int Cmd_ip(int argc, char *argv[])
762 {
763  if(argc == 2){//takes only one parameter
764   char *tmp1,*tmp2, i;
765   int ip, flag = 0;
766   //
767   // Copy the first input data into buffer.
768   //
769   strcpy(g_cTmpInputData, argv[1]);
770
771   tmp1 = tmp2 = g_cTmpInputData;
772
773   for (i = 0; i < 3; i++){
774    while(*tmp1++ != '.');//look for the separating dot sign
775    *(tmp1 - 1) = 0;//write trailing string null
776    if(i == 0){
```

```
777     ip = (int) strtol(tmp2, (char **)NULL, 10);
778     if(ip >= 0 && ip <= 255)//check borders
779      config.ip_a = ip;//first argument
780     else {flag = 1;break;}
781    }else if(i == 1){
782     ip = (int) strtol(tmp2, (char **)NULL, 10);
783     if(ip >= 0 && ip <= 255)//check borders
784      config.ip_b = ip;//second argument
785     else {flag = 1;break;}
786    }else if(i == 2){
787     ip = (int) strtol(tmp2, (char **)NULL, 10);
788     if(ip >= 0 && ip <= 255)//check borders
789      config.ip_c = ip;//third argument
790     else {flag = 1; break;}
791     ip = (int) strtol(tmp1, (char **)NULL, 10);
792     if(ip >= 0 && ip <= 255)//check borders
793      config.ip_d = ip;//fourth argument
794     else {flag = 1;break;}
795    }
796    tmp2 = tmp1;
797   }
798
799   if(!flag){
800    if(g_ConfDone){
801     //REinit Ethernet connection
802     struct ip_addr local_addr;
803     IP4_ADDR(&local_addr,config.ip_d,config.ip_c,config.ip_b,config.ip_a);
804     lwIPNetworkConfigChange(local_addr.addr, g_ulNetMask, g_ulGWAddr, g_ulIPMode);//Change IP Address and restart network
805    }
806    return 0;//success
807   }
808  }
809  return -1;//Bad command
810 }
811
812 //*****************************************************************************
813 //
814 // Implementation of an command to print actual configuration to UART and Ethernet
815 //
816 //*****************************************************************************
817 int Cmd_print(int argc, char *argv[])
818 {
819  config_print_uart();
820  return(0);
821 }
822
823 //*****************************************************************************
824 //
825 // Implementation of an command to set the quantity of battery cells
826 //
827 //*****************************************************************************
828 int Cmd_quantity(int argc, char *argv[])
829 {
830
831  if(argc == 2){//check if only one command
832   //
833   // Copy the input data into buffer.
834   //
835   strcpy(g_cTmpInputData, argv[1]);
836   long data = (long) strtol(g_cTmpInputData, (char **)NULL, 10);
837
838   if(data >= 0 && data < 13){//check if in borders
839    config.quantity_cells = data;
840    return 0;//success
841   }
842  }
843  return -1;//return bad command
844 }
845
846
847 //*****************************************************************************
848 //
849 // Implementation of an command to set the time between measurment ("Cyclestep")
850 //
851 //*****************************************************************************
852 int Cmd_cyclestep(int argc, char *argv[])
853 {
854  if(argc == 2){
855   //
856   // Copy the input data into buffer.
857   //
858   strcpy(g_cTmpInputData, argv[1]);
859
860   long data = (long) strtol(g_cTmpInputData, (char **)NULL, 10);
861
```

```
862   if( data >= 500 && data <= 60000){//check borders
863    config.cyclestep = data;
864
865    if(g_ConfDone)
866     reinit_timer0();//Reinit timer only if not in configuration from sd card file
867
868    return 0;//success
869   }
870  }
871  return -1; //Bad command
872 }
873
874 //*****************************************************************************
875 //
876 // Implementation of an command to set the measurement time
877 //
878 //*****************************************************************************
879 int Cmd_meastime(int argc, char *argv[])
880 {
881  if(argc == 2){
882   int time;
883   //
884   // Copy the input data into buffer.
885   //
886   strcpy(g_cTmpInputData, argv[1]);
887
888   time = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
889   if((time > 0) && (time < 0xfffffffe / 60)){
890    config.measuretime = time*60; //umrechnen in sek
891    return 0;
892   }
893
894
895  }
896  return -1;
897
898 }
899
900 //*****************************************************************************
901 //
902 // Implementation of an command to set the quantity loadrelay switches
903 //
904 //*****************************************************************************
905 int Cmd_relqua(int argc, char *argv[])
906 {
907  if(argc == 2){
908   int step_qua;
909   //
910   // Copy the input data into buffer.
911   //
912   strcpy(g_cTmpInputData, argv[1]);
913
914   step_qua = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
915   if(step_qua > 0 && step_qua <= config.step_quantity){
916    int i;
917    config.step_quantity = step_qua;
918    //set Relsel values more than actual relay quantity to zero
919    for(i = (config.step_quantity + 1); i < MAX_CYCLE_STEPS+1; i++)
920     config.cycle_steps[i] = (cycle_step_t) {0};
921
922    return 0;
923   }
924  }
925  return -1;
926 }
927
928 //*****************************************************************************
929 //
930 // Implementation of an command to set the selection of a loadrelay by given time
931 //
932 //*****************************************************************************
933 int Cmd_relsel(int argc, char *argv[])
934 {
935  if(argc == 5){//takes only 4 parameters
936   int stepId;
937
938   //
939   // Copy the first input data into buffer.
940   //
941   strcpy(g_cTmpInputData, argv[1]);
942
943   //
944   // Assign relayid
945   //
946   stepId = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
```

```
947   if(stepId > 0 && (stepId <= config.step_quantity + 1)){
948
949    char type, flag = false;
950
951    //
952    // Copy the second input data into buffer.
953    //
954    strcpy(g_cTmpInputData, argv[2]);
955
956    for(type = 0; type < MAX_STEP_TYPES; type++){
957     if(strcmp(cycle_types[type], g_cTmpInputData) == 0){
958      flag = true;
959      break;
960     }
961    }
962    if(flag){
963     int param;
964
965     //
966     // Copy the third input data into buffer.
967     //
968     strcpy(g_cTmpInputData, argv[3]);
969
970     param = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
971     if(param >= 0){
972      int relay;
973
974      if(type == WAIT)//check borders for each type
975       if(param > 99999)
976        return -1;
977      else if(type == CHARGE_CELLV)
978       if(param > 5000)
979        return -1;
980      else if(type == DISCHARGE_CELLV)
981       if(param > 5000)
982        return -1;
983      else if(type == CHARGE_BATV)
984       if(param > 60000)
985        return -1;
986      else if(type == DISCHARGE_BATV)
987       if(param > 60000)
988        return -1;
989      else if(type == LOOP_START)
990       if(param < 1)
991        return -1;
992
993      //
994      // Copy the fourth input data into buffer.
995      //
996      strcpy(g_cTmpInputData, argv[4]);
997      relay = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
998
999      if(relay >= 0 && relay < 5){
1000      config.cycle_steps[stepId].type = type;
1001      config.cycle_steps[stepId].param = param;//muV
1002      config.cycle_steps[stepId].relay = relay;
1003
1004      if(stepId == config.step_quantity + 1)
1005       config.step_quantity++;
1006
1007      return 0;//success
1008     }
1009
1010    }
1011   }
1012  }
1013 }
1014 return -1;
1015 }
1016
1017
1018 //***************************************************************************
1019 //
1020 // Implementation of an command to save current configuration to SD-Card
1021 //
1022 //***************************************************************************
1023 int Cmd_save(int argc, char *argv[])
1024 {
1025  config_write();
1026  return 0;
1027 }
1028
1029 //***************************************************************************
1030 //
1031 // Implementation of an command to load the configuration file from SD-Card
```

```
1032 // and reinitialize
1033 //
1034 //*****************************************************************************
1035 int Cmd_load(int argc, char *argv[])
1036 {
1037  config_read();
1038  return 0;
1039 }
1040
1041 //*****************************************************************************
1042 //
1043 // Implementation of an command to set the current system clock and RTC
1044 //
1045 //*****************************************************************************
1046 int Cmd_set_clock(int argc, char *argv[])
1047 {
1048  if(argc == 3){//takes only 2 parameters
1049   strcpy(g_cTmpInputData, argv[1]);
1050   clock_day =    10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
1051
1052   clock_month =  10 * (g_cTmpInputData[3] - '0') + (g_cTmpInputData[4] - '0') ;
1053
1054   clock_year =   1000 * (g_cTmpInputData[6] - '0') + 100 * (g_cTmpInputData[7] - '0') + 10 * (g_cTmpInputData[8] - '0') + (
         g_cTmpInputData[9] - '0');
1055
1056   strcpy(g_cTmpInputData, argv[2]);
1057   clock_hour =   10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
1058
1059   clock_min =    10 * (g_cTmpInputData[3] - '0') + (g_cTmpInputData[4] - '0') ;
1060
1061   clock_sec =    0;
1062
1063   set_rtc_values(clock_year - 2000, clock_month, clock_day, clock_hour, clock_min, clock_sec);//Set Time in RTC
1064
1065   return 0;
1066  }
1067  return -1;
1068 }
1069
1070 //*****************************************************************************
1071 //
1072 // Implementation of an command to set the minimum and maximum voltages
1073 //
1074 //*****************************************************************************
1075 int Cmd_set_limits (int argc, char *argv[]){
1076
1077  if(argc == 3){//takes only 2 parameters
1078   int min, max;
1079
1080   strcpy(g_cTmpInputData, argv[1]);
1081
1082   min = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1083
1084   strcpy(g_cTmpInputData, argv[2]);
1085
1086   max = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1087
1088   if(min >= 0 && min < max && max <= 5000){//check borders
1089
1090    config.min_voltage = min * 1000;//muV
1091    config.max_voltage = max * 1000;//muV
1092    return 0;//success
1093   }
1094  }
1095  return -1;//Bad command
1096 }
1097
1098 //*****************************************************************************
1099 //
1100 // Implementation of an command to set the absolute maximum current allowed
1101 //
1102 //*****************************************************************************
1103 int Cmd_set_current(int argc, char *argv[]){
1104
1105  if(argc == 2){//check if only one command
1106
1107   strcpy(g_cTmpInputData, argv[1]);
1108   int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1109
1110   if(data > 0 && data <= 500000){//check if in borders
1111    config.max_current = data;
1112    return 0;//success
1113   }
1114  }
1115  return -1;//return bad command
```

```
1116 }
1117
1118 //*****************************************************************************
1119 //
1120 // Implementation of an command to set the absolute maximum cycling time allowed
1121 //
1122 //*****************************************************************************
1123 int Cmd_max_time (int argc, char *argv[]){
1124
1125  if(argc == 2){//check if only one command
1126
1127   strcpy(g_cTmpInputData, argv[1]);
1128   int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1129
1130   if(data > 0 && data <= 99999){//check if in borders
1131    config.max_cycle_time = data;
1132    return 0;//success
1133   }
1134  }
1135
1136  return -1;//return bad command
1137 }
1138
1139 //*****************************************************************************
1140 //
1141 // Implementation of an command to set the absolute maximum temperature allowed
1142 //
1143 //*****************************************************************************
1144 int Cmd_max_temp (int argc, char *argv[]){
1145
1146  if(argc == 2){//check if only one command
1147
1148   strcpy(g_cTmpInputData, argv[1]);
1149   int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1150
1151   if(data >= 0 && data <= 150){//check if in borders
1152    config.max_temp = data * 10;
1153    return 0;//success
1154   }
1155  }
1156  return -1;//return bad command
1157 }
1158
1159
1160 //*****************************************************************************
1161 //
1162 // Implementation of an command to set the absolute maximum temperature allowed
1163 //
1164 //*****************************************************************************
1165 int Cmd_max_charge (int argc, char *argv[]){
1166
1167  if(argc == 2){//check if only one command
1168
1169   strcpy(g_cTmpInputData, argv[1]);
1170   int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1171
1172   if(data > 0){//check if in border
1173    config.max_charge = data * 3600;//saved in mAsec
1174    return 0;//success
1175   }
1176  }
1177  return -1;//return bad command
1178 }
1179
1180 //*********************************************************************
1181 // Function used to calibrate channelwise 16-bit adc. Use debuger and
1182 // breakpoints. Values of ADC are written to adc.txt. Use two Voltages.
1183 // One low (ca. 300 mV) and one high (ca. 3.2V). Experience made with FLUKE 45
1184 //*********************************************************************
1185 //void calibrate_adc(void){
1186 //
1187 // int i = 0;
1188 // int j = 0;
1189 // char buf[32];
1190 //
1191 // delete_file("adc.txt");
1192 // create_file("adc.txt");
1193 //
1194 //
1195 // for(j = 1; j <= 12; j++){
1196 //
1197 //
1198 //  switch_relais(j);
1199 //  sprintf(buf,"\n CELL LOW %d\n", j);
1200 //  add_to_file("adc.txt", buf);
```

```
1201 //
1202 //   for(i=1 ; i<=20 ; i++){
1203 //
1204 //    sprintf(buf,"%d\n", adc_get_single_value());
1205 //    add_to_file("adc.txt", buf);
1206 //   }
1207 //   UARTprintf("\n%d LOW done",j);
1208 //   UARTprintf("\nnext");
1209 //
1210 // }
1211 // UARTprintf("\nALL LOW DONE!");
1212 //
1213 // for(j = 1; j <= 12; j++){
1214 //
1215 //
1216 //    switch_relais(j);
1217 //    sprintf(buf,"\n CELL HIGH %d\n", j);
1218 //    add_to_file("adc.txt", buf);
1219 //
1220 //    for(i=1 ; i<=20 ; i++){
1221 //
1222 //     sprintf(buf,"%d\n", adc_get_single_value());
1223 //     add_to_file("adc.txt", buf);
1224 //     UARTprintf(buf);
1225 //    }
1226 //    UARTprintf("\n%d HIGH done",j);
1227 //    UARTprintf("\nnext");
1228 //
1229 //   }
1230 //   UARTprintf("\nALL HIGH DONE!");
1231 //
1232 //}
1233
1234
1235 void calibrate_adc_quadratisch_waitLowToHigh(){
1236  long avg;
1237  long temp;
1238
1239  temp = adc_get_single_value(channel1);
1240  avg = temp;
1241
1242  do {
1243   g_bFeedWatchdog = true;
1244
1245   temp = adc_get_single_value(channel1);
1246 //   UARTprintf("AusgabeADCcount1: %d\n",temp);
1247   avg += temp;
1248   avg >>= 1;
1249 //   UARTprintf("Diff1: %d - %d = %d\n",temp, avg, temp-avg);
1250  } while(((temp-avg) < CALIBRATION_THRESHOLD) && !g_Flag_calibrationRedo && !g_Flag_calibrationStop);
1251 }
1252
1253 void calibrate_adc_quadratisch_waitHighToLow(){
1254  long avg;
1255  long temp;
1256
1257  temp = adc_get_single_value(channel1);
1258  avg = temp;
1259
1260  do {
1261   g_bFeedWatchdog = true;
1262
1263
1264   temp = adc_get_single_value(channel1);
1265 //   UARTprintf("AusgabeADCcount2: %d\n",temp);
1266   avg += temp;
1267   avg >>= 1;
1268 //   UARTprintf("Diff2: %d - %d = %d\n",temp, avg, temp-avg);
1269  } while(((temp-avg) > -CALIBRATION_THRESHOLD) && !g_Flag_calibrationRedo && !g_Flag_calibrationStop);
1270 }
1271
1272 void calibrate_adc_quadratisch(){
1273 short k;
1274 char fileBuffer[6*MAX_NO_CHANNELS+1]; // (5Zeichen+Tab bzw. newLine) * MAX_NO_MEASUREMENTS + \0
1275 char *p_FileBuf=fileBuffer;
1276 unsigned long avg[MAX_NO_CHANNELS];
1277 unsigned long ul_delay_count = 4500;
1278 display_handler_t oldDisplayState = handlerState;
1279
1280
1281 delete_file("adc.txt");
1282 create_file("adc.txt");
1283 add_to_file("adc.txt", "\045Calibration File:\n\045Ch1\tCh2\tCh3\tCh4\tCh5\tCh6\tCh7\tCh8\tCh9\tCh10\tCh11\tCh12\n");
1284
1285 for(g_calibrationVoltage = 1; g_calibrationVoltage <= CALIBRATION_STEPS; g_calibrationVoltage++){
```

```
1286  for(g_calibrationChannel = 0; g_calibrationChannel < config.quantity_cells; g_calibrationChannel++){
1287
1288   switch_relais(g_calibrationChannel+1);
1289
1290   handlerState=CALPLUG;
1291   calibrate_adc_quadratisch_waitLowToHigh();
1292
1293   if(g_Flag_calibrationStop==1){
1294    handlerState=oldDisplayState;
1295
1296    add_to_file("adc.txt", "\n calibration canceled\n");
1297    UARTprintf("%s","\n calibration canceled\n");
1298    g_Flag_calibrationStop=0;
1299    Cmd_config_exit(0,NULL); //rücksprung aus der config in die main
1300    return;
1301   }
1302
1303   ul_delay_count=4500;
1304   while (ul_delay_count--) // ca. 0.45 ms delay warten Enable High
1305    ;
1306   handlerState=CALMEAS;
1307   avg[g_calibrationChannel] = 0;
1308
1309   for(k = 0; k < NO_MEASUREMENTS; k++){
1310    avg[g_calibrationChannel] += adc_get_single_value(channel1);
1311 //    avg[g_calibrationChannel] += adc_get_single_value();
1312
1313   }
1314 //  avg[g_calibrationChannel] >>= NO_MEASUREMENTS_EXP; //Bildung des Mittelwertes
1315   avg[g_calibrationChannel] = (unsigned long) (avg[g_calibrationChannel] / NO_MEASUREMENTS + 0.5);
1316      UARTprintf("\nCh. %d : %d\n",g_calibrationChannel+1, avg[g_calibrationChannel]);
1317
1318   handlerState=CALPULL;
1319   calibrate_adc_quadratisch_waitHighToLow();
1320
1321   if(g_Flag_calibrationStop==1){
1322    handlerState=oldDisplayState;
1323
1324
1325    add_to_file("adc.txt", "\n calibration canceled\n");
1326    UARTprintf("%s","\n calibration canceled\n");
1327    g_Flag_calibrationStop=0;
1328    Cmd_config_exit(0,NULL);
1329    return;
1330   }
1331   if(g_Flag_calibrationRedo==1)
1332    break;
1333
1334  }
1335
1336  if(g_Flag_calibrationRedo==1){
1337   sprintf(p_FileBuf, "%c", 37);
1338   p_FileBuf+=1;
1339   g_calibrationVoltage--;
1340   g_Flag_calibrationRedo=0;
1341   UARTprintf("\n Wiederholung Messreihe ");
1342
1343 //  calibrate_adc_quadratisch_waitHighToLow();
1344
1345  }
1346
1347   sprintf(p_FileBuf,"%08d", avg[0]);     //%08 max. größe des ADC count
1348   p_FileBuf+=8;          //um 8 stellen den pointer verschieben
1349   for(k = 1; k < g_calibrationChannel; k++){ //bei mehr als einem CH-->Messwerte werden nebeneinander augegeben
1350    sprintf(p_FileBuf,"\t%08d", avg[k]);
1351    p_FileBuf+=9;
1352   }
1353   sprintf(p_FileBuf,"\n");
1354   p_FileBuf=fileBuffer;
1355   add_to_file("adc.txt", fileBuffer);
1356
1357 }
1358
1359  handlerState = oldDisplayState;
1360  Cmd_config_exit(0,NULL);
1361
1362
1363
1364 }
```

Quellcode E.3: Anhang/c/control.c

```
1 /*
```

```
2  Project:        battery_cycling_sw_v4
3  File:          control.c
4
5  Auhtor:         Thomas Wisniewski
6  Credits:        Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Johannes Roehn
10          Stellaris Ware
11
12 last modified:       2013/12/11
13
14 Project Status       Under Construction
15 Status:         running
16
17 CCS:         5.5.1.00031
18 Stellarisware:       8555
19
20 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
21          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
22          NTC-Connectors, MAX3232 and Suplly Circuits
23
24 Description:       Controlling TCP-link. Processing and
25          execution of commands.
26 */
27
28
29 #include "header/control.h"
30 #include "inc/hw_types.h"
31 #include "utils/uartstdio.h"
32 #include <stdlib.h>
33 #include <string.h>
34 #include <stdio.h>
35 #include "header/mycmdline.h"
36
37
38 struct tcp_pcb* control_tpcb;  // Server-Socket für die TCP-Verbindung
39 // Stream zur Speicherung aller erhaltenen Daten Ã¼ber den TCP-Socket
40
41 // TCP-Socket für die aktuelle Steuerungs-Verbindung
42 struct tcp_pcb* control_connection = (struct tcp_pcb*)NULL;
43
44 // Global location buffer
45 extern char *g_cLocalBuf;
46
47 //*****************************************************************************
48 //
49 // The buffer that holds the answer line.
50 //
51 //*****************************************************************************
52 char g_cNewConBuf[32];
53
54
55 /*
56  * Initialisierung der Kommunikation.
57  * Hier wird der Server-TCP-Socket und der UDP-Socket initialisiert.
58  * Zudem wird auch der FIFO für die Befehle initialisiert.
59  */
60 void control_init(void)
61 {
62  UARTprintf("  Initializing control...");
63
64  control_tpcb = tcp_new();
65  tcp_bind(control_tpcb, IP_ADDR_ANY, 56936);
66  control_tpcb = tcp_listen(control_tpcb);
67  tcp_accept(control_tpcb, &control_new_con);
68  UARTprintf("done\n");
69 }
70
71 /*
72  * "Interrupt" bei Erhalt einer neuen TCP-Verbindung.
73  * Diese Funktion wird vom lwIP-Stack ausgeführt, wenn eine
74  * neue TCP-Verbindung angefordert wird. Zunachst wird eine evtl.
75  * bestehende alte Verbindung getrennt. Anschließend wird die
76  * neue Verbindung angenommen.
77  */
78 err_t control_new_con(void* arg, struct tcp_pcb* newpcb, err_t err)
79 {
80  if(control_connection)
81   tcp_close(control_connection);
82  control_connection = newpcb;
83  tcp_accepted(newpcb);
84  tcp_recv(newpcb, &control_rx);
85  UARTprintf("New control connection established!\n");
86  if(control_connection){
```

```
87   telnet_write(" Enter Command (type <help> to see list of commands):\n");
88   sprintf(g_cNewConBuf, "%s: > \n", g_cLocalBuf);
89   telnet_write(g_cNewConBuf);
90   }
91   //
92   // Print a prompt to the console.
93   //
94   UARTprintf("\n Enter Command (type <help> to see list of commands):");
95   UARTprintf("\n%s: > ", g_cLocalBuf);
96   return ERR_OK;
97  }
98
99  /*
100  * "Interrupt" bei Erhalt von Daten Über den TCP-Socket.
101  * Diese Funktion wird vom lwIP-Stack ausgefÃ¼hrt, wenn neue
102  * Daten Ã¼ber den TCP-Socket eingetroffen sind.
103  * Die Daten werden hier zunächst im FIFO zwischengespeichert.
104  * AnschlieÃŸend wird die Funktion control_tick() ausgefÃ¼hrt, um
105  * die erhaltenen Daten auszuwerten.
106  */
107 err_t control_rx(void* arg, struct tcp_pcb* tpcb, struct pbuf* p, err_t err)
108 {
109
110   if(p == NULL)
111   {
112    UARTprintf("Control connection closed!\n");
113    control_connection = NULL;
114    return ERR_OK;
115   }
116
117   struct pbuf* current_p=p;
118
119
120   unsigned char* receive_data = (unsigned char*) current_p->payload;
121   unsigned long received_data_length = current_p->tot_len;
122
123
124   control_tick(receive_data, received_data_length);
125
126
127
128   tcp_recved(tpcb, p->tot_len);
129   pbuf_free(p);
130
131
132   return ERR_OK;
133  }
134
135  /*
136  * Auswertung der im FIFO enthaltenen Befehle.
137  * Es wird zunachst geprüft, ob eine vollstandige Zeile
138  * im FIFO vorhanden ist. Wenn ja, wird diese an die Funktion
139  * Cmd_interprete() übergeben. Wenn nicht, beendet
140  * sich die Funktion einfach.
141  */
142 void control_tick(unsigned char *data_check, unsigned long data_length)
143 {
144  long i;
145  tBoolean hadCommand = true;
146  char g_cEthGetBuf[64] = {0};
147
148  //OLLI
149  if (data_length >= 64){
150   UARTprintf("\n\nBad telnet command!\n");
151   telnet_write("\n\nBad telnet command!\n");
152   return;
153  }
154
155  while(hadCommand)
156  {
157
158   for(i = 0; i < data_length; i++)
159   {
160    char element = *data_check++;
161
162    if(element == '\n' || element =='\r' )
163    {
164
165     // There is a new command, execute
166
167     Cmd_interprete(g_cEthGetBuf);
168
169     hadCommand = false;
170
171     break;
```

```
    }

    g_cEthGetBuf[i] = element;
   }
  }
}

void telnet_write(const void *data){
 tcp_write(control_connection, data, strlen(data), TCP_WRITE_FLAG_COPY | TCP_WRITE_FLAG_MORE);
}


void telnet_immediate_write(const void *data){

 if(tcp_write(control_connection, data, strlen(data),TCP_WRITE_FLAG_COPY | TCP_WRITE_FLAG_MORE) != ERR_OK){
  tcp_output(control_connection);//Clear queue

  while(tcp_write(control_connection, data, strlen(data),TCP_WRITE_FLAG_COPY | TCP_WRITE_FLAG_MORE)){//Retry sending till ERR_OK is
       returned
   //tcp_output(control_connection);//Clear queue;#
   //UARTprintf("\nRetry sending!");
  }

 }else;
  tcp_output(control_connection);//Clear queue

}
```

Quellcode E.4: Anhang/c/discover.c

```
/*
Project:        battery_cycling_sw_v4
File:          discover.c

Auhtor:         Fabian Schwartau
Credits:        Thomas Wisniewski
          Matthias Schneider
          Tobias Steinmann
          Stellaris Ware

last modified:      2013/03/07

Project Status      Under Construction
Status:         running

CCS:        5.5.1.00031
Stellarisware:      8555

Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:      Discover the Board via Ethernet Broadcast message
          by answer with IP-address-package
*/




#include <string.h>
#include <stdio.h>
#include "utils/lwiplib.h"
#include "utils/uartstdio.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "inc/lm3s9b92.h"
#include "stdio.h"
#include "lwip/udp.h"



void discover_init(void);
void discover_rx(void* arg, struct udp_pcb* upcb, struct pbuf* p, struct ip_addr* addr, u16_t port);
void ipToString(char* string, unsigned long ip, int maxLen);

// UDP-Socket für die UDP-Nachrichten
struct udp_pcb* g_upcb;

/*
 * Initialisierung des UDP-Sockets.
 */
```

```
54 void discover_init(void)
55 {
56  volatile unsigned long ulLoop;
57  UARTprintf("  Initializing discover...");
58  g_upcb = udp_new();
59  err_t error = udp_bind(g_upcb, IP_ADDR_ANY, 56936);
60  if(error != ERR_OK)
61  {
62      UARTprintf("failed: Unable to bind udp socket for discover: %d\n", error);
63   return;
64  }
65  udp_recv(g_upcb, &discover_rx, (void*)0);
66  UARTprintf("done\n");
67 }
68
69 /*
70  * "Interrupt" beim Erhalten eines neuen Pakets.
71  * Diese Funktion wird durch den lwIP-Stack ausgeführt, wenn ein
72  * neues UDP-Paket auf dem Socket g_upcb angekommen ist.
73  * Die Funktion antwortet jedem Paket mit der IP-Adresse des Moduls.
74  */
75 void discover_rx(void* arg, struct udp_pcb* upcb, struct pbuf* p, struct ip_addr* addr, u16_t port)
76 {
77  char buffer[30];
78  UARTprintf("Got discovery packet:\n");
79     UARTprintf("  length: %d\n", p->len);
80     UARTprintf("  payload: %s\n", p->payload);
81
82
83
84  //udp_connect(g_upcb, addr, port);
85  ipToString(buffer, lwIPLocalIPAddrGet(), 29);
86  struct pbuf* ipPacket = pbuf_alloc(PBUF_TRANSPORT, strlen(buffer), PBUF_RAM);
87  strcpy((char*)ipPacket->payload, buffer);
88  udp_sendto(g_upcb, ipPacket, addr, port);
89
90
91  UARTprintf("  SRC IP: %d.%d.%d.%d\n", addr->addr & 0xff, (addr->addr >> 8) & 0xff, (addr->addr >> 16) & 0xff, (addr->addr >> 24) & 0
       xff);
92  UARTprintf("  SRC Port: %d\n", port);
93  UARTprintf("  Answer sent\n");
94
95
96  /*UARTprintf("  SRC2 IP: %d.%d.%d.%d\n", upcb->remote_ip.addr & 0xff, (upcb->remote_ip.addr >> 8) & 0xff, (upcb->remote_ip.addr >> 16)
       & 0xff, (upcb->remote_ip.addr >> 24) & 0xff);
97  UARTprintf("  SRC2 Port: %d\n", upcb->remote_port);
98  UARTprintf("  DST IP: %d.%d.%d.%d\n", upcb->local_ip.addr & 0xff, (upcb->local_ip.addr >> 8) & 0xff, (upcb->local_ip.addr >> 16) & 0
       xff, (upcb->local_ip.addr >> 24) & 0xff);
99  UARTprintf("  DST Port: %d\n", upcb->local_port);*/
100  pbuf_free(p);
101  pbuf_free(ipPacket);
102 }
103
104 /*
105  * Funktion zur Konvertierung einer in einem unsigned long gespeicherten
106  * IP-Adresse in einen String.
107  */
108 void ipToString(char* string, unsigned long ip, int maxLen)
109 {
110  sprintf(string, "%d.%d.%d.%d", ip & 0xff, (ip >> 8) & 0xff, (ip >> 16) & 0xff, (ip >> 24) & 0xff);
111 }
```

Quellcode E.5: Anhang/c/display.c

```
1  /*
2  Project:       battery_cycling_sw_v4
3  File:          display.c
4
5  Auhtor:        Thomas Wisniewski
6  Credits:       Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Johannes Roehn
10          Oliver Schmidt
11          Stellaris Ware
12
13 last modified:      2015/04/28
14
15 Project Status      Under Construction
16 Status:         running
17
18 CCS:        5.5.1.00031
19 Stellarisware:      8555
```

```
20
21 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
22          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
23          NTC-Connectors, MAX3232 and Suplly Circuits
24
25 Description:       Source Code LED Display
26
27 */
28
29
30
31 /*****************************************************************************
32 *
33 * Own Includings
34 *
35 *****************************************************************************/
36
37 #include <utils/lwiplib.h>
38 #include "inc/hw_ints.h"
39 #include "inc/hw_memmap.h"
40 #include "inc/hw_types.h"
41 #include "driverlib/debug.h"
42 #include "driverlib/gpio.h"
43 #include "driverlib/interrupt.h"
44 #include "driverlib/pin_map.h"
45 #include "driverlib/rom.h"
46 #include "driverlib/sysctl.h"
47 #include "driverlib/uart.h"
48 #include "utils/uartstdio.h"
49 #include "driverlib/timer.h"
50 #include "header/display.h"
51 #include "stdio.h"
52 #include "header/config.h"
53 #include "header/mycmdline.h"
54
55
56
57 //*****************************************************************************
58 //Extern variables for Clock
59 //*****************************************************************************
60 extern volatile unsigned long clock_msec;
61 extern volatile unsigned long clock_sec;
62 extern volatile unsigned long clock_min;
63 extern volatile unsigned long clock_hour;
64 extern volatile unsigned long clock_day;
65 extern volatile unsigned long clock_month;
66 extern volatile unsigned long clock_year;
67 //*****************************************************************************
68 // extern variables for calibration
69 //*****************************************************************************
70 extern volatile short g_calibrationChannel;
71 extern volatile short g_calibrationVoltage;
72 extern volatile char g_Flag_calibrationRedo;
73 extern volatile char g_Flag_calibrationStop;
74 //*****************************************************************************
75 //
76 // Extern variables for measure data
77 //
78 //*****************************************************************************
79 extern volatile unsigned int g_iCellVoltages[];
80 extern volatile int g_iCellTemperature[];
81 extern volatile int g_iCurrent;
82 extern volatile int gBat_charge;
83
84 extern volatile unsigned int timer2second;
85 extern volatile unsigned int cycle_active;
86 extern volatile unsigned int measurement_active;
87 extern volatile unsigned int current_cycle_step;
88 extern const char *cycle_types[];
89
90 void display_init(void)
91 {
92  UARTprintf("Initializing Display...");
93
94     //
95     // Enable the peripherals used by this example.
96     //
97     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
98     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
99
100    GPIOPinTypeGPIOOutput(LEDPORTCTRL, RS | RW | E ); // set RS | R/W | E as digital output
101    GPIOPinTypeGPIOOutput(LEDPORTDATA, D4 | D5 | D6 | D7); // set  D4 | D5 | D6 | D7 as digital output
102
103    GPIOPadConfigSet(LEDPORTCTRL, RS | RW | E, GPIO_STRENGTH_8MA, GPIO_PIN_TYPE_STD_WPU);//set outputs to max 8 mA for better safety
104    GPIOPadConfigSet(LEDPORTDATA, D4 | D5 | D6 | D7, GPIO_STRENGTH_8MA, GPIO_PIN_TYPE_STD_WPU);
```

```
105
106
107     //*************************************************************
108     //Display Reset
109     GPIOPinWrite(LEDPORTCTRL, RS | RW | E, 0x0); // set RS | R/W | E to logic "0"
110     GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7 , 0x0); // set  D4 | D5 | D6 | D7 to logic "0"
111
112  // ca. 20 ms delay für Display reset
113     unsigned long ul_delay_count = 200000;
114  while (ul_delay_count) ul_delay_count--;
115  //-------------------------------------
116
117
118  //
119  //Function set: 8 bit Interface = 0x30
120  //
121  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
122  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
123
124  // ca. 0.45 ms delay warten Enable High
125  ul_delay_count = 4500;
126  while (ul_delay_count) ul_delay_count--;
127
128  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
129
130  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);  // Reset pins
131
132  // ca. 10 ms delay warten für nächsten Befehl
133  ul_delay_count = 100000;
134  while (ul_delay_count) ul_delay_count--;
135
136  //
137  //Function set: 8 bit Interf445ace = 0x30
138  //
139  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
140  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
141
142  // ca. 0.45 ms delay warten Enable High
143  ul_delay_count = 4500;
144  while (ul_delay_count) ul_delay_count--;
145
146  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
147
148  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0); // Reset pins
149
150  // ca. 5 ms delay warten für nächsten Befehl
151  ul_delay_count = 50000;
152  while (ul_delay_count) ul_delay_count--;
153
154
155  //
156  //Function set: 8 bit Interface = 0x30
157  //
158  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
159  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
160
161  // ca. 0.45 ms delay warten Enable High
162  ul_delay_count = 4500;
163  while (ul_delay_count) ul_delay_count--;
164
165  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
166
167  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0); // Reset pins
168
169  // ca. 2 ms delay warten für nächsten Befehl
170  ul_delay_count = 20000;
171  while (ul_delay_count) ul_delay_count--;
172
173
174
175  //
176  //Function set: 4 bit Interface = 0x20
177  //
178  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
179  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x2);
180
181  // ca. 0.45 ms delay warten Enable High
182  ul_delay_count = 4500;
183  while (ul_delay_count) ul_delay_count--;
184
185  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
186  GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);  // Reset pins
187
188  // ca. 1 ms delay warten für nächsten Befehl
189  ul_delay_count = 10000;
```

```
190   while (ul_delay_count) ul_delay_count--;
191
192   clear_display();
193
194   //Function set: 4 bit Interface, 2(4) rows, 5x7dot = 0x28
195   LEDcommand(0x28);
196   //Display on, cursor off, blink off = 0x0C
197   LEDcommand(0x0C);
198   //Entry Mode set: Increment, No shift = 0x06
199   LEDcommand(0x06);
200
201
202   //
203   // Enable Timer 1
204   //
205   TimerEnable(TIMER1_BASE, TIMER_A);
206
207
208   UARTprintf("done\n");
209 }
210
211 void display_reinit(void){
212
213    //*********************************************************
214       //Display Reset
215       GPIOPinWrite(LEDPORTCTRL, RS | RW | E, 0x0); // set RS | R/W | E to logic "0"
216       GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7 , 0x0); // set  D4 | D5 | D6 | D7 to logic "0"
217
218    // ca. 20 ms delay für Display reset
219       unsigned long ul_delay_count = 200000;
220    while (ul_delay_count) ul_delay_count--;
221    //--------------------------------
222
223
224    //
225    //Function set: 8 bit Interface = 0x30
226    //
227    GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
228    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
229
230    // ca. 0.45 ms delay warten Enable High
231    ul_delay_count = 4500;
232    while (ul_delay_count) ul_delay_count--;
233
234    GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
235
236    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);  // Reset pins
237
238    // ca. 10 ms delay warten für nächsten Befehl
239    ul_delay_count = 100000;
240    while (ul_delay_count) ul_delay_count--;
241
242    //
243    //Function set: 8 bit Interface = 0x30
244    //
245    GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
246    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
247
248    // ca. 0.45 ms delay warten Enable High
249    ul_delay_count = 4500;
250    while (ul_delay_count) ul_delay_count--;
251
252    GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
253
254    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0); // Reset pins
255
256    // ca. 5 ms delay warten für nächsten Befehl
257    ul_delay_count = 50000;
258    while (ul_delay_count) ul_delay_count--;
259
260
261    //
262    //Function set: 8 bit Interface = 0x30
263    //
264    GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
265    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
266
267    // ca. 0.45 ms delay warten Enable High
268    ul_delay_count = 4500;
269    while (ul_delay_count) ul_delay_count--;
270
271    GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
272
273    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0); // Reset pins
274
```

```
275     // ca. 2 ms delay warten für nächsten Befehl
276     ul_delay_count = 20000;
277     while (ul_delay_count) ul_delay_count--;
278
279     //
280     //Function set: 4 bit Interface = 0x20
281     //
282     GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
283     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x2);
284
285     // ca. 0.45 ms delay warten Enable High
286     ul_delay_count = 4500;
287     while (ul_delay_count) ul_delay_count--;
288
289     GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
290     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);  // Reset pins
291
292     // ca. 1 ms delay warten für nächsten Befehl
293     ul_delay_count = 10000;
294     while (ul_delay_count) ul_delay_count--;
295
296     clear_display();
297
298     //Function set: 4 bit Interface, 2(4) rows, 5x7dot = 0x28
299     LEDcommand(0x28);
300     //Display on, cursor off, blink off = 0x0C
301     LEDcommand(0x0C);
302     //Entry Mode set: Increment, No shift = 0x06
303     LEDcommand(0x06);
304 }
305
306
307 void buttons_init(void)
308 {
309  UARTprintf("Initializing Buttons...");
310
311  //
312     // Enable the GPIO that is used for the first push button (GPIOF_PIN0).
313     //
314     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
315     ROM_GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
316     ROM_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA,
317                          GPIO_PIN_TYPE_STD_WPU);
318
319  //
320     // Enable the GPIO that is used for the second push button (GPIOD_PIN1).
321     //
322     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
323     ROM_GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_1);
324     ROM_GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_STRENGTH_2MA,
325                          GPIO_PIN_TYPE_STD_WPU);
326
327  //
328     // Enable the GPIO that is used for the third push button (GPIOC_PIN4).
329     //
330     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
331     ROM_GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_4);
332     ROM_GPIOPadConfigSet(GPIO_PORTC_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
333                          GPIO_PIN_TYPE_STD_WPU);
334
335
336  //
337     // Enable the GPIO that is used for the forth push button (GPIOH_PIN5).
338     //
339     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
340     ROM_GPIOPinTypeGPIOInput(GPIO_PORTH_BASE, GPIO_PIN_5);
341     ROM_GPIOPadConfigSet(GPIO_PORTH_BASE, GPIO_PIN_5, GPIO_STRENGTH_2MA,
342                          GPIO_PIN_TYPE_STD_WPU);
343
344
345     //
346     // Enable interrupts
347     //
348     ROM_IntEnable(INT_GPIOF);
349     GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_RISING_EDGE);  //trigger on rising edge
350     GPIOPinIntEnable(GPIO_PORTF_BASE,GPIO_PIN_0); //enable interrupt on PF0
351
352     ROM_IntEnable(INT_GPIOD);
353     GPIOIntTypeSet(GPIO_PORTD_BASE,GPIO_PIN_1,GPIO_RISING_EDGE);  //trigger on rising edge
354     GPIOPinIntEnable(GPIO_PORTD_BASE,GPIO_PIN_1); //enable interrupt on PD1
355
356     ROM_IntEnable(INT_GPIOC);
357     GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_4,GPIO_RISING_EDGE);  //trigger on rising edge
358     GPIOPinIntEnable(GPIO_PORTC_BASE,GPIO_PIN_4); //enable interrupt on PC4
359
```

```
360     ROM_IntEnable(INT_GPIOH);
361     GPIOIntTypeSet(GPIO_PORTH_BASE,GPIO_PIN_5,GPIO_RISING_EDGE);  //trigger on rising edge
362     GPIOPinIntEnable(GPIO_PORTH_BASE,GPIO_PIN_5); //enable interrupt on PC4
363
364     // Set statemachine to state Start
365     handlerState = START;
366  UARTprintf("done\n");
367 }
368
369 //*****************************************************************************
370 //
371 // This is the handler for the GPIO PORT F PIN 0 interrupt. (Button1) linker Knopf
372 //
373 //*****************************************************************************
374 void GPIOF00Handler(void){
375
376  //Disable Interrupts
377  TimerDisable(TIMER1_BASE, TIMER_A);
378  ROM_IntDisable(INT_GPIOF);
379  GPIOPinIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);
380
381  clear_display();
382
383   // Switch state
384   if(handlerState == CELLA)
385   handlerState = CELLD;
386
387   else if(handlerState == CELLD)
388   handlerState = CELLC;
389
390   else if(handlerState == CELLC)
391   handlerState = CELLB;
392
393   else if(handlerState == CELLB)
394   handlerState = CELLA;
395
396   else if(handlerState == MEAS)
397    Cmd_interprete("start");
398
399   else if(handlerState == MEASSTOP){
400    Cmd_interprete("stop");
401    handlerState = MEAS;
402   }
403
404   else if(handlerState == CALPULL){
405    //handlerState wird in config.c calibrate_adc_quadratisch_waitLowToHigh() abgearbeitet
406   g_Flag_calibrationStop=1;
407   }
408
409   else if(handlerState == CALPLUG){
410    //handlerState wird in config.c calibrate_adc_quadratisch_waitLowToHigh() abgearbeitet
411   g_Flag_calibrationStop=1;
412   }
413
414   else if(handlerState ==  CALMEAS){
415    //handlerState wird in config.c calibrate_adc_quadratisch_waitLowToHigh() abgearbeitet
416   g_Flag_calibrationStop=1;
417   }
418
419
420
421  //Enable Interrupts
422  ROM_IntEnable(INT_GPIOF);
423  TimerEnable(TIMER1_BASE, TIMER_A);
424 }
425
426 //*****************************************************************************
427 //
428 // This is the handler for the GPIO PORT D PIN 1 interrupt. (Button2) Mittelerer Knopf
429 //
430 //*****************************************************************************
431 void GPIOD01Handler(void){
432
433  //Disable Interrupts
434  TimerDisable(TIMER1_BASE, TIMER_A);
435  GPIOPinIntClear(GPIO_PORTD_BASE,GPIO_PIN_1);
436
437
438  clear_display();
439
440   // Switch state
441   switch(handlerState){
442    case ERROR:    handlerState = START; break;
443    case ERRCURR: handlerState = START; break;
444    case ERRVOLT: handlerState = START; break;
```

```
445     case ERRTIME: handlerState = START; break;
446     case ERRTEMP: handlerState = START; break;
447     case ERRCHARGE:handlerState = START; break;
448     case START: handlerState = CLOCK; break;
449     case CLOCK: handlerState = MEAS; break;
450     case MEAS:
451     case MEASSTOP: if(cycle_active)
452           handlerState = CURRSTEP;
453            else
454              handlerState = CELLA;
455            break;
456     case CURRSTEP: handlerState = CELLA; break;
457     case CELLA: handlerState = CURRENT;
458     case CELLB: handlerState = CURRENT;
459     case CELLC: handlerState = CURRENT;
460     case CELLD: handlerState = CURRENT; break;
461     case CURRENT: handlerState = IP; break;
462     case IP: handlerState = CONF; break;
463     case CONF: handlerState = START; break;
464
465    }
466
467
468
469   //Enable Interrupts
470   TimerEnable(TIMER1_BASE, TIMER_A);
471  }
472
473
474  //*****************************************************************************
475  //
476  // This is the handler for the GPIO PORT C PIN 4 interrupt. (Button3) rechter Knopf
477  //
478  //*****************************************************************************
479  void GPIOC04Handler(void){
480   //Disable Interrupts
481   TimerDisable(TIMER1_BASE, TIMER_A);
482   ROM_IntDisable(INT_GPIOC);
483   GPIOPinIntClear(GPIO_PORTC_BASE,GPIO_PIN_4);
484
485   clear_display();
486
487    // Switch state
488    if(handlerState == CELLA)
489    handlerState = CELLB;
490
491    else if(handlerState == CELLB)
492    handlerState = CELLC;
493
494    else if(handlerState == CELLC)
495    handlerState = CELLD;
496
497    else if(handlerState == CELLD)
498    handlerState = CELLA;
499
500    else if(handlerState == MEAS)
501    handlerState = MEASSTOP;
502
503    else if(handlerState == MEASSTOP)
504    handlerState = MEAS;
505
506    else if(handlerState == CALPULL){
507     //handlerState wird in config.c calibrate_adc_quadratisch_waitLowToHigh() abgearbeitet
508    g_Flag_calibrationRedo=1;
509    }
510
511
512   //Enable Interrupts
513   ROM_IntEnable(INT_GPIOC);
514   TimerEnable(TIMER1_BASE, TIMER_A);
515  }
516
517  //*****************************************************************************
518  //
519  // This is the handler for the GPIO PORT H PIN 5 interrupt. (Button4)
520  // NO FUNCTION YET!
521  //
522  //*****************************************************************************
523  void GPIOH05Handler(void){
524   //Disable Interrupts
525   TimerDisable(TIMER1_BASE, TIMER_A);
526   ROM_IntDisable(INT_GPIOH);
527   GPIOPinIntClear(GPIO_PORTH_BASE,GPIO_PIN_5);
528
529   clear_display();
```

```
530
531   //Enable Interrupts
532   ROM_IntEnable(INT_GPIOH);
533   TimerEnable(TIMER1_BASE, TIMER_A);
534 }
535
536
537 //*****************************************************************************
538 //
539 // The  handler for the Timer1A interrupt.
540 // Refresh the Content of the LED-Display dependen on current state
541 //
542 //*****************************************************************************
543 void Timer1IntHandler(void){
544
545   unsigned long ulIPAddress;
546   char display_buffer[21];
547   int bat_voltage = 0;
548   int i = 0;
549
550
551   //Write content to Displays
552   switch(handlerState) {
553
554     case ERROR:
555
556       write_to_display("ERROR in config.txt!", 0, 0);
557       write_to_display("    default values   ", 1, 0);
558       write_to_display("       loaded       ", 2, 0);
559       write_to_display("***** CONTINUE *****", 3 ,0);
560       break;
561
562     case ERRCHARGE:
563
564       write_to_display(" Cycling canceled!  ", 0, 0);
565       write_to_display("  maximum charge    ", 1, 0);
566       write_to_display("       exceeded     ", 2, 0);
567       write_to_display("***** CONTINUE *****", 3 ,0);
568       break;
569
570     case ERRCURR:
571
572       write_to_display(" Cycling canceled!  ", 0, 0);
573       write_to_display("  maximum current   ", 1, 0);
574       write_to_display("       exceeded     ", 2, 0);
575       write_to_display("***** CONTINUE *****", 3 ,0);
576       break;
577
578     case ERRVOLT:
579
580       write_to_display(" Cycling canceled!  ", 0, 0);
581       write_to_display("   voltage limits   ", 1, 0);
582       write_to_display("       exceeded     ", 2, 0);
583       write_to_display("***** CONTINUE *****", 3 ,0);
584       break;
585
586     case ERRTIME:
587
588       write_to_display(" Cycling finished!  ", 0, 0);
589       write_to_display(" maximum cycle time ", 1, 0);
590       write_to_display("       reached      ", 2, 0);
591       write_to_display("***** CONTINUE *****", 3 ,0);
592       break;
593
594     case ERRTEMP:
595
596       write_to_display(" Cycling canceled!  ", 0, 0);
597       write_to_display("maximum temperature ", 1, 0);
598       write_to_display("       exceeded     ", 2, 0);
599       write_to_display("***** CONTINUE *****", 3 ,0);
600       break;
601
602     case START:
603
604       write_to_display("********************", 0, 0);
605       write_to_display("***     Battery    ***", 1, 0);
606       write_to_display("**  Cycle Machine  **", 2, 0);
607       write_to_display("*****    MENU    *****", 3 ,0);
608       break;
609
610     case CLOCK:
611
612       sprintf(display_buffer, "      %02d:%02d:%02d      ", clock_hour, clock_min, clock_sec);
613       write_to_display(display_buffer, 0, 0);
614       sprintf(display_buffer, "     %02d.%02d.%04d     ", clock_day, clock_month, clock_year);
```

```
615     write_to_display(display_buffer, 1, 0);
616     write_to_display("SW: 20150428...ZG1  ", 2, 0);
617     write_to_display("       MENU        ", 3 ,0);
618     break;
619
620   case MEAS:
621
622     sprintf(display_buffer, "Measurement Cycling:");
623     write_to_display(display_buffer, 0, 0);
624
625     if(measurement_active && cycle_active)
626      sprintf(display_buffer, "active      active  ");
627     else if(measurement_active && !cycle_active)
628      sprintf(display_buffer, "active      not act.");
629     else
630      sprintf(display_buffer, "not act.    not act.");
631     write_to_display(display_buffer, 1, 0);
632
633
634     write_to_display("START   MENU    STOP", 3 ,0);
635     break;
636
637   case MEASSTOP:
638
639     sprintf(display_buffer, "Measurement Cycling:");
640     write_to_display(display_buffer, 0, 0);
641
642     sprintf(display_buffer, "    Confirm STOP    ");
643
644     write_to_display(display_buffer, 1, 0);
645
646
647     write_to_display(" YES    MENU     NO ", 3 ,0);
648     break;
649
650   case CELLA:
651
652     for(i = 0; i <= 2; i++){
653      sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1, g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
654                       , g_iCellTemperature[i]/10, abs(g_iCellTemperature[i]%10));
655      write_to_display(display_buffer, i, 0);
656     }
657
658     write_to_display("<PREV   MENU   NEXT>", 3 ,0);
659     break;
660
661   case CELLB:
662
663     for(i = 3; i <= 5; i++){
664      sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1, g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
665                       , g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
666      write_to_display(display_buffer, i-3, 0);
667     }
668
669     write_to_display("<PREV   MENU   NEXT>", 3 ,0);
670     break;
671
672   case CELLC:
673
674     for(i = 6; i <= 8; i++){
675      sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1, g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
676                       , g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
677      write_to_display(display_buffer, i-6, 0);
678     }
679
680     write_to_display("<PREV   MENU   NEXT>", 3 ,0);
681     break;
682
683   case CELLD:
684
685     for(i = 9; i <= 11; i++){
686      sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1, g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
687                       , g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
688      write_to_display(display_buffer, i-9, 0);
689     }
690
691     write_to_display("<PREV   MENU   NEXT>", 3 ,0);
692     break;
693
694   case CURRENT:{
695
696     bat_voltage = 0;
697     for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
698      bat_voltage += g_iCellVoltages[i];
699     bat_voltage /= 1000;//mV
```

```
700
701     sprintf(display_buffer, "Voltage:   %02d.%03d V ",bat_voltage/1000, bat_voltage%1000);
702     write_to_display(display_buffer, 0, 0);
703
704     if(g_iCurrent < 0)
705       sprintf(display_buffer, "Current: -%03d.%03d A ", abs(g_iCurrent/1000), abs(g_iCurrent%1000));
706     else
707       sprintf(display_buffer, "Current:  %03d.%03d A ", g_iCurrent/1000, g_iCurrent%1000);
708     write_to_display(display_buffer, 1, 0);
709
710     if(gBat_charge < 0)
711       sprintf(display_buffer, "Charge : -%03d.%03d Ah", abs(gBat_charge/3600000), abs((gBat_charge/3600)%1000));
712     else
713       sprintf(display_buffer, "Charge :  %03d.%03d Ah", gBat_charge/3600000, (gBat_charge/3600)%1000);
714     write_to_display(display_buffer, 2, 0);
715
716     write_to_display("        MENU        ", 3 ,0);
717     break;
718   }
719   case IP:
720
721     sprintf(display_buffer, "     IP-Address:    ");
722     write_to_display(display_buffer, 0, 0);
723     ulIPAddress = lwIPLocalIPAddrGet();
724     sprintf(display_buffer, "   %03d.%03d.%03d.%03d  "  , ulIPAddress & 0xff,
725       (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
726       (ulIPAddress >> 24) & 0xff);
727     write_to_display(display_buffer, 1, 0);
728     write_to_display("        MENU        ", 3 ,0);
729     break;
730
731   case CONF:
732
733     sprintf(display_buffer, "Quantity cells:%02d  ", config.quantity_cells);
734     write_to_display(display_buffer, 0, 0);
735     sprintf(display_buffer, "Cyclestep/msec:%05d "  , config.cyclestep);
736     write_to_display(display_buffer, 1, 0);
737     if(config.measuretime > 0)
738       sprintf(display_buffer, "Time left/min :%05d"  , (config.measuretime - timer2second)/60); // umrechnung sek in min
739     else
740       sprintf(display_buffer, "Time left(min):  inf");
741     write_to_display(display_buffer, 2, 0);
742     write_to_display("        MENU        ", 3 ,0);
743     break;
744
745   case CURRSTEP:
746
747     sprintf(display_buffer, "Current step %d/%d", current_cycle_step, config.step_quantity);
748     write_to_display(display_buffer, 0, 0);
749     sprintf(display_buffer, "type %s"  , cycle_types[config.cycle_steps[current_cycle_step].type]);
750     write_to_display(display_buffer, 1, 0);
751     if(config.cycle_steps[current_cycle_step].type == 1)
752       sprintf(display_buffer, "%d min relay %d"  ,config.cycle_steps[current_cycle_step].param ,config.cycle_steps[current_cycle_step].
            relay);
753     else if(config.cycle_steps[current_cycle_step].type == 6 || config.cycle_steps[current_cycle_step].type == 7)
754       sprintf(display_buffer, "loop" );
755     else
756       sprintf(display_buffer, "%d mV relay %d"  ,config.cycle_steps[current_cycle_step].param ,config.cycle_steps[current_cycle_step].
            relay);
757
758     write_to_display(display_buffer, 2, 0);
759     write_to_display("        MENU        ", 3 ,0);
760     break;
761
762   case CALPLUG:
763     sprintf(display_buffer, "Ch.:%02d, V.:%02d       ",g_calibrationChannel+1,g_calibrationVoltage);
764       write_to_display("********************", 0, 0);
765     write_to_display(display_buffer, 1, 0);
766     write_to_display("    plug in cables  ", 2, 0);
767     write_to_display("STOP ***************", 3 ,0);
768     break;
769
770   case CALMEAS:
771       write_to_display("********************", 0, 0);
772     write_to_display("     measurement     ", 1, 0);
773     write_to_display("        WAIT         ", 2, 0);
774     write_to_display("STOP ***************", 3 ,0);
775     break;
776
777   case CALPULL:
778       write_to_display("********************", 0, 0);
779     write_to_display("  measurement DONE  ", 1, 0);
780     write_to_display("    pull cables     ", 2, 0);
781     write_to_display("STOP ********* =>Ch1", 3 ,0);
782     break;
```

```
783
784
785
786
787  }
788
789  //
790  // Clear the timer interrupt.
791     //
792    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
793 }
794
795
796 //*****************************************************************************
797 //
798 // Clear the LED-Display
799 //
800 //*****************************************************************************
801 void clear_display(void){
802
803
804  LEDcommand(0x01);
805
806
807 }
808
809 //*****************************************************************************
810 //
811 // Write string to Display by row and column
812 //
813 //*****************************************************************************
814 void write_to_display(char* inputText, unsigned char row, unsigned char col) {
815  unsigned char address_d = 0;  // address of the data in the screen.
816
817  //define address
818  switch(row)
819   {
820    case 0: address_d = 0x80 + col;  // at zeroth row
821    break;
822    case 1: address_d = 0xC0 + col;  // at first row
823    break;
824    case 2: address_d = 0x94 + col;  // at second row
825    break;
826    case 3: address_d = 0xD4 + col;  // at third row
827    break;
828    default: address_d = 0x80 + col; // returns to first row if invalid row number is detected
829    break;
830   }
831
832  //set position by address
833  set_position(address_d);
834
835  // Place a string, letter by letter.
836  while(*inputText)
837   write_char(*inputText++);
838 }
839
840 //*****************************************************************************
841 //
842 // Set the position by address
843 //
844 //*****************************************************************************
845 void set_position(unsigned char address) {
846
847  unsigned long ul_delay_count;
848
849  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
850  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0xf0) >> 4);//data
851
852  // ca. 1 mus delay warten Enable High
853  ul_delay_count = 4500;
854  while (ul_delay_count) ul_delay_count--;
855
856  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
857  //10ns hold time
858  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
859
860
861  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
862  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0x0f));//data
863
864  // ca. 1 mus delay warten Enable High
865  ul_delay_count = 4500;
866  while (ul_delay_count) ul_delay_count--;
867
```

```
868  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
869  //10ns hold time
870  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
871
872 }
873
874 //*****************************************************************************
875 //
876 // Write character to Display
877 //
878 //*****************************************************************************
879 void write_char(unsigned char inputData) {
880
881  unsigned long ul_delay_count;
882
883  GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);      // Enable the Display and writeoperation
884  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0xf0) >> 4);//data
885
886  // ca. 1 mus delay warten Enable High
887  ul_delay_count = 4500;
888  while (ul_delay_count) ul_delay_count--;
889
890  GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);     // Disable the Display
891  //10 ns hold time
892  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
893
894
895  GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);      // Enable the Display and writeoperation
896  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0x0f));//data
897
898  // ca. 1 mus delay warten Enable High
899  ul_delay_count = 4500;
900  while (ul_delay_count) ul_delay_count--;
901
902  GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);     // Disable the Display
903  //10ns hold time
904  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
905
906 }
907
908 //*****************************************************************************
909 //
910 // Send Command to Display
911 //
912 //*****************************************************************************
913 void LEDcommand(unsigned char command){
914
915  unsigned long ul_delay_count;
916
917  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
918  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0xf0) >> 4);//data
919
920  // ca. 1 mus delay warten Enable High
921  ul_delay_count = 4500;
922  while (ul_delay_count) ul_delay_count--;
923
924  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
925  //10ns hold time
926  GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0); // Reset pins
927
928  // ca. 100 mus delay warten auf nächsten Befehl
929  ul_delay_count = 1000;
930  while (ul_delay_count) ul_delay_count--;
931
932
933  GPIOPinWrite(LEDPORTCTRL, E, E);      // Enable the Display
934  GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0x0f));//data
935
936  // ca. 1 mus delay warten Enable High
937  ul_delay_count = 4500;
938  while (ul_delay_count) ul_delay_count--;
939
940  GPIOPinWrite(LEDPORTCTRL, E, 0x00);     // Disable the Display
941  //10ns hold timne
942  GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0); // Reset pins
943
944  // ca. 1 ms delay warten auf  nächsten Befehl
945  ul_delay_count = 10000;
946  while (ul_delay_count) ul_delay_count--;
947 }
```

Quellcode E.6: Anhang/c/ethernet.c

```
/*
Project:        battery_cycling_sw_v4
File:           ethernet.c

Auhtor:         Thomas Wisniewski
Credits:        Matthias Schneider
          Tobias Steinmann
          Fabian Schwartau
          Johannes Roehn
          Stellaris Ware

last modified:      2014/01/13

Project Status      Under Construction
Status:         running

CCS:        5.5.1.00031
Stellarisware:      8555

Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:      Initialize Ethernet with lwip. Define IP-Address,
          call initialization of TCP and UDP.
*/


//*****************************************************************************
//
// Includings
//
//*****************************************************************************/
#include <utils/lwiplib.h>
#include <utils/uartstdio.h>
#include <utils/locator.h>
#include <inc/hw_types.h>
#include <inc/hw_ints.h>
#include <inc/hw_memmap.h>
#include <driverlib/rom.h>
#include <driverlib/sysctl.h>
#include <driverlib/gpio.h>
#include <driverlib/interrupt.h>
#include "driverlib/systick.h"
#include <lwip/ip_addr.h>
#include "stdio.h"
#include <inc/lm3s9b92.h>

//*****************************************************************************
//
// Own Includings
//
//*****************************************************************************/
#include "header/control.h"
#include "header/discover.h"
#include "header/config.h"
#include "header/ethernet.h"

int ethernet_init(void);
void lwIPHostTimerHandler(void);

// Zur Anzeige einer Aktivität in der Konsole
static char g_pcTwirl[4] = { '\\', '|', '/', '-' };
static unsigned long g_ulTwirlPos = 0;

// TCP-Socket für die aktuelle Steuerungs-Verbindung
extern struct tcp_pcb* control_connection;

// Aktuelle IP Adresse
static unsigned long g_ulLastIPAddr = 0;

unsigned long ulUser0, ulUser1;

//MAC Adresse aus define auslesen DEBUG
unsigned long long macAddress = MACADDR;


// MAC Adresse für weitere Verwendung im Programm
unsigned char pucMACArray[8];


/*
 * Initialize Ethernetfunction
 */
int init_ethernet(void)
```

```
86  {
87    unsigned long ul_delay_count;
88
89    UARTprintf("Initializing Ethernet:\n");
90
91    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOE;
92    volatile unsigned long ulLoop;
93    ulLoop = SYSCTL_RCGC2_R;
94
95    GPIO_PORTE_DIR_R = 0x00;  // Auf Input setzen
96    GPIO_PORTE_DEN_R = 0xFF;  // Einschalten der GPIOs
97    GPIO_PORTE_PUR_R = 0xFF; // Pull Ups aktivieren
98
99    // Einschalten und Resetten des Eternet Controllers
100   ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ETH);
101   ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_ETH);
102
103
104   ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
105   GPIOPinConfigure(GPIO_PF2_LED1);
106   GPIOPinConfigure(GPIO_PF3_LED0);
107   GPIOPinTypeEthernetLED(GPIO_PORTF_BASE, GPIO_PIN_2 | GPIO_PIN_3);
108
109 //   // Lesen der MAC-Adresse aus den User-Registern
110 //   // Diese scheint ab Werk programmiert zu sein
111 //   ROM_FlashUserGet(&ulUser0, &ulUser1);
112 //   if((ulUser0 == 0xffffffff) || (ulUser1 == 0xffffffff))
113 //   {
114 //    // We should never get here.  This is an error if the MAC address has
115 //    // not been programmed into the device.  Exit the program
116 //    UARTprintf("  MAC Address Not Programmed!\n");
117 //    return 0;
118 //   }
119 //
120 //   // Umwandlung der 24/24 MAC Adresse aus dem NV RAM in eine 32/16 MAC
121 //   // Adresse. Anchließend wird die MAC dem Ethernet Controller
122 //   // übergeben
123 //   pucMACArray[0] = ((ulUser0 >>  0) & 0xff);
124 //   pucMACArray[1] = ((ulUser0 >>  8) & 0xff);
125 //   pucMACArray[2] = ((ulUser0 >> 16) & 0xff);
126 //   pucMACArray[3] = ((ulUser1 >>  0) & 0xff);
127 //   pucMACArray[4] = ((ulUser1 >>  8) & 0xff);
128 //   pucMACArray[5] = ((ulUser1 >> 16) & 0xff);
129
130   //Manuelles definieren der  MAC Adresse als Locally Administered Adress
131   pucMACArray[0] = ((macAddress >> 40) & 0xff);
132   pucMACArray[1] = ((macAddress >> 32) & 0xff);
133   pucMACArray[2] = ((macAddress >> 24) & 0xff);
134   pucMACArray[3] = ((macAddress >> 16) & 0xff);
135   pucMACArray[4] = ((macAddress >>  8) & 0xff);
136   pucMACArray[5] = ((macAddress >>  0) & 0xff);
137
138   // Initialisierung von lwIP, mit DHCP.
139   lwIPInit(pucMACArray, 0, 0, 0, IPADDR_USE_DHCP);
140
141   // Initialisierung von lwIP, mit statischer IP.
142   //unsigned long lowerIP = GPIO_PORTE_DATA_R;
143   //struct ip_addr local_addr;
144
145   //IP4_ADDR(&local_addr,config.ip_d,config.ip_c,config.ip_b,config.ip_a);
146   //lwIPInit(pucMACArray, local_addr.addr, 0xFFFFFF00, 0, IPADDR_USE_STATIC);
147
148   // Setup the device locator service
149   LocatorInit();
150   LocatorMACAddrSet(pucMACArray);
151   LocatorAppTitleSet("LM3S9D92 enet_lwip");
152
153   // ca. 30 ms delay for stabilize Ethernetconnection
154   ul_delay_count = 300000;
155   while (ul_delay_count) ul_delay_count--;
156   //------------------------------
157
158   tcp_nagle_disable(control_connection);
159
160   return 1;
161  }
162
163 /*
164  * Timer Interrupt für Ethernet spezifische Aktionen.
165  * Hier wird u.a. erkannt, wenn eine neue IP-Adresse zugewiesen
166  * wurde. Die Neue IP wird dann über UART ausgegeben.
167  */
168 void lwIPHostTimerHandler(void)
169 {
170  unsigned long ulIPAddress;
```

```
171
172  // Get the local IP address.
173  ulIPAddress = lwIPLocalIPAddrGet();
174
175  // See if an IP address has been assigned.
176  if(ulIPAddress == 0)
177  {
178   // Draw a spinning line to indicate that the IP address is being
179   // discoverd.
180
181         UARTprintf("\b%c", g_pcTwirl[g_ulTwirlPos]);
182
183   // Update the index into the twirl.
184   g_ulTwirlPos = (g_ulTwirlPos + 1) & 3;
185  }
186
187  // Check if IP address has changed, and display if it has
188  else if(ulIPAddress != g_ulLastIPAddr)
189  {
190   // Ausgabe der neuen MAC Adresse
191   lwIPLocalMACGet(pucMACArray);
192   UARTprintf("\n  MAC: %x:%x:%x:%x:%x:%x\n", pucMACArray[0],
193   pucMACArray[1], pucMACArray[2], pucMACArray[3],
194   pucMACArray[4], pucMACArray[5]);
195
196   //
197   // Ausgabe der neuen IP Adresse
198   ulIPAddress = lwIPLocalIPAddrGet();
199         UARTprintf("  IP: %d.%d.%d.%d\n", ulIPAddress & 0xff,
200   (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
201   (ulIPAddress >> 24) & 0xff);
202
203   // Speichern der neuen IP-Adresse
204   g_ulLastIPAddr = ulIPAddress;
205
206   // Ausgabe der neuen Netzmaske
207   ulIPAddress = lwIPLocalNetMaskGet();
208   UARTprintf("  Netmask: %d.%d.%d.%d\n", ulIPAddress & 0xff,
209   (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
210   (ulIPAddress >> 24) & 0xff);
211
212   // Ausgabe der neuen Gateway Adresse
213   ulIPAddress = lwIPLocalGWAddrGet();
214   UARTprintf("  Gateway: %d.%d.%d.%d\n", ulIPAddress & 0xff,
215   (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
216   (ulIPAddress >> 24) & 0xff);
217
218   discover_init();  // Initialisierung des UDP Broadcast Dienstes
219   control_init();   // Initialisierung der TCP-Steuerung
220
221   UARTprintf("\nInitialization Ethernet...done\n\n");
222  }
223
224 }
```

Quellcode E.7: Anhang/c/led.c

```
1  /*
2  Project:       battery_cycling_sw_v4
3  File:          led.c
4
5  Auhtor:        Thomas Wisnewski
6  Credits:
7
8  last modified:      2013/05/27
9
10 Project Status      Under Construction
11 Status:        running
12
13 CCS:        5.5.1.00031
14 Stellarisware:      8555
15
16 Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
17         16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
18         NTC-Connectors, MAX3232 and Suplly Circuits
19
20 Description:      Initialize GPIOs for Frontpanel LEDs
21
22 */
23
24 #include <string.h>
25 #include <stdarg.h>
26 #include <stdio.h>
```

```
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"

#include "driverlib/uart.h"
#include <driverlib/gpio.h>


// Initialize GPIO-Ports for LED in Frontpanel
void led_init(void){
 UARTprintf("Initializing Front LEDs...");

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);    // enable peripheral
    GPIOPinTypeGPIOOutput(GPIO_PORTJ_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7 ); // set PJ4 to PJ7 as digital output
    GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_STRENGTH_8MA,
      GPIO_PIN_TYPE_STD_WPD);

    UARTprintf("done\n");
}
```

Quellcode E.8: Anhang/c/main.c

```
/*
Project:        battery_cycling_sw_v4
File:          main.c

Auhtor:         Thomas Wisnewski
Credits:        Matthias Schneider
          Tobias Steinmann
          Fabian Schwartau
          Johannes Roehn
          Oliver Schmidt
          Stellaris Ware

last modified:      2014/03/19

Project Status      Under Construction
Status:         running

CCS:        5.5.1.00031
Stellarisware:      8555

Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:      Main File for BATSEN Battery Cycling Machine
*/



//*****************************************************************************
*
* Includings
*
*****************************************************************************/
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "string.h"
#include <stdio.h>
#include <stdlib.h>



//*****************************************************************************
*
* Own Includings
*
*****************************************************************************/
#include "utils/uartstdio.h"
```

```
57 #include "driverlib/rom_map.h"
58 #include "header/myuart.h"
59 #include "header/mysdcard.h"
60 #include "header/myssi.h"
61 #include "header/myadc.h"
62 #include "header/relais.h"
63 #include "header/config.h"
64 #include "header/mycmdline.h"
65 #include "header/clocktimer.h"
66 #include "header/temperature.h"
67 #include "header/ethernet.h"
68 #include "header/control.h"
69 #include "header/mypwm.h"
70 #include "header/display.h"
71 #include "header/rtc.h"
72 #include "header/led.h"
73 #include "header/watchdog.h"
74 #include "header/onewire.h"
75 #include "driverlib/timer.h"
76 #include "utils/lwiplib.h"
77 #include "third_party/fatfs/src/ff.h"
78 #include "third_party/fatfs/src/diskio.h"
79
80 //OPto test
81 #include "driverlib/ssi.h"
82 #include "inc/hw_ssi.h"
83
84 /*****************************************************************************
85 *
86 * Globals
87 *
88 *****************************************************************************/
89
90 //Variables for the cat command
91 int g_cat_argc;
92 char g_cat_argv[20], g_cat_flag = 0;
93
94 //Flag for Display ReINIT
95 volatile int g_bDispInit = 0;
96
97 //Flag for temperature measurement
98 volatile int g_bMeasureTemp = 0;
99
100 // String buffer for print operations to UART and Ethernet
101 extern char print_buffer[];
102 // TCP-Socket für die aktuelle Steuerungs-Verbindung
103 extern struct tcp_pcb* control_connection;
104
105 extern char g_cAnsBuf[];
106
107 extern volatile tBoolean g_bFeedWatchdog;
108 // measurement und cycling activ
109 extern volatile unsigned int cycle_active;
110 extern volatile unsigned int measurement_active;
111 // Calibration Flag
112 extern volatile char g_Flag_calibrationModeOn;
113
114 // Instanz der Konfiguration
115 config_t config;
116
117 // Define global location buffer
118 char *g_cLocalBuf;
119
120 // Define Command line pointer
121 tCmdLineEntry *g_psCmdTable;
122
123 // Define string for main location
124 const char *g_cMainLocalBuf = "Main";
125
126 //*****************************************************************************
127 //Extern variables for Clock
128 //*****************************************************************************
129 extern volatile unsigned long clock_msec;
130 extern volatile unsigned long clock_sec;
131 extern volatile unsigned long clock_min;
132 extern volatile unsigned long clock_hour;
133 extern volatile unsigned long clock_day;
134 extern volatile unsigned long clock_month;
135 extern volatile unsigned long clock_year;
136
137 // define command set for main program
138 tCmdLineEntry g_sMainCmdTable[] =
139 {
140     { "help",   Cmd_help,           "   : Display list of commands" },
141     { "h",      Cmd_help,         "     : alias for help" },
```

```
142     { "?",       Cmd_help,          "      : alias for help" },
143     { "browser",my_start_cmd_line,      ": SD-Card browser" },
144     { "config", config_cmd_line,       " : Configuration" },
145     { "alive",   Cmd_alive,            "  : Check if Cycling is active" },
146     { "start",   Cmd_start,            "  : Start measurement cycling" },
147     { "stop",    Cmd_stop,             "   : Stop measurement cycling" },
148     { 0, 0, 0 }
149 };
150
151
152 /* / TEST
153 static void adc_delay_millis(unsigned long milliseconds)
154 {
155  unsigned long counter = 50000 * milliseconds;
156
157  while(counter--)
158   ;
159 }
160 // END TEST
161 */
162
163 void main(void) {
164  // Set the system clock to run at 50MHz from the PLL.
165     //
166     ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
167                        SYSCTL_XTAL_16MHZ);
168
169
170 //     //
171 //     // Enable processor interrupts.
172 //     //
173 //     IntMasterEnable();
174
175     // set Command line pointer to the beginning of the main command structure
176  g_psCmdTable = &g_sMainCmdTable[0];
177
178  // set location buffer to main
179  g_cLocalBuf = (char*)g_cMainLocalBuf;
180
181
182
183     //initializing program
184     myUARTinit();
185     buttons_init();
186  ssi1_init();
187  mysdcardinit();
188     init_config();
189     led_init();
190     init_relais();
191     init_clock_timer();
192     rtc_init();
193     init_temperature();
194     pwm_init();
195     display_init();
196     myadc_init();
197     watchdog_init();
198
199     //Initialisierung Display
200     handlerState=START;
201
202     //DEBUG INIT TRIGGER OUTPUT
203     //GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_5);
204     //GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, 0);
205
206     //*************************************************************
207     //Set Priority of interrupts
208     //*************************************************************
209
210     //Systick 100Hz Timer for FATFs, Clock and Ethernet + Watchdog
211     ROM_IntPrioritySet(FAULT_SYSTICK,  0x00); //1st priority
212     ROM_IntPrioritySet(INT_WATCHDOG, 0x00); //1st priority
213
214     //UART
215 //     ROM_IntPrioritySet(INT_UART0,  0x20);  //2nd priority
216     ROM_IntPrioritySet(INT_UART0,  0x60);  //2nd priority
217     //Ethernet interrupt
218 //     ROM_IntPrioritySet(INT_ETH,   0x20);  //2nd priority
219
220     //Cycle Control Timer
221     ROM_IntPrioritySet(INT_TIMER2A,  0x20);  //2nd priority (SAFETY Reasons)
222     //Cycle LED periodic toggle Timer
223 //     ROM_IntPrioritySet(INT_TIMER3A,  0x20);  //2nd priority
224     ROM_IntPrioritySet(INT_TIMER3A,  0x80);  //2nd priority
225
226     //Cycle Measurement Timer
```

```
227 //    ROM_IntPrioritySet(INT_TIMER0A,  0x40);  //3rd priority
228     ROM_IntPrioritySet(INT_TIMER0A,  0x20);  //3rd priority
229     //ADC Temperature Interrupts
230     ROM_IntPrioritySet(INT_ADC0SS0,  0x40); //3rd priority
231     ROM_IntPrioritySet(INT_ADC1SS1,  0x40); //3rd priority
232
233
234     //Display timer
235     ROM_IntPrioritySet(INT_TIMER1A,  0x60);  //4th priority
236     //Display buttons interrupts
237     ROM_IntPrioritySet(INT_GPIOC,  0x80);  //5th priority
238     ROM_IntPrioritySet(INT_GPIOD,  0x80);  //5th priority
239     ROM_IntPrioritySet(INT_GPIOF,  0x80);  //5th priority
240
241     ROM_IntPrioritySet(INT_SYSCTL,  0xE0); //lowest priority
242
243     //*************************************************************
244
245
246     //
247  // Enable processor interrupts.
248  //
249  IntMasterEnable();
250
251
252     //
253     // Print a prompt to the console.
254     //
255     UARTprintf("\n\n");
256     UARTprintf("\n  Time:   %02d:%02d:%02d", clock_hour, clock_min, clock_sec);
257     UARTprintf("\n  Date:   %02d.%02d.%04d\n\n", clock_day, clock_month, clock_year);
258     UARTprintf("************************************************\n");
259     UARTprintf("***         Battery Cycle Machine          ****\n");
260     UARTprintf("************************************************\n");
261
262     UARTprintf("\n Enter Command (type <help> to see list of commands):");
263     UARTprintf("\n%s: > ", g_cLocalBuf);
264
265     //calibrate_adc();
266     //calibrate_adc_quadratisch();
267     //
268     // Loop forever
269     //
270     UARTprintf("\nchar:%d\n",sizeof(char));
271     UARTprintf("short:%d\n",sizeof(short));
272     UARTprintf("int:%d\n",sizeof(int));
273     UARTprintf("long:%d\n",sizeof(long));
274     UARTprintf("long long:%d\n",sizeof(long long));
275     UARTprintf("cycle_step_t:%d\n",sizeof(cycle_step_t));
276  //g_Flag_calibrationModeOn = 1; //Force calibration
277
278     while(1){
279
280      //WATCHDOG FEED
281      g_bFeedWatchdog = true;//if not fed in the main loop watchdog will reset the system
282
283
284      //Check flag for calibration
285      if(g_Flag_calibrationModeOn){
286       if(!cycle_active && !measurement_active){
287     calibrate_adc_quadratisch();
288
289       }
290       else{
291        UARTprintf("\n ERROR: measurement active! \n");
292       }
293    g_Flag_calibrationModeOn=0;
294      }
295
296      //Check flag for the execution of the cat command
297      if(g_cat_flag){
298       FRESULT fresult;
299          unsigned short usBytesRead;
300
301          //
302          // Disable Timer 1 for Display INT
303          //
304          TimerDisable(TIMER1_BASE, TIMER_A);
305
306          //write message to display being disabled
307          write_to_display("     Display not    ", 0, 0);
308    write_to_display("  available during  ", 1, 0);
309    write_to_display("network transmission", 2, 0);
310    write_to_display("********************", 3 ,0);
311
```

```
312
313             //
314             // First, check to make sure that the current path (CWD), plus
315             // the file name, plus a separator and trailing null, will all
316             // fit in the temporary buffer that will be used to hold the
317             // file name.  The file name must be fully specified, with path,
318             // to FatFs.
319             //
320             if(strlen(g_cCwdBuf) + strlen(g_cat_argv) + 1 + 1 > sizeof(g_cTmpBuf))
321             {
322              sprintf(print_buffer, "Resulting path name is too long\n");
323
324                 UARTprintf(print_buffer);
325
326                 if(control_connection){
327               telnet_write(print_buffer);
328                 }
329                 //return(0);
330             } else {
331
332         //
333         // Copy the current path to the temporary buffer so it can be manipulated.
334         //
335         strcpy(g_cTmpBuf, g_cCwdBuf);
336
337         //
338         // If not already at the root level, then append a separator.
339         //
340         if(strcmp("/", g_cCwdBuf))
341         {
342          strcat(g_cTmpBuf, "/");
343         }
344
345         //
346         // Now finally, append the file name to result in a fully specified file.
347         //
348         strcat(g_cTmpBuf, g_cat_argv);
349
350         //
351         // Open the file for reading.
352         //
353         fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
354
355         //
356         // If there was some problem opening the file, then return
357         // an error.
358         //
359         if(fresult != FR_OK)
360         {
361          //return(fresult);
362         } else {
363
364          //If Ethernet connected no output via UART
365          if(control_connection)
366           UARTprintf("\nOutput only via connected Ethernet!\n");
367          //
368          // Enter a loop to repeatedly read data from the file and display it,
369          // until the end of the file is reached.
370          //
371          do{
372
373           //WATCHDOG FEED
374           g_bFeedWatchdog = true;
375
376           //
377           // Read a block of data from the file.  Read as much as can fit
378           // in the temporary buffer, including a space for the trailing null.
379           //
380           fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf) - 1,
381                &usBytesRead);
382
383           //
384           // If there was an error reading, then print a newline and
385           // return the error to the user.
386           //
387           if(fresult != FR_OK)
388           {
389            sprintf(print_buffer, "\n");
390            UARTprintf(print_buffer);
391            if(control_connection){
392             telnet_write(print_buffer);
393            }
394            //return(fresult);
395            break;
396           }
```

```
397
398         //
399         // Null terminate the last block that was read to make it a
400         // null terminated string that can be used with printf.
401         //
402         g_cTmpBuf[usBytesRead] = 0;
403
404         //
405         // Print the last chunk of the file that was received.
406         //
407         sprintf(print_buffer, "%s", g_cTmpBuf);
408
409         //Send data over Etherenet OR UART due to performance optimization
410         //Output over both interfaces slows down the Ethernet connection
411         if(control_connection){
412          telnet_immediate_write(print_buffer);
413         } else
414          UARTprintf(print_buffer);
415
416         //
417         // Continue reading until less than the full number of bytes are
418         // read.  That means the end of the buffer was reached.
419         //
420
421       } while(usBytesRead == sizeof(g_cTmpBuf) - 1);
422
423       if(control_connection){
424        telnet_immediate_write("\nEnd of file\n Enter Command (type <help> to see list of commands):\n");
425        sprintf(g_cAnsBuf, "%s: > \n", g_cLocalBuf);
426        telnet_immediate_write(g_cAnsBuf);
427       }
428
429       //
430       // Print a prompt to the console.
431       //
432       UARTprintf("\nEnd of file");
433       UARTprintf("\n Enter Command (type <help> to see list of commands):");
434       UARTprintf("\r%s: > ", g_cLocalBuf);
435
436        }
437         }
438           //Delete Flag
439           g_cat_flag = 0;
440
441           //
442           // Enable Timer 1 for Display INT
443           //
444         TimerEnable(TIMER1_BASE, TIMER_A);
445       }
446
447       //Re INIT Display every minute
448       if(g_bDispInit){
449        g_bDispInit = 0;
450        TimerDisable(TIMER1_BASE, TIMER_A);
451        display_reinit();
452        TimerEnable(TIMER1_BASE, TIMER_A);
453       }
454
455
456 //      if(g_bMeasureTemp) {
457 //       get_temperature(0); //Parameter only for compat.
458 //       g_bMeasureTemp = 0;
459 //      }
460     }
461 }
```

Quellcode E.9: Anhang/c/myadc.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          myadc.c
4
5  Auhtor:         Thomas Wisnewski
6  Credits:        Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Johannes Roehn
10          Oliver Schmidt
11          Stellaris Ware
12
13 last modified:       2014/03/20
14
15 Project Status       Under Construction
```

```
Status:         running

CCS:         5.5.1.00031
Stellarisware:       8555

Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:       Source Code for AD7798 usage

*/



/*****************************************************************************
*
* Includings
*
*****************************************************************************/


#include "inc/hw_ints.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib/rom_map.h"

#include "driverlib/interrupt.h"

#include "inc/hw_memmap.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/ssi.h"
#include "inc/hw_ssi.h"
#include <stdio.h>
#include <stdlib.h>

/*****************************************************************************
*
* Own Includings
*
*****************************************************************************/
#include "header/myadc.h"
#include "header/config.h"
#include "header/relais.h"
#include "header/myssi.h"

//*****************************************************************************
//
// A global data buffer used for actual Cell voltage
//
//*****************************************************************************
volatile unsigned int g_iCellVoltages[12] = {0};
volatile int g_iCurrent = 0;


#define DIFF_V 5000
#define DIFF_C 100

/* variables to store internal calibration result */
static unsigned int offset_regs[3];
static unsigned int fullscale_regs[3];

// variables for a aktive relay
extern volatile unsigned int power_relay_active;

// AD7798 initialization and configuration
int myadc_init(void)
{
 UARTprintf("Initialization of external ADC...");
 //adc_clear_fifo();  // SSI1 Fifo löschen

 // ADC reset
 spiChipSelect(ADC);

 SSIDataPut(SSI1_BASE, 0xFF);
 SSIDataPut(SSI1_BASE, 0xFF);
 SSIDataPut(SSI1_BASE, 0xFF);
 SSIDataPut(SSI1_BASE, 0xFF);

 spiChipSelect(NONE);
```

```
101
102
103     // SPI interface of the ADC needs 500us to come up again
104 // adc_delay_micros(500L);
105  adc_delay_millis(1L);
106     //Initialisierung
107  adc_write_reg24(configuration, POLARITY_MODE | CONFIG_BUF);
108
109 // if(!adc_internal_calibration(channel1)) return 0;
110 // if(!adc_internal_calibration(channel2)) return 0;
111 // if(!adc_internal_calibration(channel3)) return 0;
112
113
114
115  adc_calibrate_hall_sensor();
116
117
118
119  UARTprintf("done.\n");
120  return 1;
121 }
122
123
124
125 void myadc_reinit(void)
126 {
127  UARTprintf("Re-Initialization of external ADC...");
128  //adc_clear_fifo();  // SSI1 Fifo löschen
129
130  // ADC reset
131  spiChipSelect(ADC);
132
133  SSIDataPut(SSI1_BASE, 0xFF);
134  SSIDataPut(SSI1_BASE, 0xFF);
135  SSIDataPut(SSI1_BASE, 0xFF);
136  SSIDataPut(SSI1_BASE, 0xFF);
137
138  spiChipSelect(NONE);
139
140
141     // SPI interface of the ADC needs 500us to come up again
142 // adc_delay_micros(500L);
143  adc_delay_millis(1L);
144     //Initialisierung
145  adc_write_reg24(configuration, POLARITY_MODE | CONFIG_BUF);
146
147 // if(!adc_internal_calibration(channel1)) return 0;
148 // if(!adc_internal_calibration(channel2)) return 0;
149 // if(!adc_internal_calibration(channel3)) return 0;
150
151
152  UARTprintf("done.\n");
153 // adc_clear_fifo(); // SSI1 Fifo löschen
154 //
155 // // ADC reset
156 // spiChipSelect(ADC);
157 // SSIDataPut(SSI1_BASE, 0xFF);
158 // SSIDataPut(SSI1_BASE, 0xFF);
159 // SSIDataPut(SSI1_BASE, 0xFF);
160 // SSIDataPut(SSI1_BASE, 0xFF);
161 // spiChipSelect(NONE);
162 //
163 // // SPI interface of the ADC needs 500us to come up again
164 // adc_delay_micros(500L);
165 //// adc_delay_millis(1L);
166 // //restore register values
167 // adc_write_reg24(mode, MODE_SEL_PWRDOWN | FILTER_MODE);
168 // adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH3);
169 // adc_write_reg24(offset, offset_regs[2]);
170 // adc_write_reg24(fullscale, fullscale_regs[2]);
171 // adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH2);
172 // adc_write_reg24(offset, offset_regs[1]);
173 // adc_write_reg24(fullscale, fullscale_regs[1]);
174 // adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH1);
175 // adc_write_reg24(offset, offset_regs[0]);
176 // adc_write_reg24(fullscale, fullscale_regs[0]);
177 }
178
179
180
181 //OLLI
182
183 int adc_internal_calibration(adc_channel_t channel)
184 {
185  unsigned int *offset_reg;
```

```
186  unsigned int *fullscale_reg;
187  unsigned char conv_stat;
188  unsigned int config_data = adc_read_reg24(configuration);
189  unsigned int mode_data = adc_read_reg24(mode);
190
191  config_data &= ~CONFIG_CH_CLR; // Kanalauswahl löschen
192
193  switch(channel){ // Kanal selektieren
194  case channel1:
195   config_data |= CONFIG_CH1;
196   offset_reg    = &(offset_regs[0]);
197   fullscale_reg = &(fullscale_regs[0]);
198   break;
199  case channel2:
200   config_data |= CONFIG_CH2;
201   offset_reg    = &(offset_regs[1]);
202   fullscale_reg = &(fullscale_regs[1]);
203   break;
204  case channel3:
205   config_data |= CONFIG_CH3;
206   offset_reg    = &(offset_regs[2]);
207   fullscale_reg = &(fullscale_regs[2]);
208   break;
209  default: return 0;
210  }
211
212  adc_write_reg24(configuration, config_data);
213
214
215
216  adc_write_reg24(mode, MODE_SEL_PWRDOWN | FILTER_MODE);  // In den "power down" Modus gehen, um das "mode register" zu ändern
217 // adc_write_reg24(offset, 0x000001);        // Initialisierung des "offset registers" mit 0
218 // adc_write_reg24(fullscale, 0x0FFFFF);     // Initialisierung des "full scale registers" mit 0
219 //
220 //
221 // // Prüfen, ob die Werte korrekt geschrieben wurden
222 // if(adc_read_reg24(offset) != 0){
223 //  UARTprintf("failed: offset calibration reset for Ch. %d failed!\n",(int)channel);
224 //  return 0;
225 // }
226 // if(adc_read_reg24(fullscale) != 0){
227 //  UARTprintf("failed: full scale calibration reset for Ch. %d failed!\n",(int)channel);
228 //  return 0;
229 // }
230
231  // Durchführung der internen Kalibrierung des ADCs
232  adc_write_reg24(mode, MODE_SEL_INTZSCAL | FILTER_MODE);  // Internal zero offset calibration
233  do{
234   conv_stat = adc_read_reg8(status);
235  }
236  while(conv_stat & STATUS_nRDY);
237  f_adc_error |= conv_stat & (STATUS_ERR | STATUS_NOREF);
238
239  adc_write_reg24(mode, MODE_SEL_INTFSCAL | FILTER_MODE);  // Internal full scale calibration
240  do{
241   conv_stat = adc_read_reg8(status);
242  }
243  while(conv_stat & STATUS_nRDY);
244  f_adc_error |= conv_stat & (STATUS_ERR | STATUS_NOREF);
245
246
247 // if(f_adc_error){
248 //  UARTprintf("\nADC ERROR!");
249 //  while(1);
250 // }
251
252
253  // Prüfen, ob die Kalibierung erfolgreich war (sich die Werte geändert haben)
254  *offset_reg = adc_read_reg24(offset);
255  *fullscale_reg = adc_read_reg24(fullscale);
256
257  // Die Werte ausgeben
258  UARTprintf("\nOFFS VAL: %x", *offset_reg);
259  UARTprintf("\nFS VAL: %x", *fullscale_reg);
260
261
262  if(*offset_reg == 0){
263   UARTprintf("failed: offset calibration for Ch. %d failed!\n",(int)channel);
264   return 0;
265  }
266
267  if(*fullscale_reg == 0){
268   UARTprintf("failed: full scale calibration for Ch. %d failed!\n",(int)channel);
269   return 0;
270  }
```

```
271   return 1;
272 }
273
274
275
276
277
278 // Get the voltage from the ADC
279 unsigned int adc_get_voltage(unsigned int choose)
280 {
281  unsigned int adc_digit;
282  unsigned int voltage;
283
284  float A,A1,B,C,Erg;
285  char buffer[100];  //Für UART ausgaben
286
287
288  static int adc_digit_save=0;
289  static unsigned int voltage_save=0;
290
291
292  switch_relais(choose);
293  // 1ms warten, damit Relais einschwingen können
294  adc_delay_millis(10L);
295
296
297  adc_digit  = (adc_get_single_value(channel1)) ;
298 // Berechnung der Spannung U = A * adc_digit^2 + B * adc_digit + C
299
300 // A1 = adc_digit*config.A[choose]; // Problem: wenn in einer Zeile passiert bei ca UTek=3,653V ^= Dig=46423 ein überlauf?!? Deswegen
        in 2 Zeilen!
301 // A  = A1  * adc_digit;
302
303  A = adc_digit*adc_digit*config.A[choose];
304
305  B  = adc_digit*config.B[choose];
306
307  C  = config.C[choose];
308
309  Erg = A+B+C;
310  voltage = (unsigned int)(Erg+0.5); //runden
311
312 //  Ausgange der adc_digit und der Spannung in der UART
313 // sprintf(buffer,"ADC: DIGI:%d  U:%d\n",output,voltage);
314 // UARTprintf(buffer);
315
316  if(voltage<=25000)      //Alles unter 25mV ist Null
317   voltage=0;
318
319  if(config.quantity_cells!=1){  //Wenn nur ein Channel ausgewählt ist, schaltet das Relay nicht zurück
320   switch_relais(0);
321  }
322
323 // Test um Sprünge in der Spannung zu entdecken
324  if(abs(voltage_save - voltage) > DIFF_V){
325
326   UARTprintf("Abweichung der Spannung um %duV\n",DIFF_V);
327   sprintf(buffer,"ADC: alt:%d  neu:%d\n",adc_digit_save, adc_digit);
328   UARTprintf(buffer);
329   sprintf(buffer,"U:   alt:%d  neu:%d\n",voltage_save, voltage);
330   UARTprintf(buffer);
331   sprintf(buffer,"Koeff: %f, %10.6f, %10.6f\n",config.A[choose]*1000000, config.B[choose], config.B[choose]);
332   UARTprintf(buffer);
333  }
334
335  adc_digit_save = adc_digit;
336  voltage_save = voltage;
337 // Test_Ende
338  g_iCellVoltages[choose - 1] = voltage;
339  return voltage;
340 }
341
342
343 // Get the current from the ADC
344 int adc_get_current(void){
345  char buffer[100];
346
347  unsigned int adc_digit;
348  int adc_offset;
349  int current ;
350
351  static int current_save=0;
352  static unsigned int adc_digit_save=0;
353
354  adc_digit=adc_get_single_value(channel2);  // 32768 entspricht 0V
```

```
355
356 //Hall effekt calibrierung wenn kein Relay angezogen ist
357  if(power_relay_active == 0){
358   adc_offset = (int)(32768UL - adc_digit);
359
360   if(abs(adc_offset-config.zerocurrentch1) > 10){
361    config.zerocurrentch1 = adc_offset;
362    UARTprintf("\n Neuer Current Offset %d\n",adc_offset);
363   }
364  }
365  else{
366   adc_offset=config.zerocurrentch1;
367  }
368
369
370
371 // UARTprintf("\nCALI!:%d\n",(int)(32768UL - adc_get_single_value(channel2)));
372
373
374  float tmp1 = (adc_digit+adc_offset) * KOEFF_CH1 * ADC_LSB;
375  float tmp2 = KOEFF_BG_CH1;
376
377  current = (int)((tmp1 - tmp2)*1000000 + 0.5);
378
379 // current = (unsigned long long)((a) + b) * 76293945 / 1000000000 * 3745  / 100 - 93625;
380
381
382 // current = (unsigned long long)(adc_get_single_value(channel2) + config.zerocurrentch1) * 76293945 / 1000000000 * 3745  / 100 -
        93625;
383  if(abs(current) <= 50){
384   current = 0;
385  }
386
387  if(current <= -75000 || current >= 75000){
388 //  current = (unsigned long long)((adc_get_single_value(channel3)) + config.zerocurrentch2) * 76293945 / 1000000000 * 250 - 625000;
389   adc_digit=adc_get_single_value(channel3);
390   adc_offset=config.zerocurrentch2;
391   tmp1 = (adc_digit+adc_offset) * KOEFF_CH2 * ADC_LSB;
392   tmp2 = KOEFF_BG_CH2;
393
394
395   current = (int)((tmp1 - tmp2)*1000000 + 0.5);
396
397   if(abs(current) <= 10){
398    current = 0;
399   }
400  }
401
402
403 // Test um Sprünge in dem Strom zu entdecken
404  if(abs(current_save - current) > DIFF_C){
405
406   UARTprintf("Abweichung des Strom um %dmA\n",DIFF_C);
407   sprintf(buffer,"ADC: alt:%d  neu:%d\n",adc_digit_save, adc_digit);
408   UARTprintf(buffer);
409   sprintf(buffer,"C:   alt:%d  neu:%d\n",current_save, current);
410   UARTprintf(buffer);
411   sprintf(buffer,"Koeff: %d\n",config.zerocurrentch1);
412   UARTprintf(buffer);
413  }
414
415  current_save = current;
416  adc_digit_save = adc_digit;
417 // Test_Ende
418
419
420  g_iCurrent = (-1) * current;//positive current for charging, negative for discharge
421 // g_iCurrent =0;
422  return (int)g_iCurrent;
423 }
424
425
426
427
428
429 unsigned int adc_get_single_value(adc_channel_t channel)
430 {
431  unsigned char conv_stat;
432  unsigned int temp;//, temp2;
433
434  switch(channel){
435  case channel1:
436   adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH1);
437   break;
438  case channel2:
```

```
439    adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH2);
440    break;
441   case channel3:
442    adc_write_reg24(configuration, POLARITY_MODE | CONFIG_CH3);
443    break;
444   default: return 0;
445   }
446
447   //set ADC into single conversion mode
448
449   adc_write_reg24(mode, MODE_SEL_SINCONV | FILTER_MODE);
450
451   // wait for conversion to be done
452   do{
453    conv_stat = adc_read_reg8(status);
454    if(!conv_stat){
455     f_adc_error = EX_STATUS_ADC_COM_ERR;
456     UARTprintf("ADC_ERROR Ch.:%d: SATUS_REG = 0x00\n",channel);
457     break;
458    }
459   }
460   while(conv_stat & STATUS_nRDY);
461
462  // if(conv_stat & (STATUS_ERR | STATUS_NOREF)){
463  //  f_adc_error = conv_stat & (STATUS_ERR | STATUS_NOREF);
464   if(conv_stat & (STATUS_NOREF)){
465     f_adc_error = conv_stat & (STATUS_NOREF);
466    UARTprintf("ADC_ERROR: ErrorFlag");
467   }
468
469
470
471   temp=adc_read_reg24(data);
472  // while(GPIOPinRead(GPIO_PORTE_BASE, GPIO_PIN_2) != 0);
473  // while(adc_write_read(COM_nWEN) != 0xff)
474  //   ;
475   temp = (temp+128) >> 8; // Hier wird das Gesamtergebnis gerundet und das LSByte abgeschnitten
476
477
478    return temp;
479  }
480
481
482
483
484  void adc_calibrate_hall_sensor(void)
485  {
486   //Calibrate HALL-Sensor to 0
487   int avg=0;
488   char k;
489
490
491   for(k = 0; k < NO_CURRENT_MEASUREMENTS; k++){
492    avg += (int)(32768UL - adc_get_single_value(channel2));
493   }
494
495   config.zerocurrentch1 = (int)(avg / NO_CURRENT_MEASUREMENTS + 0.5);
496
497  // config.zerocurrentch1 = (int)(32768UL - adc_get_single_value(channel2));
498
499  // UARTprintf("\nSollCaliWert=173\n Ist CALI:%d\n",config.zerocurrentch1);
500
501   avg = 0;
502   for(k = 0; k < NO_CURRENT_MEASUREMENTS; k++){
503    avg += (int)(32768UL - adc_get_single_value(channel3));
504   }
505
506   config.zerocurrentch2 = (int)(avg / NO_CURRENT_MEASUREMENTS + 0.5);
507
508
509  // config.zerocurrentch2 = (int)(32768UL - adc_get_single_value(channel3));
510  }
511
512  /* Read a 8-bit wide register (communication, status, ID, IO) */
513  static unsigned char adc_read_reg8(adc_reg8_t reg)
514  {
515   unsigned char temp;
516
517   switch(reg) {
518   case communication: temp = COM_COM;    break;
519   case status:        temp = COM_STATUS; break;
520   case id:            temp = COM_ID;     break;
521   case io:            temp = COM_IO;     break;
522   default:            return 0;
523   }
```

```
524
525   spiChipSelect(ADC);
526   adc_write_read(COM_RnW | temp);
527   temp = adc_write_read(0x00);
528   spiChipSelect(NONE);
529   return temp;
530 }
531
532
533
534
535
536 /* Read a 24-bit wide register (mode, configuration, data, offset, fullscale) */
537 static unsigned int adc_read_reg24(adc_reg24_t reg)
538 {
539   unsigned int temp;
540
541   switch(reg) {
542   case mode:          temp = COM_MODE;      break;
543   case configuration: temp = COM_CONFIG;    break;
544   case data:          temp = COM_DATA;      break;
545   case offset:        temp = COM_OFFSET;    break;
546   case fullscale:     temp = COM_FULLSCALE; break;
547   default:            return 0;
548   }
549
550   spiChipSelect(ADC);
551   adc_write_read((unsigned char)(COM_RnW | temp));
552 #if (ADC_TYPE == AD7799)
553   if((reg != mode) && (reg != configuration)){
554     temp = ((unsigned int)(adc_write_read(0x00)))<<16;
555 //   temp = ssi_get8()<<16;
556   }
557 #endif
558   temp |= ((unsigned int)(adc_write_read(0x00)))<<8;
559   temp |= (unsigned int)adc_write_read(0x00);
560 // temp |= ssi_get8()<<8;
561 // temp |= ssi_get8();
562
563   spiChipSelect(NONE);
564   return temp;
565 }
566
567
568 /* Write a 8-bit wide register (communication, ID, IO) */
569 static void adc_write_reg8(adc_reg8_t reg, unsigned char value)
570 {
571   unsigned char temp;
572
573   switch(reg) {
574   case communication: temp = COM_COM;     break;
575   case status:        return;                   /* status register is read-only */
576   case id:            temp = COM_ID;      break;
577   case io:            temp = COM_IO;      break;
578   default:            return;
579   }
580   spiChipSelect(ADC);
581   adc_write_read(temp);
582   adc_write_read(value);
583   spiChipSelect(NONE);
584 }
585
586
587
588 /* Write a 24-bit wide register (mode, configuration, data, offset, fullscale) */
589 static void adc_write_reg24(adc_reg24_t reg, unsigned int value)
590 {
591   unsigned char temp;
592
593   switch(reg) {
594   case mode:          temp = COM_MODE;      break;
595   case configuration: temp = COM_CONFIG;    break;
596   case data:          temp = COM_DATA;      break;
597   case offset:        temp = COM_OFFSET;    break;
598   case fullscale:     temp = COM_FULLSCALE; break;
599   default:            return;
600   }
601   spiChipSelect(ADC);
602   adc_write_read(temp);
603 #if (ADC_TYPE == AD7799)
604   if((reg != mode) && (reg != configuration))  // MODE and CONFIGURATION registers are 16 bit long
605     adc_write_read((unsigned char)(value >> 16));
606 #endif
607   adc_write_read((unsigned char)(value >> 8));
608   adc_write_read((unsigned char)(value & 0xff));
```

```
609
610  spiChipSelect(NONE);
611 }
612
613 /*
614  * Schreib-, Leseoperation für den ADC über SSI
615  */
616 static unsigned char adc_write_read(unsigned char data)
617 {
618  unsigned long ans;
619
620  adc_clear_fifo();
621  SSIDataPut(SSI1_BASE, data);
622  SSIDataGet(SSI1_BASE, &ans);
623
624  return (unsigned char)(ans&0xff);
625 }
626
627 /*
628  * Löschen des Hardware FIFOs vom SPI-Interface
629  */
630 static void adc_clear_fifo(void)
631 {
632  unsigned long dummy;
633
634  while(SSIDataGetNonBlocking(SSI1_BASE, &dummy))  // clear the fifo
635   ;
636 }
637
638 static void adc_delay_micros(unsigned long microseconds)
639 {
640  unsigned long counter = 50 * microseconds;
641
642  while(counter--)
643   ;
644 }
645
646 static void adc_delay_millis(unsigned long milliseconds)
647 {
648  unsigned long counter = 5000 * milliseconds;
649
650  while(counter--)
651   ;
652 }
```

Quellcode E.10: Anhang/c/mycmdline.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:           mycmdline.c
4
5  Auhtor:         Thomas Wisniewski
6  Credits:        Matthias Schneider
7           Stellaris Ware
8
9  last modified:      2013/03/14
10
11 Project Status      Under Construction
12 Status:         running
13
14 CCS:         5.5.1.00031
15 Stellarisware:       8555
16
17 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
18          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
19          NTC-Connectors, MAX3232 and Suplly Circuits
20
21 Description:       Source File to implement a command line.
22          Modified version of a stellarisware file.
23
24 */
25
26 //*****************************************************************************
27 //
28 // cmdline.c - Functions to help with processing command lines.
29 //
30 // Copyright (c) 2007-2012 Texas Instruments Incorporated.  All rights reserved.
31 // Software License Agreement
32 //
33 // Texas Instruments (TI) is supplying this software for use solely and
34 // exclusively on TI's microcontroller products. The software is owned by
35 // TI and/or its suppliers, and is protected under applicable copyright
36 // laws. You may not combine this software with "viral" open-source
```

```
37 // software in order to form a larger program.
38 //
39 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
40 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
41 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
42 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
43 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
44 // DAMAGES, FOR ANY REASON WHATSOEVER.
45 //
46 // This is part of revision 8555 of the Stellaris Firmware Development Package.
47 //
48 //*****************************************************************************
49
50 //*****************************************************************************
51 //
52 //! \addtogroup cmdline_api
53 //! @{
54 //
55 //*****************************************************************************
56
57 #include <string.h>
58 #include "header/mycmdline.h"
59 #include "header/control.h"
60 #include "utils/uartstdio.h"
61 #include <stdio.h>
62
63 //*****************************************************************************
64 //
65 // Defines the maximum number of arguments that can be parsed.
66 //
67 //*****************************************************************************
68 #ifndef CMDLINE_MAX_ARGS
69 #define CMDLINE_MAX_ARGS        8
70 #endif
71
72 //*****************************************************************************
73 //
74 // Defines the size of the buffer that holds the command line.
75 //
76 //*****************************************************************************
77 #define CMD_BUF_SIZE    64
78
79 //*****************************************************************************
80 //
81 // The buffer that holds the command line.
82 //
83 //*****************************************************************************
84 char g_cCmdBuf[CMD_BUF_SIZE];
85
86 //*****************************************************************************
87 //
88 // The buffer that holds the answer line.
89 //
90 //*****************************************************************************
91 char g_cAnsBuf[CMD_BUF_SIZE];
92
93
94 // TCP-Socket für die aktuelle Steuerungs-Verbindung
95 extern struct tcp_pcb* control_connection;
96
97 extern volatile unsigned int cycle_active;
98
99 // Global location buffer
100 extern char *g_cLocalBuf;
101
102 //*****************************************************************************
103 //
104 //! Process a command line string into arguments and execute the command.
105 //!
106 //! \param pcCmdLine points to a string that contains a command line that was
107 //! obtained by an application by some means.
108 //!
109 //! This function will take the supplied command line string and break it up
110 //! into individual arguments.  The first argument is treated as a command and
111 //! is searched for in the command table.  If the command is found, then the
112 //! command function is called and all of the command line arguments are passed
113 //! in the normal argc, argv form.
114 //!
115 //! The command table is contained in an array named <tt>g_sCmdTable</tt> which
116 //! must be provided by the application.
117 //!
118 //! \return Returns \b CMDLINE_BAD_CMD if the command is not found,
119 //! \b CMDLINE_TOO_MANY_ARGS if there are more arguments than can be parsed.
120 //! Otherwise it returns the code that was returned by the command function.
121 //
```

```
122 //*****************************************************************************
123 int
124 CmdLineProcess(char *pcCmdLine)
125 {
126     static char *argv[CMDLINE_MAX_ARGS + 1];
127     char *pcChar;
128     int argc;
129     int bFindArg = 1;
130     tCmdLineEntry *pCmdEntry;
131     extern tCmdLineEntry *g_psCmdTable;
132 
133     //
134     // Initialize the argument counter, and point to the beginning of the
135     // command line string.
136     //
137     argc = 0;
138     pcChar = pcCmdLine;
139 
140     //
141     // Advance through the command line until a zero character is found.
142     //
143     while(*pcChar)
144     {
145         //
146         // If there is a space, then replace it with a zero, and set the flag
147         // to search for the next argument.
148         //
149         if(*pcChar == ' ')
150         {
151             *pcChar = 0;
152             bFindArg = 1;
153         }
154 
155         //
156         // Otherwise it is not a space, so it must be a character that is part
157         // of an argument.
158         //
159         else
160         {
161             //
162             // If bFindArg is set, then that means we are looking for the start
163             // of the next argument.
164             //
165             if(bFindArg)
166             {
167 
168                 if(argc < CMDLINE_MAX_ARGS)
169                 {
170                     argv[argc] = pcChar;
171                     argc++;
172                     bFindArg = 0;
173                 }
174 
175                 //
176                 // The maximum number of arguments has been reached so return
177                 // the error.
178                 //
179                 else
180                 {
181                     return(CMDLINE_TOO_MANY_ARGS);
182                 }
183             }
184         }
185 
186         //
187         // Advance to the next character in the command line.
188         //
189         pcChar++;
190     }
191 
192     //
193     // If one or more arguments was found, then process the command.
194     //
195     if(argc)
196     {
197         //
198         // Start at the beginning of the command table, to look for a matching
199         // command.
200         //
201         pCmdEntry = g_psCmdTable;
202 
203         //
204         // Search through the command table until a null command string is
205         // found, which marks the end of the table.
206         //
```

```
207         while(pCmdEntry->pcCmd)
208         {
209             //
210             // If this command entry command string matches argv[0], then call
211             // the function for this command, passing the command line
212             // arguments.
213             //
214             if(!strcmp(argv[0], pCmdEntry->pcCmd))
215             {
216                 return(pCmdEntry->pfnCmd(argc, argv));
217             }
218
219             //
220             // Not found, so advance to the next entry.
221             //
222             pCmdEntry++;
223         }
224     }
225
226     //
227     // Fall through to here means that no matching command was found, so return
228     // an error.
229     //
230     return(CMDLINE_BAD_CMD);
231 }
232
233 //*****************************************************************************
234 //
235 // Close the Doxygen group.
236 //! @}
237 //
238 //*****************************************************************************
239
240
241
242
243 //*****************************************************************************
244 //
245 // Function to interprete command. Check if command is valid and execute.
246 //
247 //*****************************************************************************
248 void Cmd_interprete(char *g_cCmdBuf){
249
250  int nStatus;
251
252  //
253  // Echo received command
254  //
255  UARTprintf("%s", g_cCmdBuf);
256  sprintf(g_cAnsBuf, "%s\n", g_cCmdBuf);
257  telnet_write(g_cAnsBuf);
258
259  //
260  // Pass the line from the user to the command processor.
261  // It will be parsed and valid commands executed.
262  //
263  nStatus = CmdLineProcess(g_cCmdBuf);
264
265  //
266  // Handle the case of bad command.
267  //
268  if(nStatus == CMDLINE_BAD_CMD)
269  {
270   UARTprintf("\n Bad command!");
271   if(control_connection){
272    sprintf(g_cAnsBuf, " Bad command!\n");
273    telnet_write(g_cAnsBuf);
274   }
275  }
276
277  //
278  // Handle the case of too many arguments.
279  //
280  else if(nStatus == CMDLINE_TOO_MANY_ARGS)
281  {
282   UARTprintf("\n Too many arguments for command processor!");
283   if(control_connection){
284    sprintf(g_cAnsBuf,  " Too many arguments for command processor!\n");
285    telnet_write(g_cAnsBuf);
286   }
287  }
288
289  if(control_connection){
290   telnet_write(" Enter Command (type <help> to see list of commands):\n");
291   sprintf(g_cAnsBuf, "%s: > \n", g_cLocalBuf);
```

```
292   telnet_write(g_cAnsBuf);
293  }
294
295  //
296  // Print a prompt to the console.
297  //
298  UARTprintf("\n Enter Command (type <help> to see list of commands):");
299  UARTprintf("\r%s: > ", g_cLocalBuf);
300 }
301
302 //*****************************************************************************
303 //
304 // Command to check whetet Cycling mode is current active. Returns "YES" or "NO"
305 //
306 //*****************************************************************************
307 int
308 Cmd_alive(int argc, char *argv[])
309 {
310  if(cycle_active){
311      UARTprintf("\nYES");
312   if(control_connection)
313    telnet_write("YES\n");
314   }
315     else{
316      UARTprintf("\nNO");
317   if(control_connection)
318    telnet_write("NO\n");
319     }
320     //
321     // Return success.
322     //
323     return(0);
324 }
```

Quellcode E.11: Anhang/c/mypwm.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          mypwm.c
4
5  Auhtor:         Thomas Wisniewski
6  Credits:        Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Stellaris Ware
10
11 last modified:       2013/03/27
12
13 Project Status       Under Construction
14 Status:         running
15
16 CCS:         5.5.1.00031
17 Stellarisware:       8555
18
19 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
20          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
21          NTC-Connectors, MAX3232 and Suplly Circuits
22
23 Description:       Source Code for pwm initalization for use LED display
24
25 */
26
27
28
29 /*****************************************************************************
30 *
31 * Own Includings
32 *
33 *****************************************************************************/
34
35
36 #include "inc/hw_ints.h"
37 #include "inc/hw_memmap.h"
38 #include "inc/hw_types.h"
39 #include "inc/hw_ssi.h"
40 #include "driverlib/debug.h"
41 #include "driverlib/gpio.h"
42 #include "driverlib/interrupt.h"
43 #include "driverlib/pin_map.h"
44 #include "driverlib/rom.h"
45 #include "driverlib/ssi.h"
46 #include "driverlib/sysctl.h"
47 #include "driverlib/uart.h"
```

```
48 #include "utils/uartstdio.h"
49 #include "driverlib/pwm.h"
50
51
52
53 //*****************************************************************************
54 //
55 // Initialize PWM for contrast setting of LED-Display
56 //
57 //*****************************************************************************
58 void pwm_init(void)
59 {
60
61  UARTprintf("Initializing PWM...");
62
63  unsigned long ulPeriod;
64     //
65     // Enable the peripherals used by this example.
66     //
67     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
68     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
69
70     //
71     // Set GPIO H6 as PWM pin.  It is used to output the PWM4 signal.
72     //
73     GPIOPinConfigure(GPIO_PH6_PWM4);
74     ROM_GPIOPinTypePWM(GPIO_PORTH_BASE, GPIO_PIN_6);
75
76     //
77  // Compute the PWM period based on the system clock.
78  //
79  ulPeriod = ROM_SysCtlClockGet() / 440;
80
81  //
82  // Set the PWM period to 440 (A) Hz.
83  //
84  ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_2,
85        PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
86  ROM_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, ulPeriod);
87
88  //
89  // Set PWM4 to a duty cycle of 25%.
90  //
91  ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, ulPeriod / 600);
92  //ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, (ulPeriod * 3) / 4);
93
94  //
95  // Enable the PWM0 and PWM1 output signals.
96  //
97  ROM_PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT , true);
98
99  //
100 // Enable the PWM generator.
101 //
102 ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_2);
103
104  UARTprintf("done\n");
105 }
```

Quellcode E.12: Anhang/c/mysdcard.c

```
1  /*
2  Project:       battery_cycling_sw_v4
3  File:          mysdcard.c
4
5  Auhtor:        Thomas Wisniewski
6  Credits:       Matthias Schneider
7           Johannes Roehn
8           Oliver Schmidt
9           Stellaris Ware
10
11 last modified:      2015/04/13
12
13 Project Status      Under Construction
14 Status:        running
15
16 CCS:         5.5.1.00031
17 Stellarisware:      8555
18
19 Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
20          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
21          NTC-Connectors, MAX3232 and Suplly Circuits
22
```

```
Description:      Source Code for SDCard at SPI used with FatFs

        Mainly copied from ti stellarisware example:
        stellarisware/boards/ek-lm329b96/sdcard
*/

/*****************************************************************************
*
* Original Stellaris Ware Includings
*
*****************************************************************************/

#include <string.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ssi.h"
#include <inc/lm3s9b92.h>
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include <driverlib/gpio.h>
#include "driverlib/rom.h"
#include "driverlib/timer.h"
#include "header/mycmdline.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "third_party/fatfs/src/ff.h"
#include "third_party/fatfs/src/diskio.h"

/*****************************************************************************
*
* Own Includings
*
*****************************************************************************/

#include "header/mysdcard.h"
#include "header/myadc.h"
#include "header/temperature.h"
#include "header/clocktimer.h"
#include "header/config.h"
#include "header/control.h"
#include "header/display.h"
#include "header/relais.h"
#include "stdio.h"
#include "utils/lwiplib.h"
#include "header/myssi.h"


//*****************************************************************************
//Extern variables for Clock
//*****************************************************************************
extern volatile unsigned long clock_msec;
extern volatile unsigned long clock_sec;
extern volatile unsigned long clock_min;
extern volatile unsigned long clock_hour;
extern volatile unsigned long clock_day;
extern volatile unsigned long clock_month;
extern volatile unsigned long clock_year;

//*****************************************************************************
//Extern variable for current active power relay
//*****************************************************************************
extern volatile unsigned int power_relay_active;
extern volatile unsigned int cycle_active;
extern volatile unsigned int measurement_active;
extern volatile unsigned int current_cycle_step;
extern volatile int gBat_charge;
//*****************************************************************************
//Extern variable for current, cell voltages and temperatures
//*****************************************************************************
extern volatile int g_iCurrent;
extern volatile unsigned int g_iCellVoltages[12];
extern volatile int g_iCellTemperature[12];

//Variables for cat command execution in main
extern int g_cat_argc;
extern char g_cat_argv[], g_cat_flag;

//Minute Counter for maximum Time
extern volatile int max_time_counter;

extern volatile unsigned int wait_start_time;

```

```
108 //Mutex to lock SSI1 access
109 extern volatile char g_cSsi1Mutex;
110
111 //Clock divisor for systick
112 volatile unsigned char g_cTempMeasurementDiv = 0;
113 //Extern variable for temperature measurement flag
114 extern volatile int g_bMeasureTemp;
115
116 // Calibration Flag
117 extern volatile char g_Flag_calibrationModeOn;
118
119 //*****************************************************************************
120 //Variable for filename for measurment saving
121 //*****************************************************************************
122 char MEAS_FILE[64];
123
124
125
126 ////*****************************************************************************
127 ////
128 //// Defines the size of the buffers that hold the path, or temporary
129 //// data from the SD card.  There are two buffers allocated of this size.
130 //// The buffer size must be large enough to hold the longest expected
131 //// full path name, including the file name, and a trailing null character.
132 ////
133 ////*****************************************************************************
134 //#define PATH_BUF_SIZE   80
135
136
137 ////*****************************************************************************
138 ////
139 //// This buffer holds the full path to the current working directory.
140 //// Initially it is root ("/").
141 ////
142 ////*****************************************************************************
143 //static char g_cCwdBuf[PATH_BUF_SIZE] = "/";
144 //
145 ////*****************************************************************************
146 ////
147 //// A temporary data buffer used when manipulating file paths, or reading data
148 //// from the SD card.
149 ////
150 ////*****************************************************************************
151 //static char g_cTmpBuf[PATH_BUF_SIZE];
152 //
153 //
154 ////*****************************************************************************
155 ////
156 //// The following are data structures used by FatFs.
157 ////
158 ////*****************************************************************************
159 //static FATFS g_sFatFs;
160 //static DIR g_sDirObject;
161 //static FILINFO g_sFileInfo;
162 //static FIL g_sFileObject;
163
164
165 //*****************************************************************************
166 //
167 // A structure that holds a mapping between an FRESULT numerical code,
168 // and a string represenation.  FRESULT codes are returned from the FatFs
169 // FAT file system driver.
170 //
171 //*****************************************************************************
172 typedef struct
173 {
174     FRESULT fresult;
175     char *pcResultStr;
176 }
177 tFresultString;
178
179 //*****************************************************************************
180 //
181 // A macro to make it easy to add result codes to the table.
182 //
183 //*****************************************************************************
184 #define FRESULT_ENTRY(f)        { (f), (#f) }
185
186 //*****************************************************************************
187 //
188 // A table that holds a mapping between the numerical FRESULT code and
189 // it's name as a string.  This is used for looking up error codes for
190 // printing to the console.
191 //
192 //*****************************************************************************
```

```
193 tFresultString g_sFresultStrings[] =
194 {
195     FRESULT_ENTRY(FR_OK),
196     FRESULT_ENTRY(FR_NOT_READY),
197     FRESULT_ENTRY(FR_NO_FILE),
198     FRESULT_ENTRY(FR_NO_PATH),
199     FRESULT_ENTRY(FR_INVALID_NAME),
200     FRESULT_ENTRY(FR_INVALID_DRIVE),
201     FRESULT_ENTRY(FR_DENIED),
202     FRESULT_ENTRY(FR_EXIST),
203     FRESULT_ENTRY(FR_RW_ERROR),
204     FRESULT_ENTRY(FR_WRITE_PROTECTED),
205     FRESULT_ENTRY(FR_NOT_ENABLED),
206     FRESULT_ENTRY(FR_NO_FILESYSTEM),
207     FRESULT_ENTRY(FR_INVALID_OBJECT),
208     FRESULT_ENTRY(FR_MKFS_ABORTED)
209 };
210
211 //*****************************************************************************
212 //
213 // A macro that holds the number of result codes.
214 //
215 //*****************************************************************************
216 #define NUM_FRESULT_CODES (sizeof(g_sFresultStrings) / sizeof(tFresultString))
217
218 //*****************************************************************************
219 //
220 // This function returns a string representation of an error code
221 // that was returned from a function call to FatFs.  It can be used
222 // for printing human readable error messages.
223 //
224 //*****************************************************************************
225 const char *
226 StringFromFresult(FRESULT fresult)
227 {
228     unsigned int uIdx;
229
230     //
231     // Enter a loop to search the error code table for a matching
232     // error code.
233     //
234     for(uIdx = 0; uIdx < NUM_FRESULT_CODES; uIdx++)
235     {
236         //
237         // If a match is found, then return the string name of the
238         // error code.
239         //
240         if(g_sFresultStrings[uIdx].fresult == fresult)
241         {
242             return(g_sFresultStrings[uIdx].pcResultStr);
243         }
244     }
245
246     //
247     // At this point no matching code was found, so return a
248     // string indicating unknown error.
249     //
250     return("UNKNOWN ERROR CODE");
251 }
252
253
254 //*****************************************************************************
255 //
256 // This is the table that holds the command names, implementing functions,
257 // and brief description.
258 //
259 //*****************************************************************************
260 tCmdLineEntry g_sBrowserCmdTable[] =
261 {
262     { "help",   Cmd_help,       " : Display list of commands" },
263     { "h",      Cmd_help,    "    : alias for help" },
264     { "?",      Cmd_help,    "    : alias for help" },
265     { "reinit", Cmd_reinit,      ": Re initialize the SD-Card filesystem" },
266     { "ls",     Cmd_ls,       "   : Display list of files" },
267     { "chdir",  Cmd_cd,          ": Change directory" },
268     { "cd",     Cmd_cd,       "   : alias for chdir" },
269     { "pwd",    Cmd_pwd,       "  : Show current working directory" },
270     { "cat",    Cmd_cat,       "  : Show contents of a text file (use only in Idle Mode)" },
271     { "cre",    Cmd_cre,       "  : Create a file" },
272     { "del",    Cmd_del,       "  : Delete a file" },
273     { "exit",   Cmd_sdcard_exit,      " : Exit SD-Card browser" },
274     { 0, 0, 0 }
275 };
276
277 extern tCmdLineEntry g_sMainCmdTable;
```

```
278 extern tCmdLineEntry *g_psCmdTable;
279
280 // string for main location
281 extern const char *g_cMainLocalBuf;
282
283 // Global location buffer
284 extern char *g_cLocalBuf;
285
286 // TCP-Socket für die aktuelle Steuerungs-Verbindung
287 extern struct tcp_pcb* control_connection;
288
289 // String buffer for print operations to UART and Ethernet
290 char print_buffer[80];
291
292 //define string for SD-Card location
293 char *g_cSdLocalBuf  = "SD-Card";
294
295
296 //*****************************************************************************
297 //
298 // This is the handler for this SysTick interrupt.  FatFs requires a
299 // timer tick every 10 ms for internal timing purposes.
300 //
301 //*****************************************************************************
302 void
303 SysTickHandler(void)
304 {
305     //
306     // Call the FatFs tick timer.
307     //
308  TimerClock();
309  // Do not tick the FatFS if SSI1 is beeing used for another component (ADC)
310  if (!g_cSsi1Mutex) {
311   g_cSsi1Mutex = 1;
312   disk_timerproc();
313   g_cSsi1Mutex = 0;
314  }
315     lwIPTimer(10);
316
317     //Triger a temperature measurement every second
318 //     if (!((g_cTempMeasurementDiv++) %100)) {
319
320 if (g_cTempMeasurementDiv++>=100) {
321      g_bMeasureTemp = 1; //Set flag
322      get_temperature(0);
323      g_cTempMeasurementDiv = 0; //Reset
324     }
325 }
326
327 //*****************************************************************************
328 //
329 // This function implements the "re-init" command.
330 // The SD-Card filesystem will be reinitialized after plugging in the SD-Card
331 // while the system is running
332 //
333 //*****************************************************************************
334 int
335 Cmd_reinit(int argc, char *argv[])
336 {
337    mysdcardinit();
338    return 0;
339 }
340
341 //*****************************************************************************
342 //
343 // This function implements the "ls" command.  It opens the current
344 // directory and enumerates through the contents, and prints a line for
345 // each item it finds.  It shows details such as file attributes, time and
346 // date, and the file size, along with the name.  It shows a summary of
347 // file sizes at the end along with free space.
348 //
349 //*****************************************************************************
350 int
351 Cmd_ls(int argc, char *argv[])
352 {
353
354     unsigned long ulTotalSize;
355     unsigned long ulFileCount;
356     unsigned long ulDirCount;
357     FRESULT fresult;
358     FATFS *pFatFs;
359
360     //
361     // Open the current directory for access.
362     //
```

```
363     fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
364
365     //
366     // Check for error and return if there is a problem.
367     //
368     if(fresult != FR_OK)
369     {
370         return(fresult);
371     }
372
373     ulTotalSize = 0;
374     ulFileCount = 0;
375     ulDirCount = 0;
376
377     //
378     // Give an extra blank line before the listing.
379     //
380     UARTprintf("\n");
381
382     //
383     // Enter loop to enumerate through all directory entries.
384     //
385     for(;;)
386     {
387         //
388         // Read an entry from the directory.
389         //
390         fresult = f_readdir(&g_sDirObject, &g_sFileInfo);
391
392         //
393         // Check for error and return if there is a problem.
394         //
395         if(fresult != FR_OK)
396         {
397          sprintf(print_buffer, "Error at reading File System Entry ");
398
399          UARTprintf(print_buffer);     // debug information
400             if(control_connection){
401           telnet_write(print_buffer);
402             }
403          return(fresult);
404         }
405
406         //
407         // If the file name is blank, then this is the end of the
408         // listing.
409         //
410         if(!g_sFileInfo.fname[0])
411         {
412          break;
413         }
414
415         //
416         // If the attribue is directory, then increment the directory count.
417         //
418         if(g_sFileInfo.fattrib & AM_DIR)
419         {
420             ulDirCount++;
421         }
422
423         //
424         // Otherwise, it is a file.  Increment the file count, and
425         // add in the file size to the total.
426         //
427         else
428         {
429             ulFileCount++;
430             ulTotalSize += g_sFileInfo.fsize;
431         }
432
433         //
434         // Print the entry information on a single line with formatting
435         // to show the attributes, date, time, size, and name.
436         //
437         sprintf(print_buffer, "%c%c%c%c%c %u/%02u/%02u %02u:%02u %9u  %s\n",
438                                (g_sFileInfo.fattrib & AM_DIR) ? 'D' : '-',
439                                (g_sFileInfo.fattrib & AM_RDO) ? 'R' : '-',
440                                (g_sFileInfo.fattrib & AM_HID) ? 'H' : '-',
441                                (g_sFileInfo.fattrib & AM_SYS) ? 'S' : '-',
442                                (g_sFileInfo.fattrib & AM_ARC) ? 'A' : '-',
443                                (g_sFileInfo.fdate >> 9) + 1980,
444                                (g_sFileInfo.fdate >> 5) & 15,
445                                 g_sFileInfo.fdate & 31,
446                                (g_sFileInfo.ftime >> 11),
447                                (g_sFileInfo.ftime >> 5) & 63,
```

```
448                                 g_sFileInfo.fsize,
449                                 g_sFileInfo.fname);
450
451             UARTprintf(print_buffer);
452             if(control_connection){
453           telnet_write(print_buffer);
454             }
455         }   // endfor
456
457         //
458         // Print summary lines showing the file, dir, and size totals.
459         //
460      sprintf(print_buffer, "\n%4u File(s),%10u bytes total\n%4u Dir(s)",
461                 ulFileCount, ulTotalSize, ulDirCount);
462      UARTprintf(print_buffer);
463         if(control_connection){
464       telnet_write(print_buffer);
465         }
466         //
467         // Get the free space.
468         //
469         fresult = f_getfree("/", &ulTotalSize, &pFatFs);
470
471         //
472         // Check for error and return if there is a problem.
473         //
474         if(fresult != FR_OK)
475         {
476          //UARTprintf(" Error at the end of the line ");  // debug information
477             return(fresult);
478         }
479
480         //
481         // Display the amount of free space that was calculated.
482         //
483      sprintf(print_buffer, ", %10uK bytes free\n", ulTotalSize * pFatFs->sects_clust / 2);
484
485      UARTprintf(print_buffer);
486         if(control_connection){
487       telnet_write(print_buffer);
488         }
489
490
491         //
492         // Made it to here, return with no errors.
493         //
494         return(0);
495     }
496
497     //*****************************************************************************
498     //
499     // This function implements the "cd" command.  It takes an argument
500     // that specifes the directory to make the current working directory.
501     // Path separators must use a forward slash "/".  The argument to cd
502     // can be one of the following:
503     // * root ("/")
504     // * a fully specified path ("/my/path/to/mydir")
505     // * a single directory name that is in the current directory ("mydir")
506     // * parent directory ("..")
507     //
508     // It does not understand relative paths, so dont try something like this:
509     // ("../my/new/path")
510     //
511     // Once the new directory is specified, it attempts to open the directory
512     // to make sure it exists.  If the new path is opened successfully, then
513     // the current working directory (cwd) is changed to the new path.
514     //
515     //*****************************************************************************
516     int
517     Cmd_cd(int argc, char *argv[])
518     {
519         unsigned int uIdx;
520         FRESULT fresult;
521
522         //
523         // Copy the current working path into a temporary buffer so
524         // it can be manipulated.
525         //
526         strcpy(g_cTmpBuf, g_cCwdBuf);
527
528         //
529         // If the first character is /, then this is a fully specified
530         // path, and it should just be used as-is.
531         //
532         if(argv[1][0] == '/')
```

```
533     {
534         //
535         // Make sure the new path is not bigger than the cwd buffer.
536         //
537         if(strlen(argv[1]) + 1 > sizeof(g_cCwdBuf))
538         {
539          sprintf(print_buffer, "Resulting path name is too long\n");
540 
541             UARTprintf(print_buffer);
542             if(control_connection){
543           telnet_write(print_buffer);
544             }
545             return(0);
546         }
547 
548         //
549         // If the new path name (in argv[1])  is not too long, then
550         // copy it into the temporary buffer so it can be checked.
551         //
552         else
553         {
554             strncpy(g_cTmpBuf, argv[1], sizeof(g_cTmpBuf));
555         }
556     }
557 
558     //
559     // If the argument is .. then attempt to remove the lowest level
560     // on the CWD.
561     //
562     else if(!strcmp(argv[1], ".."))
563     {
564         //
565         // Get the index to the last character in the current path.
566         //
567         uIdx = strlen(g_cTmpBuf) - 1;
568 
569         //
570         // Back up from the end of the path name until a separator (/)
571         // is found, or until we bump up to the start of the path.
572         //
573         while((g_cTmpBuf[uIdx] != '/') && (uIdx > 1))
574         {
575             //
576             // Back up one character.
577             //
578             uIdx--;
579         }
580 
581         //
582         // Now we are either at the lowest level separator in the
583         // current path, or at the beginning of the string (root).
584         // So set the new end of string here, effectively removing
585         // that last part of the path.
586         //
587         g_cTmpBuf[uIdx] = 0;
588     }
589 
590     //
591     // Otherwise this is just a normal path name from the current
592     // directory, and it needs to be appended to the current path.
593     //
594     else
595     {
596         //
597         // Test to make sure that when the new additional path is
598         // added on to the current path, there is room in the buffer
599         // for the full new path.  It needs to include a new separator,
600         // and a trailing null character.
601         //
602         if(strlen(g_cTmpBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cCwdBuf))
603         {
604          sprintf(print_buffer, "Resulting path name is too long\n");
605 
606             UARTprintf(print_buffer);
607 
608             if(control_connection){
609           telnet_write(print_buffer);
610             }
611             return(0);
612         }
613 
614         //
615         // The new path is okay, so add the separator and then append
616         // the new directory to the path.
617         //
```

```
618         else
619         {
620             //
621             // If not already at the root level, then append a /
622             //
623             if(strcmp(g_cTmpBuf, "/"))
624             {
625                 strcat(g_cTmpBuf, "/");
626             }
627
628             //
629             // Append the new directory to the path.
630             //
631             strcat(g_cTmpBuf, argv[1]);
632         }
633     }
634
635     //
636     // At this point, a candidate new directory path is in chTmpBuf.
637     // Try to open it to make sure it is valid.
638     //
639     fresult = f_opendir(&g_sDirObject, g_cTmpBuf);
640
641     //
642     // If it cant be opened, then it is a bad path.  Inform
643     // user and return.
644     //
645     if(fresult != FR_OK)
646     {
647      sprintf(print_buffer, "cd: %s\n", g_cTmpBuf);
648
649         UARTprintf(print_buffer);
650
651         if(control_connection){
652       telnet_write(print_buffer);
653         }
654         return(fresult);
655     }
656
657     //
658     // Otherwise, it is a valid new path, so copy it into the CWD.
659     //
660     else
661     {
662         strncpy(g_cCwdBuf, g_cTmpBuf, sizeof(g_cCwdBuf));
663     }
664
665
666     //
667     // Return success.
668     //
669     return(0);
670 }
671
672 //*****************************************************************************
673 //
674 // This function implements the "pwd" command.  It simply prints the
675 // current working directory.
676 //
677 //*****************************************************************************
678 int
679 Cmd_pwd(int argc, char *argv[])
680 {
681
682     //
683     // Print the CWD to the console.
684     //
685  sprintf(print_buffer, "%s\n", g_cCwdBuf);
686
687     UARTprintf(print_buffer);
688
689     if(control_connection){
690   telnet_write(print_buffer);
691     }
692
693
694     //
695     // Return success.
696     //
697     return(0);
698 }
699
700 //*****************************************************************************
701 //
702 // This function implements the "cat" command.  It reads the contents of
```

```
// a file and prints it to the console.  This should only be used on
// text files.  If it is used on a binary file, then a bunch of garbage
// is likely to printed on the console.
//
//*****************************************************************************
int
Cmd_cat(int argc, char *argv[])
{
 //set flag for execution in main loop
 g_cat_flag = 1;

 //hand over argc and argv
 g_cat_argc = argc;
 strcpy(g_cat_argv, argv[1]);

    //
    // Return success.
    //
    return(0);
}

//*****************************************************************************
//
// This function implements the "del" command.  It deletes a file
//
//*****************************************************************************
int
Cmd_del(int argc, char *argv[])
{

    FRESULT fresult;

    //
    // First, check to make sure that the current path (CWD), plus
    // the file name, plus a separator and trailing null, will all
    // fit in the temporary buffer that will be used to hold the
    // file name.  The file name must be fully specified, with path,
    // to FatFs.
    //
    if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
    {
     sprintf(print_buffer, "Resulting path name is too long\n");

        UARTprintf(print_buffer);

        if(control_connection){
      telnet_write(print_buffer);
        }
        return(0);
    }

    //
    // Copy the current path to the temporary buffer so it can be manipulated.
    //
    strcpy(g_cTmpBuf, g_cCwdBuf);

    //
    // If not already at the root level, then append a separator.
    //
    if(strcmp("/", g_cCwdBuf))
    {
        strcat(g_cTmpBuf, "/");
    }

    //
    // Now finally, append the file name to result in a fully specified file.
    //
    strcat(g_cTmpBuf, argv[1]);

    //
     // Delete File
     //
       fresult = f_unlink(g_cTmpBuf);

       //
       // Check if operation succeeded
       //

       if(fresult != FR_OK)
       {
           return(fresult);
       }


```

```
788     //
789     // Return success.
790     //
791     return(0);
792
793 }
794
795 //*****************************************************************************
796 //
797 // This function implements the "cre" command.  It creates a file
798 //
799 //*****************************************************************************
800 int
801 Cmd_cre(int argc, char *argv[])
802 {
803
804     FRESULT fresult;
805     FIL fnew;       /* new file object */
806
807     //
808     // First, check to make sure that the current path (CWD), plus
809     // the file name, plus a separator and trailing null, will all
810     // fit in the temporary buffer that will be used to hold the
811     // file name.  The file name must be fully specified, with path,
812     // to FatFs.
813     //
814     if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
815     {
816      sprintf(print_buffer, "Resulting path name is too long\n");
817
818         UARTprintf(print_buffer);
819
820         if(control_connection){
821       telnet_write(print_buffer);
822         }
823         return(0);
824     }
825
826     //
827     // Copy the current path to the temporary buffer so it can be manipulated.
828     //
829     strcpy(g_cTmpBuf, g_cCwdBuf);
830
831     //
832     // If not already at the root level, then append a separator.
833     //
834     if(strcmp("/", g_cCwdBuf))
835     {
836         strcat(g_cTmpBuf, "/");
837     }
838
839     //
840     // Now finally, append the file name to result in a fully specified file.
841     //
842     strcat(g_cTmpBuf, argv[1]);
843
844     //
845     // Create the File
846     //
847     fresult = f_open(&fnew, argv[1], FA_CREATE_ALWAYS | FA_WRITE );
848
849     //
850     // Check if creation succeeded
851     //
852
853     if(fresult != FR_OK)
854     {
855         return(fresult);
856     }
857
858     /* Close opened files */
859     f_close(&fnew);
860
861
862     //
863     // Return success.
864     //
865     return(0);
866
867 }
868
869 //*****************************************************************************
870 //
871 // This function implements the "help" command.  It prints a simple list
872 // of the available commands with a brief description.
```

```
873 //
874 //*****************************************************************************
875 int
876 Cmd_help(int argc, char *argv[])
877 {
878
879     tCmdLineEntry *pEntry;
880
881     //
882     // Print some header text.
883     //
884
885     UARTprintf("\nAvailable commands\n");
886     UARTprintf("------------------\n");
887
888     if(control_connection){
889      telnet_write("Available commands\n");
890      telnet_write("------------------\n");
891     }
892
893     //
894     // Point at the beginning of the command table.
895     //
896     pEntry = g_psCmdTable;
897
898     //
899     // Enter a loop to read each entry from the command table.  The
900     // end of the table has been reached when the command name is NULL.
901     //
902     while(pEntry->pcCmd)
903     {
904         //
905         // Print the command name and the brief description.
906         //
907      sprintf(print_buffer, "%s%s\n", pEntry->pcCmd, pEntry->pcHelp);
908
909         UARTprintf(print_buffer);
910         if(control_connection){
911       telnet_write(print_buffer);
912         }
913
914         //
915         // Advance to the next entry in the table.
916         //
917         pEntry++;
918     }
919
920
921     //
922     // Return success.
923     //
924     return(0);
925 }
926
927 int Cmd_sdcard_exit(int argc, char *argv[])
928 {
929
930     // set Command line pointer to the beginning of the main command structure
931     g_psCmdTable = &g_sMainCmdTable;
932
933     // set location buffer to main
934     g_cLocalBuf = (char*)g_cMainLocalBuf;
935
936  return(0);
937 }
938
939 //*****************************************************************************
940 //
941 // This is the table that holds the command names, implementing functions,
942 // and brief description.
943 //
944 //*****************************************************************************
945 //extern tCmdLineEntry g_sCmdTable[];
946
947
948 //*****************************************************************************
949 //
950 // Own Initialization of SSI0, SysTick, FatFs and SDcard
951 //
952 //*****************************************************************************
953
954 void mysdcardinit(void)
955 {
956
957  // variable for FatFs results
```

```
958     FRESULT fresult = FR_NOT_READY;
959
960     UARTprintf("SD card initialization...");
961
962     // Configure SysTick for a 100Hz interrupt.  The FatFs driver
963     // wants a 10 ms tick.
964
965     SysTickPeriodSet(SysCtlClockGet() / 100);
966     SysTickEnable();
967     SysTickIntEnable();
968
969     fresult = f_mount(0, &g_sFatFs);
970
971     if(fresult != FR_OK)
972     {
973         UARTprintf("f_mount error: %s\n", StringFromFresult(fresult));
974     }
975     // debugging stuff
976     //else
977     //{
978     //    UARTprintf(" f_mount successful\n");
979     //}
980
981     // reset status flag
982     fresult = FR_NOT_READY;
983
984     // Open the current directory for access.
985     fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
986
987     // Check for error and return if there is a problem.
988     if(fresult != FR_OK)
989     {
990         UARTprintf(" f_opendir error: %s\n", StringFromFresult(fresult));
991     }
992     else
993     {
994         UARTprintf("done.\n");
995     }
996     // debugging stuff
997  //else
998     //{
999     //    UARTprintf(" f_opendir successful\n");
1000    //}
1001 }
1002
1003
1004
1005
1006
1007 //*****************************************************************************
1008 //
1009 // This function implements the "read_File" command.  It reads the contents of
1010 // a file and prints it to the console.  This should only be used on
1011 // text files.  If it is used on a binary file, then a bunch of garbage
1012 // is likely to printed on the console.
1013 //
1014 //*****************************************************************************
1015 int read_file(const char *filename)
1016 {
1017
1018     FRESULT fresult;
1019     unsigned short usBytesRead;
1020
1021     //
1022     // First, check to make sure that the current path (CWD), plus
1023     // the file name, plus a separator and trailing null, will all
1024     // fit in the temporary buffer that will be used to hold the
1025     // file name.  The file name must be fully specified, with path,
1026     // to FatFs.
1027     //
1028     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1029     {
1030      UARTprintf("Resulting path name is too long\n");
1031         return(0);
1032     }
1033
1034     //
1035     // Copy the current path to the temporary buffer so it can be manipulated.
1036     //
1037     strcpy(g_cTmpBuf, g_cCwdBuf);
1038
1039     //
1040     // If not already at the root level, then append a separator.
1041     //
1042     if(strcmp("/", g_cCwdBuf))
```

```
1043        {
1044            strcat(g_cTmpBuf, "/");
1045        }
1046
1047        //
1048        // Now finally, append the file name to result in a fully specified file.
1049        //
1050        strcat(g_cTmpBuf, filename);
1051
1052        //
1053        // Open the file for reading.
1054        //
1055        fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1056
1057        //
1058        // If there was some problem opening the file, then return
1059        // an error.
1060        //
1061        if(fresult != FR_OK)
1062        {
1063            return(fresult);
1064        }
1065
1066        //
1067        // Enter a loop to repeatedly read data from the file and display it,
1068        // until the end of the file is reached.
1069        //
1070        do
1071        {
1072            //
1073            // Read a block of data from the file.  Read as much as can fit
1074            // in the temporary buffer, including a space for the trailing null.
1075            //
1076            fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf) - 1,
1077                             &usBytesRead);
1078
1079            //
1080            // If there was an error reading, then print a newline and
1081            // return the error to the user.
1082            //
1083            if(fresult != FR_OK)
1084            {
1085             UARTprintf("\n");
1086                return(fresult);
1087            }
1088
1089            //
1090            // Null terminate the last block that was read to make it a
1091            // null terminated string that can be used with printf.
1092            //
1093            g_cTmpBuf[usBytesRead] = 0;
1094
1095            //
1096            // Print the last chunk of the file that was received.
1097            //
1098
1099            UARTprintf("%s", g_cTmpBuf);
1100
1101
1102            //
1103            // Wait for the UART transmit buffer to empty.
1104            //
1105
1106      #if defined(UART_BUFFERED)
1107             UARTFlushTx(false);
1108      #endif
1109
1110        //
1111        // Continue reading until less than the full number of bytes are
1112        // read.  That means the end of the buffer was reached.
1113        //
1114        }
1115        while(usBytesRead == sizeof(g_cTmpBuf) - 1);
1116
1117        /* Close opened files */
1118        f_close(&g_sFileObject);
1119
1120
1121        //
1122        // Return success.
1123        //
1124        return(0);
1125    }
1126
1127    //*****************************************************************************
```

```
1128 //
1129 // This function implements the "read_config" command.  It reads the contents of
1130 // a file and prints it to the console.  This should only be used on
1131 // text files.  If it is used on a binary file, then a bunch of garbage
1132 // is likely to printed on the console.
1133 //
1134 //*****************************************************************************
1135 int read_into_buffer(const char *filename, char *buffer)
1136 {
1137
1138     FRESULT fresult;
1139     unsigned short usBytesRead;
1140
1141     //
1142     // First, check to make sure that the current path (CWD), plus
1143     // the file name, plus a separator and trailing null, will all
1144     // fit in the temporary buffer that will be used to hold the
1145     // file name.  The file name must be fully specified, with path,
1146     // to FatFs.
1147     //
1148     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1149     {
1150      UARTprintf("Resulting path name is too long\n");
1151         return(0);
1152     }
1153
1154     //
1155     // Copy the current path to the temporary buffer so it can be manipulated.
1156     //
1157     strcpy(g_cTmpBuf, g_cCwdBuf);
1158
1159     //
1160     // If not already at the root level, then append a separator.
1161     //
1162     if(strcmp("/", g_cCwdBuf))
1163     {
1164         strcat(g_cTmpBuf, "/");
1165     }
1166
1167     //
1168     // Now finally, append the file name to result in a fully specified file.
1169     //
1170     strcat(g_cTmpBuf, filename);
1171
1172     //
1173     // Open the file for reading.
1174     //
1175     fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1176
1177     //
1178     // If there was some problem opening the file, then return
1179     // an error.
1180     //
1181     if(fresult != FR_OK)
1182     {
1183      UARTprintf("problem opening the file\n");
1184         return(fresult);
1185     }
1186
1187     //
1188     // Enter a loop to repeatedly read data from the file and display it,
1189     // until the end of the file is reached.
1190     //
1191     char *p = buffer;
1192     do
1193     {
1194         //
1195         // Read a block of data from the file.  Read as much as can fit
1196         // in the temporary buffer, including a space for the trailing null.
1197         //
1198
1199
1200      // Änderung am 3.3. KRR
1201      // fresult = f_read(&g_sFileObject, p, sizeof(p) - 1,
1202         //                 &usBytesRead);
1203         fresult = f_read(&g_sFileObject, p, CONF_BUF_SIZE - 1,
1204                          &usBytesRead);
1205
1206         //
1207         // If there was an error reading, then print a newline and
1208         // return the error to the user.
1209         //
1210         if(fresult != FR_OK)
1211         {
1212          UARTprintf("\n");
```

```
1213            return(fresult);
1214        }
1215
1216        //
1217        // Null terminate the last block that was read to make it a
1218        // null terminated string that can be used with printf.
1219        //
1220        p[usBytesRead] = 0;
1221
1222        //
1223        // Print the last chunk of the file that was received.
1224        //
1225        //UARTprintf("%s", p);
1226        p += usBytesRead;
1227
1228        //
1229        // Wait for the UART transmit buffer to empty.
1230        //
1231  #if defined(UART_BUFFERED)
1232        UARTFlushTx(false);
1233  #endif
1234
1235    //
1236    // Continue reading until less than the full number of bytes are
1237    // read.  That means the end of the buffer was reached.
1238    //
1239    }
1240    //Olli while(usBytesRead == sizeof(p) - 1);
1241    while(usBytesRead == CONF_BUF_SIZE - 1);
1242    /* Close opened files */
1243    fresult = f_close(&g_sFileObject);
1244
1245    if(fresult != FR_OK)
1246    {
1247        return(fresult);
1248    }
1249
1250
1251    //
1252    // Return success.
1253    //
1254    return (0);
1255 }
1256
1257
1258
1259 //
1260 // This function implements the "delete_File" command.
1261 // It simply delete a file on the SD Card, selected with its filename
1262 //
1263 int delete_file(const char *filename){
1264
1265
1266   FRESULT fresult;
1267
1268      //
1269      // First, check to make sure that the current path (CWD), plus
1270      // the file name, plus a separator and trailing null, will all
1271      // fit in the temporary buffer that will be used to hold the
1272      // file name.  The file name must be fully specified, with path,
1273      // to FatFs.
1274      //
1275      if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1276      {
1277       UARTprintf("Resulting path name is too long\n");
1278          return(0);
1279      }
1280
1281      //
1282      // Copy the current path to the temporary buffer so it can be manipulated.
1283      //
1284      strcpy(g_cTmpBuf, g_cCwdBuf);
1285
1286      //
1287      // If not already at the root level, then append a separator.
1288      //
1289      if(strcmp("/", g_cCwdBuf))
1290      {
1291          strcat(g_cTmpBuf, "/");
1292      }
1293
1294      //
1295      // Now finally, append the file name to result in a fully specified file.
1296      //
1297      strcat(g_cTmpBuf, filename);
```

```
1298
1299            //
1300            // Wait for the UART transmit buffer to empty.
1301            //
1302    #if defined(UART_BUFFERED)
1303        UARTFlushTx(false);
1304    #endif
1305
1306
1307
1308    //
1309    // Delete File
1310    //
1311      fresult = f_unlink(g_cTmpBuf);
1312
1313      //
1314      // Check if operation succeeded
1315      //
1316
1317      if(fresult != FR_OK)
1318      {
1319          return(fresult);
1320      }
1321
1322
1323      //
1324      // Return success.
1325      //
1326      return(0);
1327
1328  }
1329
1330
1331  //*****************************************************************************
1332  //
1333  // This function implements the "write_to_file" command.  It overrides the content of
1334  // a file with new input.
1335  //
1336  //*****************************************************************************
1337
1338  int write_to_file(const char *filename, const char *write_Buffer){
1339
1340
1341    FIL fnew;       /* new file object */
1342    FRESULT fresult;
1343    unsigned short bw = 0;
1344
1345
1346       //
1347       // First, check to make sure that the current path (CWD), plus
1348       // the file name, plus a separator and trailing null, will all
1349       // fit in the temporary buffer that will be used to hold the
1350       // file name.  The file name must be fully specified, with path,
1351       // to FatFs.
1352       //
1353       if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1354       {
1355        UARTprintf("Resulting path name is too long\n");
1356           return(0);
1357       }
1358
1359       //
1360       // Copy the current path to the temporary buffer so it can be manipulated.
1361       //
1362       strcpy(g_cTmpBuf, g_cCwdBuf);
1363
1364       //
1365       // If not already at the root level, then append a separator.
1366       //
1367       if(strcmp("/", g_cCwdBuf))
1368       {
1369           strcat(g_cTmpBuf, "/");
1370       }
1371
1372       //
1373       // Now finally, append the file name to result in a fully specified file.
1374       //
1375       strcat(g_cTmpBuf, filename);
1376
1377           //
1378           // Wait for the UART transmit buffer to empty.
1379           //
1380    #if defined(UART_BUFFERED)
1381        UARTFlushTx(false);
1382    #endif
```

```
1383
1384
1385    //
1386    // Create File
1387    //
1388
1389      fresult = f_open(&fnew, g_cTmpBuf, FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
1390
1391
1392
1393      if(fresult != FR_OK)
1394      {
1395          return(fresult);
1396      }
1397
1398
1399
1400      fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1401
1402      if(fresult != FR_OK)
1403      {
1404          return(fresult);
1405      }
1406
1407
1408      /* Close opened files */
1409      f_close(&fnew);
1410
1411
1412      //
1413      // Return success.
1414      //
1415      return(0);
1416
1417  }
1418
1419
1420  //*****************************************************************************
1421  //
1422  // This function implements the "add_to_file" command.  It appends new input to
1423  // an existing file, selected by its filename. If file not exists, file is being created.
1424  //
1425  //*****************************************************************************
1426
1427  int add_to_file(const char *filename, const char *write_Buffer){
1428
1429
1430   FIL fnew;       /* new file object */
1431   FRESULT fresult;
1432   unsigned short bw = 0;
1433
1434
1435      //
1436      // First, check to make sure that the current path (CWD), plus
1437      // the file name, plus a separator and trailing null, will all
1438      // fit in the temporary buffer that will be used to hold the
1439      // file name.  The file name must be fully specified, with path,
1440      // to FatFs.
1441      //
1442      if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1443      {
1444       UARTprintf("Resulting path name is too long\n");
1445          return(0);
1446      }
1447
1448      //
1449      // Copy the current path to the temporary buffer so it can be manipulated.
1450      //
1451      strcpy(g_cTmpBuf, g_cCwdBuf);
1452
1453      //
1454      // If not already at the root level, then append a separator.
1455      //
1456      if(strcmp("/", g_cCwdBuf))
1457      {
1458          strcat(g_cTmpBuf, "/");
1459      }
1460
1461      //
1462      // Now finally, append the file name to result in a fully specified file.
1463      //
1464      strcat(g_cTmpBuf, filename);
1465
1466      //
1467      // Wait for the UART transmit buffer to empty.
```

```
1468        //
1469    #if defined(UART_BUFFERED)
1470        UARTFlushTx(false);
1471    #endif
1472
1473
1474    //
1475    // Open/Create File
1476    //
1477
1478     fresult = f_open(&fnew, g_cTmpBuf, FA_OPEN_ALWAYS | FA_WRITE |FA_READ);
1479
1480     if(fresult != FR_OK)
1481     {
1482         return(fresult);
1483     }
1484
1485     // Set Pointer to end of File
1486     f_lseek(&fnew,fnew.fsize);
1487
1488     // Write String to end of File
1489     fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1490     if(fresult != FR_OK)
1491     {
1492         return(fresult);
1493     }
1494
1495     /* Close opened files */
1496     f_close(&fnew);
1497
1498
1499     //
1500     // Return success.
1501     //
1502     return(0);
1503
1504
1505 }
1506
1507 //*****************************************************************************
1508 //
1509 // This function implements the "create_file" command.  It creates a file on
1510 // the current working directory with specified filename.
1511 //
1512 //*****************************************************************************
1513
1514 int create_file(const char *filename)
1515 {
1516
1517
1518  FIL fnew;       /* new file object */
1519  FRESULT fresult;
1520
1521     //
1522     // First, check to make sure that the current path (CWD), plus
1523     // the file name, plus a separator and trailing null, will all
1524     // fit in the temporary buffer that will be used to hold the
1525     // file name.  The file name must be fully specified, with path,
1526     // to FatFs.
1527     //
1528     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1529     {
1530      UARTprintf("Resulting path name is too long\n");
1531         return(0);
1532     }
1533
1534     //
1535     // Copy the current path to the temporary buffer so it can be manipulated.
1536     //
1537     strcpy(g_cTmpBuf, g_cCwdBuf);
1538
1539     //
1540     // If not already at the root level, then append a separator.
1541     //
1542     if(strcmp("/", g_cCwdBuf))
1543     {
1544         strcat(g_cTmpBuf, "/");
1545     }
1546
1547     //
1548     // Now finally, append the file name to result in a fully specified file.
1549     //
1550     strcat(g_cTmpBuf, filename);
1551
1552         //
```

```
1553          // Wait for the UART transmit buffer to empty.
1554          //
1555    #if defined(UART_BUFFERED)
1556      UARTFlushTx(false);
1557    #endif
1558
1559
1560  //
1561  // Create the File
1562  //
1563    fresult = f_open(&fnew, filename, FA_CREATE_ALWAYS | FA_WRITE );
1564
1565    //
1566    // Check if creation succeeded
1567    //
1568
1569    if(fresult != FR_OK)
1570    {
1571        return(fresult);
1572    }
1573
1574    /* Close opened files */
1575    f_close(&fnew);
1576
1577
1578      //
1579      // Return success.
1580      //
1581      return(0);
1582  }
1583
1584
1585
1586
1587  //*****************************************************************************
1588  //
1589  // Own variation of sdcard.c main function from stellarisware to implement an
1590  // command line based file explorer for browsing the SD Card
1591  //
1592  //*****************************************************************************
1593
1594  int my_start_cmd_line(int argc, char *argv[])
1595  {
1596      // set Command line pointer to the beginning of the SD-Card/Browser command structure
1597      g_psCmdTable = &g_sBrowserCmdTable[0];
1598
1599      sprintf(g_cSdLocalBuf, "SD-Card: %s>", g_cCwdBuf);
1600
1601      // set location buffer to Browser
1602      g_cLocalBuf = (char*)g_cSdLocalBuf;
1603
1604   return(0);
1605  }
1606
1607  //*****************************************************************************
1608  //
1609  // Main function for measurement of voltages, current and temperature. Multiplex
1610  // all relevant cells (depended on configuration) and write data to LogFile on
1611  // SD-Card.
1612  //
1613  //*****************************************************************************
1614  void do_measure(){
1615
1616  //*****************************************************************************
1617  // Use this commented version if current is needed measured with every cellvoltage
1618  //*****************************************************************************
1619  // unsigned int i = 0;
1620  // char buf[64];
1621  // //UARTprintf("\n");
1622  // for (i=1; i <= config.quantity_cells; i++){
1623  //   sprintf(buf, "%02d,%04d,%02d,%02d,%02d,%02d,%02d,uV,%07d,C,%03d,mA,%07d,%01d\n"
1624  //         , i, clock_year, clock_month, clock_day, clock_hour, clock_min, clock_sec
1625  //         ,adc_get_voltage(i), get_temperature(i), adc_get_current(), power_relay_active);
1626  //
1627  //   //UARTprintf(buf);
1628  //   add_to_file(MEAS_FILE, buf);
1629  //
1630  ////   if(control_connection){
1631  ////    telnet_write(buf);
1632  ////   }
1633  //
1634  // }
1635  //*****************************************************************************
1636
1637   int i = 0;
```

```
1638  char buf[64];
1639  static unsigned char voltageErrorCount;
1640  static unsigned char currentErrorCount;
1641  UARTprintf("\n");
1642 // UARTprintf("\nvorReset: Offset:%d\tFullSreg:%d\n",adc_read_offset_reg(),adc_read_fs_reg());
1643  //Turn on Measure-LED
1644  GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  GPIO_PIN_5);
1645
1646
1647 /*
1648  //reset serial interface of the ADC
1649  // TODO: angucken
1650  spiChipSelect(ADC);
1651  SSIDataPut(SSI1_BASE, 0xFF);
1652  SSIDataPut(SSI1_BASE, 0xFF);
1653  SSIDataPut(SSI1_BASE, 0xFF);
1654  SSIDataPut(SSI1_BASE, 0xFF);
1655  spiChipSelect(NONE);
1656  //UARTprintf("\nNachReset: Offset:%d\tFullSreg:%d\n",adc_read_offset_reg(),adc_read_fs_reg());
1657 */
1658  // new myadc.c
1659
1660
1661
1662  adc_get_current();//measure current and load into global variable
1663
1664  if(f_adc_error ){ // Versuch: wenn ADC error Flag gesetzt
1665   myadc_reinit();
1666   adc_get_current();
1667   f_adc_error = 0;
1668  }
1669
1670
1671 // if(config.max_current < g_iCurrent){
1672 //  myadc_init();
1673 //  adc_get_current();
1674 //  UARTprintf("ADC-NeueMessung da Über Strom\n");
1675 // }
1676
1677
1678  if(cycle_active){//check current limt if cycling is active
1679   if(abs(g_iCurrent) > config.max_current){//Check if current doesnt exceed limit
1680    currentErrorCount++;
1681    UARTprintf("\n %d ADC-NeueMessung da Über-Strom \n",currentErrorCount);
1682   }
1683   else
1684    currentErrorCount = 0;
1685
1686   if  (currentErrorCount >= CURRENTCOUNTMAX){
1687    //Stop cycling immediately but continue measurement
1688    stop_cycling();
1689
1690    handlerState = ERRCURR;//Display error message
1691    UARTprintf("\nMAXIMUM CURRENT EXCEEDED!\n");
1692    if(control_connection){
1693     telnet_write("\nMAXIMUM CURRENT EXCEEDED!\n");
1694    }
1695   }
1696   else if(abs(gBat_charge += (g_iCurrent*config.cyclestep/1000)) > config.max_charge){//check if charge limit exceeded (mAsec)
1697    //Stop cycling immediately but continue measurement
1698    stop_cycling();
1699
1700    handlerState = ERRCHARGE;//Display error message
1701    UARTprintf("\nMAXIMUM CHARGE EXCEEDED!\n");
1702    if(control_connection){
1703     telnet_write("\nMAXIMUM CHARGE EXCEEDED!\n");
1704    }
1705   }
1706  }
1707
1708
1709
1710  //Get all the values
1711  for (i = 1; i <= config.quantity_cells; i++){
1712 //   if(measurement_active){
1713   if(1){
1714    //save measurements in global variable
1715
1716    adc_get_voltage(i);
1717
1718 //  Abfang des ADC_ErrorFlags
1719    if(f_adc_error ){ // Versuch
1720     myadc_reinit();
1721     adc_get_voltage(i);
1722     UARTprintf("ADC-Error-Flag gesetzt-Reinit und neue Messung \n");
```

```
1723      f_adc_error = 0;
1724     }
1725
1726  //  ADC Test Um evtl. Spannungsaussreißer einzufangen
1727  //    if (((g_iCellVoltages[i - 1]< config.min_voltage ) | (g_iCellVoltages[i - 1] > config.max_voltage)) && (g_Flag_calibrationModeOn
          ==0)){
1728  //     myadc_reinit();
1729  //     adc_get_voltage(i);
1730  //     UARTprintf("ADC-NeueMessung da Über/unter Spannung \n");
1731  //    }
1732
1733
1734
1735     //get_temperature(i);
1736
1737     sprintf(buf, "%02d,%04d,%02d,%02d,%02d,%02d,%02d,%07d,%03d,%07d,%01d\n"
1738             , i, clock_year, clock_month, clock_day, clock_hour, clock_min, clock_sec
1739             ,g_iCellVoltages[i-1], g_iCellTemperature[i-1], g_iCurrent, power_relay_active);
1740
1741     //save the values
1742     add_to_file(MEAS_FILE, buf);
1743     UARTprintf(buf);
1744     if(control_connection){
1745      telnet_write(buf);
1746     }
1747
1748
1749
1750     if(cycle_active){//check voltage and temperature limits and stop cycling
1751      if(g_iCellVoltages[i-1] < config.min_voltage || g_iCellVoltages[i-1] > config.max_voltage){//Check for Voltage and Temperature
             limits
1752       voltageErrorCount++;
1753       UARTprintf("\n %d ADC-NeueMessung da Über/Unter Spannung \n",voltageErrorCount);
1754      }
1755      else
1756       voltageErrorCount = 0;
1757
1758      if(voltageErrorCount >= VOLTCOUNTMAX){
1759       //Stop cycling immediately continue measurement
1760       stop_cycling();
1761
1762       handlerState = ERRVOLT;//Display message
1763       UARTprintf("\nVOLTAGE LIMITS EXCEEDED!\n");
1764       if(control_connection){
1765        telnet_write("\nVOLTAGE LIMITS EXCEEDED!\n");
1766       }
1767       voltageErrorCount = 0;
1768      }
1769      else if(g_iCellTemperature[i-1] > config.max_temp){
1770       //Stop cycling immediately continue measurement
1771       stop_cycling();
1772
1773       handlerState = ERRTEMP;//Display message
1774       UARTprintf("\nTEMPERATURE LIMIT EXCEEDED!\n");
1775       if(control_connection){
1776        telnet_write("\nTEMPERATURE LIMIT EXCEEDED!\n");
1777       }
1778      }
1779     }
1780    }
1781   }
1782   //Turn off Measure-LED
1783   GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  0x00);
1784
1785   if(cycle_active)
1786    UARTprintf("Cycling active, step %d of %d running\n", current_cycle_step, config.step_quantity);
1787   else
1788    UARTprintf("Cycling not active\n");
1789
1790  }
```

Quellcode E.13: Anhang/c/myssi.c

```
1  /*
2  Project:       battery_cycling_sw_v4
3  File:          my_ssi.c
4
5  Auhtor:        Thomas Wisnewski
6  Credits:       Matthias Schneider
7           Tobias Steinmann
8           Fabian Schwartau
9           Oliver Schmidt
10          Stellaris Ware
```

```
11
12 last modified:      2015/03/20
13
14 Project Status      Under Construction
15 Status:         running
16
17 CCS:         5.5.1.00031
18 Stellarisware:      8555
19
20 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
21          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
22          NTC-Connectors, MAX3232 and Suplly Circuits
23
24 Description:       Source Code for ssi1 initalization for use with SD Card and 16-bit ADC
25
26 */
27
28
29
30 /*****************************************************************************
31 *
32 * Own Includings
33 *
34 *****************************************************************************/
35
36
37 #include "inc/hw_ints.h"
38 #include "inc/hw_memmap.h"
39 #include "inc/hw_types.h"
40 #include "inc/hw_ssi.h"
41 #include "driverlib/debug.h"
42 #include "driverlib/gpio.h"
43 #include "driverlib/interrupt.h"
44 #include "driverlib/pin_map.h"
45 #include "driverlib/rom.h"
46 #include "driverlib/ssi.h"
47 #include "driverlib/sysctl.h"
48 #include "driverlib/uart.h"
49 #include "utils/uartstdio.h"
50 #include "header/myssi.h"
51 #include "third_party/fatfs/src/diskio.h"
52
53 //Mutex to lock SSI1 access
54 volatile char g_cSsi1Mutex = 0;
55
56
57 //*****************************************************************************
58 //
59 // Initialize SSI1-Interface
60 //
61 //*****************************************************************************
62 void ssi1_init(void)
63 {
64  // Output
65  UARTprintf("Initializing SS1...");
66  // enable SSI1
67     SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
68  // enable GPIO Port F
69  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
70  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
71  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
72     // Den Takt auf 2MHz stellen (hier ist noch Luft nach oben)
73     SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3, SSI_MODE_MASTER, 2000000, 8);   // ADC=SSI_FRF_MOTO_MODE_3
74  // Pin Configuration
75     GPIOPinConfigure(GPIO_PE0_SSI1CLK);
76  GPIOPinConfigure(GPIO_PF4_SSI1RX);
77  GPIOPinConfigure(GPIO_PF5_SSI1TX);
78     GPIOPinTypeSSI(GPIO_PORTF_BASE, GPIO_PIN_4 | GPIO_PIN_5);
79     GPIOPinTypeSSI(GPIO_PORTE_BASE, GPIO_PIN_0);
80  // Einschalten des SPI Interfaces
81  SSIEnable(SSI1_BASE);
82
83  //Configure Pins for Chipselect
84  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3); // set PA3 as digital output "SS1_CS_SD"
85  GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1); // set PE1 as digital output "SS1_CS_ADC"
86
87  ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);   //deselect SS1_CS_SD
88  ROM_GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1); //deselect SS1_CS_ADC
89
90
91     // Output
92     UARTprintf("done.\n");
93 }
94
95 /**
```

```
 * Steuerung der SPI Chip Selects fuer ADC und SD
*/
void spiChipSelect(spi_cs_t chip){

 while(SSIBusy(SSI1_BASE)) //Olli BusyWaiting
  ;

 switch (chip) {
  case ADC:
   if (!g_cSsi1Mutex) //Olli MUtex setzen
    g_cSsi1Mutex = 1;
   else
    return; //This case should not happen! -> Because of IRQ priorities this can not be locked!

   GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1); // set PE1 as digital output "SS1_CS_ADC"

   SSIDisable(SSI1_BASE);

   // Den Takt auf 2MHz stellen (hier ist noch Luft nach oben) // geändert von 4Mhz auf 2MHz
   SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3, SSI_MODE_MASTER, 2000000, 8);  // ADC=SSI_FRF_MOTO_MODE_3

   SSIEnable(SSI1_BASE);



   ROM_GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0x00); //select SS1_CS_ADC


   break;


  case SD:

   //SD-Card automaticly sets CS in own sourcefiles!

   break;

  case NONE:

   GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1); //deselect SS1_CS_ADC

   SSIDisable(SSI1_BASE);

   // Den Takt auf 12,5MHz stellen (max)
   SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 12500000, 8);  // ADC=SSI_FRF_MOTO_MODE_3

   SSIEnable(SSI1_BASE);

   if (g_cSsi1Mutex) //Olli MUtex rücksetzen
    g_cSsi1Mutex = 0;
   else
    return;
   break;
 }
}
```

Quellcode E.14: Anhang/c/myuart.c

```
/*
Project:        battery_cycling_sw_v4
File:           myuart.c

Auhtor:         Thomas Wisnewski
Credits:        Matthias Schneider
          Tobias Steinmann
          Fabian Schwartau
          Stellaris Ware

last modified:      2013/05/26

Project Status      Under Construction
Status:         running

CCS:          5.5.1.00031
Stellarisware:      8555

Hardware:         Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:      Source Code for UART0 Initialization for use with
          UARTStdio.h and UARTprintf()

```

```
26 */
27
28
29
30 /*******************************************************************************
31 *
32 * Includings
33 *
34 *******************************************************************************/
35
36 #include "driverlib/uart.h"
37 #include "driverlib/interrupt.h"
38 #include "header/myuart.h"
39 #include "utils/uartstdio.h"
40 #include "driverlib/rom_map.h"
41 #include "inc/hw_ints.h"
42 #include "inc/hw_memmap.h"
43 #include "inc/hw_uart.h"
44 #include "driverlib/rom.h"
45 #include "driverlib/gpio.h"
46 #include "driverlib/pin_map.h"
47 #include "driverlib/sysctl.h"
48 #include "header/mycmdline.h"
49 #include "string.h"
50 #include "header/control.h"
51
52
53 //*****************************************************************************
54 //
55 // Defines the size of the buffer that holds the command line.
56 //
57 //*****************************************************************************
58 #define UART_BUF_SIZE     64
59
60 //*****************************************************************************
61 // Boolean for fully received uart commands
62 //*****************************************************************************
63 tBoolean received = false;
64
65 //*****************************************************************************
66 //
67 // The buffer that holds the command line.
68 //
69 //*****************************************************************************
70 char g_cUartBuf[UART_BUF_SIZE];
71
72 //*****************************************************************************
73 // index for g_cUartBuf[] loop
74 //*****************************************************************************
75 int i = 0;
76
77 //*****************************************************************************
78 //
79 // Initialize UART0
80 //
81 //*****************************************************************************
82 void myUARTinit(void)
83 {
84 // Initialisation for UART2 (RS232/485) The HARDWARE isn't finished yet!!!
85 #ifdef MYUART_SELECT_2
86  //Enable Peripheral
87     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
88     GPIOPinConfigure(GPIO_PG0_U2RX);
89     GPIOPinConfigure(GPIO_PG1_U2TX);
90     ROM_GPIOPinTypeUART(GPIO_PORTG_BASE, GPIO_PIN_0 | GPIO_PIN_1);
91     UARTStdioInit(2);
92
93
94     //Enable Interrupts
95     ROM_UARTEnable(UART2_BASE);
96     ROM_IntEnable(INT_UART2);
97     ROM_UARTIntEnable(UART2_BASE, UART_INT_RX | UART_INT_RT);
98
99
100    UARTprintf("\n\nInitializing UART2...done.\n");
101 #endif
102
103 // Initialisation for UART1 (RS232)     Prio 2
104 #ifdef MYUART_SELECT_1
105  //Enable Peripheral
106     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
107     GPIOPinConfigure(GPIO_PD2_U1RX);
108     GPIOPinConfigure(GPIO_PD3_U1TX);
109     ROM_GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_2 | GPIO_PIN_3);
110     UARTStdioInit(1);
```

```
111
112
113     //Enable Interrupts
114     ROM_UARTEnable(UART1_BASE);
115     ROM_IntEnable(INT_UART1);
116     ROM_UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);
117
118
119     UARTprintf("\n\nInitializing UART1...done.\n");
120 #endif
121     // Initialisation for UART0 (USB)     Prio 1
122 #ifdef MYUART_SELECT_0
123  //Enable Peripheral
124     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
125     GPIOPinConfigure(GPIO_PA0_U0RX);
126     GPIOPinConfigure(GPIO_PA1_U0TX);
127     ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
128     UARTStdioInit(0);
129
130
131     //Enable Interrupts
132     ROM_UARTEnable(UART0_BASE);
133     ROM_IntEnable(INT_UART0);
134     ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
135
136
137     UARTprintf("\n\nInitializing UART0...done.\n");
138 #endif
139
140 }
141
142
143 //*****************************************************************************
144 //
145 // The UART interrupt handler.
146 //
147 //*****************************************************************************
148
149 void UART0IntHandler(void)
150 {
151     unsigned long ulStatus;
152     long c;
153
154     //
155     // Get the interrupt status.
156     //
157     ulStatus = UARTIntStatus(UART0_BASE, true);
158
159     //
160     // Clear the asserted interrupts.
161     //
162     UARTIntClear(UART0_BASE, ulStatus);
163
164     //
165  // Loop while there are characters in the receive FIFO.
166  //
167     while(UARTCharsAvail(UART0_BASE)){
168      c = UARTCharGetNonBlocking(UART0_BASE);
169
170      if(c != -1){
171       if(c == 10){
172        if(g_cUartBuf[i-1] == 13){
173         g_cUartBuf[i - 1] = 0;
174        }
175        received = true;
176        i = 0;
177        break;
178       }
179       else{
180       g_cUartBuf[i] = (char) c;
181       i++;
182       if (i>=UART_BUF_SIZE){      // Check that received data <= UART_BUF_SIZE
183        i=0;
184        g_cUartBuf[0]=0;
185
186        UARTprintf("\n\nBad UART command!\n");
187        telnet_write("\n\nBad UART command!\n");
188        return;
189       }
190
191       }
192      }
193     }
194
195     //
```

```
196     // If command fully received, interpret it and execute. Reset for next command.
197     //
198     if(received){
199       Cmd_interprete(g_cUartBuf);
200   received = false;
201   memset(g_cUartBuf,0,UART_BUF_SIZE);
202     }
203
204 }
205
206 void UART1IntHandler(void)
207 {
208     unsigned long ulStatus;
209     long c;
210
211     //
212     // Get the interrupt status.
213     //
214     ulStatus = UARTIntStatus(UART1_BASE, true);
215
216     //
217     // Clear the asserted interrupts.
218     //
219     UARTIntClear(UART1_BASE, ulStatus);
220
221     //
222  // Loop while there are characters in the receive FIFO.
223  //
224     while(UARTCharsAvail(UART1_BASE)){
225      c = UARTCharGetNonBlocking(UART1_BASE);
226
227      if(c != -1){
228       if(c == 10){
229        if(g_cUartBuf[i-1] == 13){
230         g_cUartBuf[i - 1] = 0;
231        }
232        received = true;
233        i = 0;
234        break;
235       }
236       else{
237       g_cUartBuf[i] = (char) c;
238       i++;
239       if (i>=UART_BUF_SIZE){      // Check that received data <= UART_BUF_SIZE
240        i=0;
241        g_cUartBuf[0]=0;
242
243        UARTprintf("\n\nBad UART command!\n");
244        telnet_write("\n\nBad UART command!\n");
245        return;
246       }
247
248       }
249      }
250     }
251
252     //
253     // If command fully received, interpret it and execute. Reset for next command.
254     //
255     if(received){
256       Cmd_interprete(g_cUartBuf);
257   received = false;
258   memset(g_cUartBuf,0,UART_BUF_SIZE);
259     }
260
261 }
262
263 void UART2IntHandler(void)
264 {
265     unsigned long ulStatus;
266     long c;
267
268     //
269     // Get the interrupt status.
270     //
271     ulStatus = UARTIntStatus(UART2_BASE, true);
272
273     //
274     // Clear the asserted interrupts.
275     //
276     UARTIntClear(UART2_BASE, ulStatus);
277
278     //
279  // Loop while there are characters in the receive FIFO.
280  //
```

```
281     while(UARTCharsAvail(UART2_BASE)){
282      c = UARTCharGetNonBlocking(UART2_BASE);
283
284      if(c != -1){
285       if(c == 10){
286        if(g_cUartBuf[i-1] == 13){
287         g_cUartBuf[i - 1] = 0;
288        }
289        received = true;
290        i = 0;
291        break;
292       }
293       else{
294       g_cUartBuf[i] = (char) c;
295       i++;
296       if (i>=UART_BUF_SIZE){      // Check that received data <= UART_BUF_SIZE
297        i=0;
298        g_cUartBuf[0]=0;
299
300        UARTprintf("\n\nBad UART command!\n");
301        telnet_write("\n\nBad UART command!\n");
302        return;
303       }
304
305       }
306      }
307     }
308
309     //
310     // If command fully received, interpret it and execute. Reset for next command.
311     //
312     if(received){
313      Cmd_interprete(g_cUartBuf);
314   received = false;
315   memset(g_cUartBuf,0,UART_BUF_SIZE);
316     }
317
318 }
```

Quellcode E.15: Anhang/c/onewire.c

```
1  /*
2   * onewire.c
3   *
4   *  Created on: 28.01.2015
5   *      Author: f.groth
6   */
7
8
9  #include "header/onewire.h"
10
11 //*****************************************************************************
12 //
13 // Initialize one-wire bus
14 //
15 //*****************************************************************************
16 void initOneWireUART(void)
17 {
18  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
19     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
20     GPIOPinConfigure(GPIO_PD5_U2RX);
21     GPIOPinConfigure(GPIO_PD6_U2TX);
22     GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_5 | GPIO_PIN_6);
23     UARTEnable(UART2_BASE);
24
25     //Make strong Pull-Up available and enable
26     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
27     GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_4);
28     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
29 }
30
31 //*****************************************************************************
32 //
33 // Reset one-wire devices and detect presence
34 //
35 //*****************************************************************************
36 int initOneWire(char enablePullup)
37 {
38  char presence = 0x00;
39
40  UARTDisable(UART2_BASE);
41     UARTConfigSetExpClk(UART2_BASE, SysCtlClockGet(), 9600,
42                                 (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
```

```
43                                      UART_CONFIG_WLEN_8));
44      UARTEnable(UART2_BASE);
45
46      //Empty UART FIFO - Just to be sure
47      while(UARTCharGetNonBlocking(UART2_BASE) != -1)
48        ;
49
50   //Disable strong pull-up
51   GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
52
53   UARTCharPut(UART2_BASE, 0xF0);
54   presence = UARTCharGet(UART2_BASE);
55   if (presence != 0xF0) {
56    UARTDisable(UART2_BASE);
57    UARTConfigSetExpClk(UART2_BASE, SysCtlClockGet(), 115200,
58            (UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
59             UART_CONFIG_WLEN_8));
60       UARTEnable(UART2_BASE);
61
62       if (enablePullup)
63     //Enable strong pull-up
64     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
65       return 0;
66   } else {
67    if (enablePullup)
68     //Enable strong pull-up
69     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
70    return -1;
71   }
72  }
73
74  //*****************************************************************************
75  //
76  // Write to the one-wire bus
77  //
78  //*****************************************************************************
79  void writeOneWire(char byte, char enablePullup)
80  {
81   int i;
82
83   //Disable strong pull-up
84   GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
85
86   for (i=0; i < 8; i++) {
87    UARTCharPut(UART2_BASE, (0x01<<i & byte) ? 0xFF : 0x00);
88    UARTCharGet(UART2_BASE);
89   }
90
91   if (enablePullup)
92    //Enable strong pull-up
93    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
94  }
95
96  //*****************************************************************************
97  //
98  // Read from one-wire bus
99  //
100 //*****************************************************************************
101 char readOneWire(char enablePullup)
102 {
103  int i;
104  char buffer, byte = 0x00;
105
106  //Disable strong pull-up
107  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
108
109  for (i=0; i < 8; i++) {
110   UARTCharPut(UART2_BASE, 0xFF);
111   buffer = UARTCharGet(UART2_BASE);
112   byte |= (buffer & 0x01)<<i;
113  }
114
115  if (enablePullup)
116   //Enable strong pull-up
117   GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
118
119  return byte;
120 }
121
122 //*****************************************************************************
123 //
124 // Identify devices on the bus
125 //
126 //*****************************************************************************
127 uint8_t searchOneWire(uint64_t* devices, uint8_t maxDevices, uint8_t isAlarm)
```

```
128 {
129  //char i;
130  //char buffer, byte = 0x00;
131  bool lastDevice = false;
132  int idBitNumber = 0;
133  int lastZero = 0;
134  char idBit, compIdBit;
135  int lastDiscrepancy = 0;
136  //int lastFamilyDiscrepancy = 0;
137  char searchDirection;
138  char deviceCount = 0;
139
140     //Empty UART FIFO - Just to be sure
141     while(UARTCharGetNonBlocking(UART2_BASE) != -1)
142       ;
143
144  //Disable strong pull-up
145  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
146
147  for (deviceCount = 0; !lastDevice && (deviceCount < maxDevices); deviceCount++) {
148   if (initOneWire(0)) {
149    //TODO: Goto 2
150    //Enable strong pull-up
151    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
152    return false;
153   }
154
155   idBitNumber = 0;
156   lastZero = 0;
157
158   writeOneWire(isAlarm ? 0xEC : 0xF0, 0);
159
160   devices[deviceCount] = 0x00000000;
161   while (idBitNumber < 64) {
162    //Read ID-Bit and complementary ID-Bit
163    UARTCharPut(UART2_BASE, 0xFF);
164    idBit = UARTCharGet(UART2_BASE) & 0x01;
165    UARTCharPut(UART2_BASE, 0xFF);
166    compIdBit = UARTCharGet(UART2_BASE) & 0x01;
167
168    if (idBit & compIdBit) {
169     lastDiscrepancy = 0;
170     //lastFamilyDiscrepancy = 0;
171     lastDevice = false;
172     //Enable strong pull-up
173     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
174     return 0;
175    }
176
177    if (!(idBit | compIdBit)) {
178     if (idBitNumber == lastDiscrepancy) {
179      searchDirection = 0x01;
180     } else  {
181      if (idBitNumber > lastDiscrepancy) {
182       searchDirection = 0x00;
183       //TODO: This changed today
184       lastZero = idBitNumber;
185      } else
186       if (!(searchDirection = (devices[deviceCount-1] >> idBitNumber) & 0x01))
187        lastZero = idBitNumber;
188     }
189    } else {
190     searchDirection = idBit;
191    }
192
193    devices[deviceCount] |= (uint64_t)searchDirection<<idBitNumber;
194    UARTCharPut(UART2_BASE, searchDirection ? 0xFF : 0x00);
195    UARTCharGet(UART2_BASE);
196    idBitNumber++;
197   }
198
199   lastDiscrepancy = lastZero;
200   if (!lastDiscrepancy)
201    lastDevice = true;
202  }
203
204  //Enable strong pull-up
205  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
206
207  return deviceCount;
208 }
209
210 //*****************************************************************************
211 //
212 // Match one-wire device
```

```
213 //
214 //*****************************************************************************
215 int matchOneWireDevice(uint64_t * addr)
216 {
217  int i;
218
219  //Disable strong pull-up
220  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
221
222  if (initOneWire(0))
223   return -1;
224
225  writeOneWire(0x55, 0);
226
227  for (i=0; i < 8; i++)
228   writeOneWire(((char *)addr)[i], 0);
229
230  //Enable strong pull-up
231  GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
232
233  return 0;
234 }
```

Quellcode E.16: Anhang/c/relais.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          relais.c
4
5  Auhtor:         Thomas Wisnewski
6           Oliver Schmidt
7  Credits:        Stellaris Ware
8
9  last modified:      2015/03/19
10
11 Project Status      Under Construction
12 Status:         running
13
14 CCS:         5.5.1.00031
15 Stellarisware:      8555
16
17 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
18          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
19          NTC-Connectors, MAX3232 and Suplly Circuits
20
21 Description:      Source Code for controlling installed relais
22
23 */
24
25 /*****************************************************************************
26 *
27 * Includings
28 *
29 *****************************************************************************/
30 #include <string.h>
31 #include <stdarg.h>
32 #include <stdio.h>
33 #include <stdlib.h>
34 #include "inc/hw_memmap.h"
35 #include "inc/hw_types.h"
36 #include "utils/uartstdio.h"
37 #include "utils/ustdlib.h"
38 #include "inc/hw_ints.h"
39 #include "inc/hw_types.h"
40 #include "inc/hw_uart.h"
41 #include <inc/hw_ssi.h>
42 #include "inc/hw_gpio.h"
43 #include "driverlib/debug.h"
44 #include "driverlib/interrupt.h"
45 #include "driverlib/rom.h"
46 #include "driverlib/rom_map.h"
47 #include "driverlib/sysctl.h"
48 #include "driverlib/uart.h"
49 #include <driverlib/ssi.h>
50 #include <driverlib/gpio.h>
51 #include <driverlib/sysctl.h>
52
53 /*****************************************************************************
54 *
55 * Own Includings
56 *
57 *****************************************************************************/
58 #include "header/relais.h"
```

```
59 #include "header/control.h"
60 #include "header/config.h"
61
62 extern volatile unsigned int power_relay_active;
63
64 // TCP-Socket für die aktuelle Steuerungs-Verbindung
65 extern struct tcp_pcb* control_connection;
66
67 //*****************************************************************************
68 //
69 // Defines the size of the buffer that holds the input data.
70 //
71 //*****************************************************************************
72 #define RELAIS_INPUT_DATA_SIZE     64
73
74 //*****************************************************************************
75 //
76 // A temporary data buffer used for input data
77 //
78 //*****************************************************************************
79 static char g_cTmpInpData[RELAIS_INPUT_DATA_SIZE];
80
81
82 //*****************************************************************************
83 //
84 // Initialize GPIO ports for relay-mux and powerrelays
85 //
86 //*****************************************************************************
87 void init_relais(void)
88 {
89
90  UARTprintf("Initializing Relais...");
91
92  //Enable Peripheral for Relais-Mux
93  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    // enable peripheral
94  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);    // enable peripheral
95
96  //Enable Peripheral for Relais on power section
97  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);    // enable peripheral
98  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);    // enable peripheral
99
100 // //Unlock PB7 for changing from NMI-Mode
101 //
102 // //
103 // // Unlock access to the commit register
104 // //
105 // HWREG(GPIO_PORTB_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY_DD;
106 //
107 // //
108 // // Set the commit register for PB7 to allow changing the function
109 // //
110 // HWREG(GPIO_PORTB_BASE + GPIO_O_CR) = 0x80;
111 //
112 // //
113 // // Enable the alternate function for PB7 (NMI)
114 // //
115 // HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) |= 0x80;
116 //
117 // //
118 // // Turn on the digital enable for PB7
119 // //
120 // HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= 0x80;
121
122
123  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 ); // set PB2 and PB3 as digital output
124  GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7); // set PH3, PH4 and PH7 as digital output
125  GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7); // set PC5, PC6 and PC7 as digital output
126  GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_7); // set PG7 as digital output
127
128  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3, 0x00); // set PB2 and PB3 to logic "0"
129  GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7, 0x00); // set PH3, PH4 and PH7 to logic "0"
130
131  GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0x00); // set PC5, PC6 and PC7 to logic "0"
132  GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7,  GPIO_PIN_7); // set PG7 to logic "1" "Strobe"
133
134  //Switch off all relays
135  switch_relais(0);
136  switch_power_relais(0);
137
138  UARTprintf("done\n");
139 }
140
141 //*****************************************************************************
142 //
143 // Function for switching relays-mux
```

```
144 //
145 //*****************************************************************************
146 void switch_relais(int choose)
147 {
148
149   GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PH7 (Enable) to logic "1"
150
151
152   switch(choose){
153
154   case 0:
155
156    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
157    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
158    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
159    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
160
161   break;
162
163
164
165   case 1:
166
167    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
168    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
169    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
170    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
171    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
172
173   break;
174
175   case 2:
176
177    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
178    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
179    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
180    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
181    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
182
183   break;
184
185   case 3:
186
187    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
188    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
189    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
190    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
191    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
192
193   break;
194
195   case 4:
196
197    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
198    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
199    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
200    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
201    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
202
203   break;
204
205   case 5:
206
207    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
208    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
209    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
210    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
211    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
212
213   break;
214
215   case 6:
216
217    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
218    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
219    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
220    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
221    GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
222
223   break;
224
225   case 7:
226
227    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
228    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
```

```
229     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
230     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
231     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
232
233    break;
234
235    case 8:
236
237     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
238     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
239     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
240     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
241     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
242
243    break;
244
245    case 9:
246
247     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
248     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
249     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
250     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
251     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
252
253    break;
254
255    case 10:
256
257     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
258     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
259     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
260     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
261     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
262
263    break;
264
265    case 11:
266
267     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
268     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
269     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
270     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
271     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
272
273    break;
274
275    case 12:
276
277     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
278     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
279     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
280     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
281     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic "0"
282
283    break;
284
285    default:
286
287     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
288     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
289     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
290     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
291
292
293    break;
294   }
295
296
297
298
299 }
300
301
302 //*****************************************************************************
303 //
304 // Function for switching powerrelay
305 // To switch one a relay needs 60ms
306 // Relay 1 needs 50ms to switch off
307 //*****************************************************************************
308 int switch_power_relais(int choose)
309 {
310  int active;
311  unsigned long ul_delay_count;
312
313   switch(choose){
```

```
314
315   case 0:
316
317    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
318    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
319    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
320    relay_delay_micros(7142l); //50ms
321    active = 0;
322
323   break;
324
325   case 1:
326    //switch all relays off
327    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
328    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
329    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
330    //wait 55msek
331    //alt relay_delay_micros(7800l);
332    relay_delay_micros(5000l); //35ms
333    //switch the chosen relay
334    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
335    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
336    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
337    active = 1;
338   break;
339
340   case 2:
341
342    //switch all relays off
343    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
344    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
345    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
346    //wait 70ms
347    //alt relay_delay_micros(10000l);
348    relay_delay_micros(7142l); //50ms
349    //switch the chosen relay
350    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
351    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
352    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
353    active = 2;
354
355   break;
356
357   case 3:
358    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
359    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
360    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
361
362    //alt relay_delay_micros(10000l);
363    relay_delay_micros(7142l); //50ms
364
365
366    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
367    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
368    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
369    active = 3;
370   break;
371
372   case 4:
373    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
374    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
375    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
376
377    //alt relay_delay_micros(10000l);
378    relay_delay_micros(7142l); //50ms
379
380    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, GPIO_PIN_5);
381    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
382    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
383    active = 4;
384   break;
385
386   default:
387
388    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
389    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
390    GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
391    relay_delay_micros(7142l); //50ms
392    active = 0;
393   break;
394
395   }
396
397   GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, 0x00); // set PG7 (Strobe) to logic "0"
398   // ca. 10 ms delay
```

```
399   ul_delay_count = 100000;
400   while (ul_delay_count) ul_delay_count--;
401   //-------------------------------
402   GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PG7 (Strobe) to logic "1"
403   // ca. 10 ms delay
404   ul_delay_count = 100000;
405   while (ul_delay_count) ul_delay_count--;
406   //-------------------------------
407
408   if(active)//switch the current active LED
409    GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  GPIO_PIN_7);
410   else
411    GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  0);
412
413   return active;
414 }
415
416 //*****************************************************************************
417 //
418 // This function implements the "Relais" command. It activates ONE chosen power relais
419 //
420 //
421 //*****************************************************************************
422 int
423 Cmd_relais(int argc, char *argv[])
424 {
425  char buf[32];
426
427  if(argc == 2){
428   int relais;
429
430   //
431   // Copy the first input data into buffer.
432   //
433   strcpy(g_cTmpInpData, argv[1]);
434
435   relais = (int) strtol(g_cTmpInpData, (char **)NULL, 10);
436
437   if(relais >= 0 && relais <= 4){
438
439    power_relay_active = switch_power_relais(relais);
440
441    UARTprintf("\nRelais %02d activated", power_relay_active);
442    if(control_connection){
443     sprintf(buf, "Relais %02d activated\n", power_relay_active);
444     telnet_write(buf);
445    }
446
447    //
448    // Return success.
449    //
450    return 0;
451   }
452  }
453  return -1;//Bad command
454 }
455
456
457
458 static void relay_delay_micros(unsigned long microseconds)
459 {
460  //10000 entspricht 70ms gemessen mit Oscilloscope
461  unsigned long counter = 50 * microseconds;
462
463  while(counter--)
464   ;
465 }
```

Quellcode E.17: Anhang/c/rtc.c

```
1  /*
2  Project:       battery_cycling_sw_v4
3  File:          rtc.c
4
5  Auhtor:        Thomas Wisnewski
6                 Johannes Roehn
7  Credits:       Stellaris Ware
8
9  last modified:      2013/12/11
10
11 Project Status      Under Construction
12 Status:        running
13
```

```
14 CCS:          5.5.1.00031
15 Stellarisware:       8555
16
17 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
18          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
19          NTC-Connectors, MAX3232 and Suplly Circuits
20
21 Description:       Source Code extern RTC-Clock
22
23 */
24
25 /*****************************************************************************
26 *
27 * Includings
28 *
29 *****************************************************************************/
30 #include "inc/hw_types.h"
31 #include "driverlib/sysctl.h"
32 #include "inc/hw_i2c.h"
33 #include "driverlib/i2c.h"
34 #include "inc/hw_memmap.h"
35 #include "driverlib/gpio.h"
36 #include "driverlib/rom.h"
37 #include "utils/uartstdio.h"
38 #include "header/rtc.h"
39
40
41 //*****************************************************************************
42 //Extern variables for Clock
43 //*****************************************************************************
44 extern volatile unsigned long clock_msec;
45 extern volatile unsigned long clock_sec;
46 extern volatile unsigned long clock_min;
47 extern volatile unsigned long clock_hour;
48 extern volatile unsigned long clock_day;
49 extern volatile unsigned long clock_month;
50 extern volatile unsigned long clock_year;
51
52 //*****************************************************************************
53 //
54 // Initialize RTC-Clock and setup systemclok by values from RTC
55 //
56 //*****************************************************************************
57 void rtc_init(void){
58
59
60  UARTprintf("Initializing RTC...");
61
62  //Enable peripherals
63  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
64  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
65
66  GPIOPinConfigure(GPIO_PA6_I2C1SCL);
67  GPIOPinConfigure(GPIO_PA7_I2C1SDA);
68     ROM_GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7);
69
70  ROM_I2CMasterInitExpClk(I2C1_MASTER_BASE,SysCtlClockGet(),false);      //InitializeMasterandSlave
71
72
73  set_clock_values();//Read clock values from RTC
74
75
76  UARTprintf("done\n");
77
78 }
79
80
81 //
82 // Function for setting clock values from rtc. Called once while Initializing on startup.
83 //
84 void set_clock_values(void){
85
86
87  unsigned char adc_dat;
88
89
90  //Set year
91  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
92
93  I2CMasterDataPut(I2C1_MASTER_BASE,0x06); // location address
94  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
95  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
96
97  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
98  while(I2CMasterBusy(I2C1_MASTER_BASE));
```

```
99
100 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
101 while(I2CMasterBusy(I2C1_MASTER_BASE));
102
103 adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
104 while(I2CMasterBusy(I2C1_MASTER_BASE));
105
106 clock_year = ((adc_dat >> 4) & 0x0F) * 10 + (adc_dat & 0x0F) + 2000;
107
108
109 //Set month
110 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
111
112 I2CMasterDataPut(I2C1_MASTER_BASE,0x05); // location address
113 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
114 while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
115
116 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
117 while(I2CMasterBusy(I2C1_MASTER_BASE));
118
119 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
120 while(I2CMasterBusy(I2C1_MASTER_BASE));
121
122 adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
123 while(I2CMasterBusy(I2C1_MASTER_BASE));
124
125 clock_month = ((adc_dat >> 4) & 0x01) * 10 + (adc_dat & 0x0F);
126
127 //Set date
128 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
129
130 I2CMasterDataPut(I2C1_MASTER_BASE,0x04); // location address
131 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
132 while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
133
134 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
135 while(I2CMasterBusy(I2C1_MASTER_BASE));
136
137 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
138 while(I2CMasterBusy(I2C1_MASTER_BASE));
139
140 adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
141 while(I2CMasterBusy(I2C1_MASTER_BASE));
142
143 clock_day = ((adc_dat >> 4) & 0x03) * 10 + (adc_dat & 0x0f);
144
145 //Set hour
146 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
147
148 I2CMasterDataPut(I2C1_MASTER_BASE,0x02); // location address
149 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
150 while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
151
152 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
153 while(I2CMasterBusy(I2C1_MASTER_BASE));
154
155 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
156 while(I2CMasterBusy(I2C1_MASTER_BASE));
157
158 adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
159 while(I2CMasterBusy(I2C1_MASTER_BASE));
160
161 clock_hour = ((adc_dat >> 4) & 0x03) * 10 + (adc_dat & 0x0f);
162
163 //Set minute
164 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
165
166 I2CMasterDataPut(I2C1_MASTER_BASE,0x01); // location address
167 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
168 while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
169
170 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
171 while(I2CMasterBusy(I2C1_MASTER_BASE));
172
173 I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
174 while(I2CMasterBusy(I2C1_MASTER_BASE));
175
176 adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
177 while(I2CMasterBusy(I2C1_MASTER_BASE));
178
179 clock_min = ((adc_dat >> 4) & 0x07) * 10 + (adc_dat & 0x0f);
180
181 //Set second
182 I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
183
```

```
184  I2CMasterDataPut(I2C1_MASTER_BASE,0x00); // location address
185  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
186  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
187
188  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
189  while(I2CMasterBusy(I2C1_MASTER_BASE));
190
191  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
192  while(I2CMasterBusy(I2C1_MASTER_BASE));
193
194  adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
195  while(I2CMasterBusy(I2C1_MASTER_BASE));
196
197  clock_sec = ((adc_dat >> 4) & 0x07) * 10 + (adc_dat & 0x0F);
198
199 }
200
201 //
202 // Function for setting values in rtc.
203 //
204 void set_rtc_values(unsigned int year, unsigned int month, unsigned int day, unsigned int hour, unsigned int min, unsigned int sec ){
205
206  unsigned char val;
207
208  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
209
210  I2CMasterDataPut(I2C1_MASTER_BASE,0x06); // location address
211  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
212  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
213
214  val = ( (((unsigned char) (((year - (year % 10))/10) << 4)) & 0xF0) | (((unsigned char) (year % 10))) & 0x0F ); //BCD representation
         of the value to be written
215  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
216  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
217  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
218
219  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
220
221  I2CMasterDataPut(I2C1_MASTER_BASE,0x05); // location address
222  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
223  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
224
225  val = ( (((unsigned char) (((month - (month % 10))/10) << 4)) & 0x10) | (((unsigned char) (month % 10))) & 0x0F ); //BCD
         representation of the value to be written
226  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
227  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
228  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
229
230  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
231
232  I2CMasterDataPut(I2C1_MASTER_BASE,0x04); // location address
233  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
234  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
235
236  val = ( (((unsigned char) (((day - (day % 10))/10) << 4)) & 0x30) | (((unsigned char) (day % 10))) & 0x0F ); //BCD representation of
         the value to be written
237  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
238  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
239  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
240
241  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
242
243  I2CMasterDataPut(I2C1_MASTER_BASE,0x02); // location address
244  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
245  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
246
247  val = ( (((unsigned char) (((hour - (hour % 10))/10) << 4)) & 0x30) | (((unsigned char) (hour % 10))) & 0x0F ); //BCD representation
         of the value to be written
248  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
249  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
250  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
251
252  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
253
254  I2CMasterDataPut(I2C1_MASTER_BASE,0x01); // location address
255  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
256  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
257
258  val = ( (((unsigned char) (((min - (min % 10))/10) << 4)) & 0x70) | (((unsigned char) (min % 10))) & 0x0F ); //BCD representation of
         the value to be written
259  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
260  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
261  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
262
263  I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
```

```
264
265  I2CMasterDataPut(I2C1_MASTER_BASE,0x00); // location address
266  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
267  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
268
269  val = ( (((unsigned char) (((sec - (sec % 10))/10) << 4)) & 0x70) | (((unsigned char) (sec % 10))) & 0x0F ); //BCD representation of
         the value to be written
270  I2CMasterDataPut(I2C1_MASTER_BASE, val); // location address
271  I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
272  while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
273
274 }
```

Quellcode E.18: Anhang/c/startup_ccs.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          startup_ccs.c
4
5  Auhtor:         Thomas Wisnewski
6  Credits:        Stellaris Ware
7
8  last modified:       2013/02/05
9
10 Project Status       Under Construction
11 tatus:         running
12
13 CCS:         5.5.1.00031
14 Stellarisware:       8555
15
16 Hardware:         Stellaris EKS-LM3S9B92 on Extension Board with
17          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
18          NTC-Connectors, MAX3232 and Suplly Circuits
19
20 Description:       Startup Configuration for CCS, mainly defining the ISRs
21          origining from stellaris ware at:
22          boards/ek-lm3s9b92/uart_echo
23 */
24 #include "inc/hw_memmap.h"
25 #include "inc/hw_types.h"
26 #include "driverlib/gpio.h"
27 #include "header/mycmdline.h"
28 #include "header/clocktimer.h"
29
30 //*****************************************************************************
31 //
32 // startup_ccs.c - Startup code for use with TI's Code Composer Studio.
33 //
34 // Copyright (c) 2009-2012 Texas Instruments Incorporated.  All rights reserved.
35 // Software License Agreement
36 //
37 // Texas Instruments (TI) is supplying this software for use solely and
38 // exclusively on TI's microcontroller products. The software is owned by
39 // TI and/or its suppliers, and is protected under applicable copyright
40 // laws. You may not combine this software with "viral" open-source
41 // software in order to form a larger program.
42 //
43 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
44 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
45 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
46 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
47 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
48 // DAMAGES, FOR ANY REASON WHATSOEVER.
49 //
50 // This is part of revision 8555 of the EK-LM3S9B92 Firmware Package.
51 //
52 //*****************************************************************************
53
54 //*****************************************************************************
55 //
56 // Forward declaration of the default fault handlers.
57 //
58 //*****************************************************************************
59 void ResetISR(void);
60 static void NmiSR(void);
61 static void FaultISR(void);
62 static void IntDefaultHandler(void);
63 extern void UARTStdioIntHandler(void);
64 extern void SysTickHandler(void);
65
66
67 //*****************************************************************************
68 //
```

```
69  // External declaration for the reset handler that is to be called when the
70  // processor is started
71  //
72  //*****************************************************************************
73  extern void _c_int00(void);
74
75  //*****************************************************************************
76  //
77  // Linker variable that marks the top of the stack.
78  //
79  //*****************************************************************************
80  extern unsigned long __STACK_TOP;
81
82  //*****************************************************************************
83  //
84  // External declaration for the interrupt handler used by the application.
85  //
86  //*****************************************************************************
87  extern void UARTIntHandler(void);
88  extern void Timer0IntHandler(void);
89  extern void Timer1IntHandler(void);
90  extern void Timer2IntHandler(void);
91  extern void Timer3IntHandler(void);
92  extern void ADC0Handler(void);
93  extern void ADC1Handler(void);
94  extern void lwIPEthernetIntHandler(void);
95  extern void UART0IntHandler(void);
96  extern void UART1IntHandler(void);
97  extern void UART2IntHandler(void);
98
99  extern void GPIOD01Handler(void);
100 extern void GPIOF00Handler(void);
101 extern void GPIOC04Handler(void);
102 extern void GPIOH05Handler(void);
103 extern void watchdog_int_handler(void);
104
105 //*****************************************************************************
106 //
107 // The vector table.  Note that the proper constructs must be placed on this to
108 // ensure that it ends up at physical address 0x0000.0000 or at the start of
109 // the program if located at a start address other than 0.
110 //
111 //*****************************************************************************
112 #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
113 void (* const g_pfnVectors[])(void) =
114 {
115     (void (*)(void))((unsigned long)&__STACK_TOP),
116                                             // The initial stack pointer
117     ResetISR,                               // The reset handler
118     NmiSR,                                  // The NMI handler
119     FaultISR,                               // The hard fault handler
120     IntDefaultHandler,                      // The MPU fault handler
121     IntDefaultHandler,                      // The bus fault handler
122     IntDefaultHandler,                      // The usage fault handler
123     0,                                      // Reserved
124     0,                                      // Reserved
125     0,                                      // Reserved
126     0,                                      // Reserved
127     IntDefaultHandler,                      // SVCall handler
128     IntDefaultHandler,                      // Debug monitor handler
129     0,                                      // Reserved
130     IntDefaultHandler,                      // The PendSV handler
131     SysTickHandler,                        // The SysTick handler
132     IntDefaultHandler,                      // GPIO Port A
133     IntDefaultHandler,                      // GPIO Port B
134     GPIOC04Handler,                        // GPIO Port C
135     GPIOD01Handler,                         // GPIO Port D
136     IntDefaultHandler,                      // GPIO Port E
137     UART0IntHandler,                       // UART0 Rx and Tx
138  //UARTStdioIntHandler,                     // UART0 Rx and Tx
139     UART1IntHandler,                        // UART1 Rx and Tx
140     //IntDefaultHandler,                        // UART1 Rx and Tx
141     IntDefaultHandler,                      // SSI0 Rx and Tx
142     IntDefaultHandler,                      // I2C0 Master and Slave
143     IntDefaultHandler,                      // PWM Fault
144     IntDefaultHandler,                      // PWM Generator 0
145     IntDefaultHandler,                      // PWM Generator 1
146     IntDefaultHandler,                      // PWM Generator 2
147     IntDefaultHandler,                      // Quadrature Encoder 0
148     ADC0Handler,                       // ADC Sequence 0
149     IntDefaultHandler,                      // ADC Sequence 1
150     IntDefaultHandler,                      // ADC Sequence 2
151     IntDefaultHandler,                      // ADC Sequence 3
152     //IntDefaultHandler,                        // Watchdog timer
153     watchdog_int_handler,       // Watchdog timer
```

```
154     Timer0IntHandler,                       // Timer 0 subtimer A
155     IntDefaultHandler,                      // Timer 0 subtimer B
156     Timer1IntHandler,                       // Timer 1 subtimer A
157     IntDefaultHandler,                      // Timer 1 subtimer B
158     Timer2IntHandler,                      // Timer 2 subtimer A
159     IntDefaultHandler,                      // Timer 2 subtimer B
160     IntDefaultHandler,                      // Analog Comparator 0
161     IntDefaultHandler,                      // Analog Comparator 1
162     IntDefaultHandler,                      // Analog Comparator 2
163     IntDefaultHandler,                      // System Control (PLL, OSC, BO)
164     IntDefaultHandler,                      // FLASH Control
165     GPIOF00Handler,                     // GPIO Port F
166     IntDefaultHandler,                      // GPIO Port G
167     GPIOH05Handler,                      // GPIO Port H
168     UART2IntHandler,                     // UART2 Rx and Tx
169     IntDefaultHandler,                      // SSI1 Rx and Tx
170     Timer3IntHandler,                     // Timer 3 subtimer A
171     IntDefaultHandler,                      // Timer 3 subtimer B
172     IntDefaultHandler,                      // I2C1 Master and Slave
173     IntDefaultHandler,                      // Quadrature Encoder 1
174     IntDefaultHandler,                      // CAN0
175     IntDefaultHandler,                      // CAN1
176     IntDefaultHandler,                      // CAN2
177     lwIPEthernetIntHandler,                 // Ethernet
178     IntDefaultHandler,                      // Hibernate
179     IntDefaultHandler,                      // USB0
180     IntDefaultHandler,                      // PWM Generator 3
181     IntDefaultHandler,                      // uDMA Software Transfer
182     IntDefaultHandler,                      // uDMA Error
183     IntDefaultHandler,                      // ADC1 Sequence 0
184     ADC1Handler,                        // ADC1 Sequence 1
185     IntDefaultHandler,                      // ADC1 Sequence 2
186     IntDefaultHandler,                      // ADC1 Sequence 3
187     IntDefaultHandler,                      // I2S0
188     IntDefaultHandler,                      // External Bus Interface 0
189     IntDefaultHandler                       // GPIO Port J
190 };
191
192 //*****************************************************************************
193 //
194 // This is the code that gets called when the processor first starts execution
195 // following a reset event.  Only the absolutely necessary set is performed,
196 // after which the application supplied entry() routine is called.  Any fancy
197 // actions (such as making decisions based on the reset cause register, and
198 // resetting the bits in that register) are left solely in the hands of the
199 // application.
200 //
201 //*****************************************************************************
202 void
203 ResetISR(void)
204 {
205     //
206     // Jump to the CCS C initialization routine.
207     //
208     __asm("    .global _c_int00\n"
209           "    b.w     _c_int00");
210 }
211
212 //*****************************************************************************
213 //
214 // This is the code that gets called when the processor receives a NMI.  This
215 // simply enters an infinite loop, preserving the system state for examination
216 // by a debugger.
217 //
218 //*****************************************************************************
219 static void
220 NmiSR(void)
221 {
222     //
223     // Enter an infinite loop.
224     //
225     while(1)
226     {
227     }
228 }
229
230 //*****************************************************************************
231 //
232 // This is the code that gets called when the processor receives a fault
233 // interrupt.  This simply enters an infinite loop, preserving the system state
234 // for examination by a debugger.
235 //
236 //*****************************************************************************
237 static void
238 FaultISR(void)
```

```
{
 unsigned long ul_delay_count;
 GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);

 Cmd_stop(0,0);
    //
    // Enter an infinite loop.
    //
    while(1)
    {

     GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_6, ~GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_6));

  // ca. 0,5 s delay
  ul_delay_count = 5000000;
  while (ul_delay_count) ul_delay_count--;
  //-------------------------------------

    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

Quellcode E.19: Anhang/c/temperature.c

```
/*
Project:        battery_cycling_sw_v4
File:          main.c

Auhtor:         Thomas Wisniewski
Credits:        Matthias Schneider
          Tobias Steinmann
          Fabian Schwartau
          Oliver Schmdit
          Stellaris Ware

last modified:      2015/02/23

Project Status      Under Construction
Status:         running

CCS:        5.5.1.00031
Stellarisware:      8555

Hardware:       Stellaris EKS-LM3S9B92 on Extension Board with
          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
          NTC-Connectors, MAX3232 and Suplly Circuits

Description:      Code for initializing intern ADCs for temperature measurement
*/

/*****************************************************************************
*
* Includings
*
*****************************************************************************/
#include "driverlib/rom.h"
#include "third_party/fatfs/src/ff.h"
#include "third_party/fatfs/src/diskio.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
```

```
43 #include "driverlib/rom.h"
44 #include "driverlib/sysctl.h"
45 #include "driverlib/timer.h"
46 #include <stdio.h>
47 #include "utils/uartstdio.h"
48 #include "math.h"
49 #include "header/onewire.h"
50
51
52 // A global buffer containing the addresses of all connected onewire devices
53 uint64_t g_lOneWireSlaves[13] = {0}; //Maximale Anzahl
54 unsigned char g_cOneWireSlaveCount = 0;
55 //Currently active sensor
56 unsigned char g_cOneWireActiveSlave = 0;
57
58 //*****************************************************************************
59 //
60 // A global data buffer used for actual Cell temperature
61 //
62 //*****************************************************************************
63 volatile int g_iCellTemperature[12] = {0};
64
65 //Buffer for ADC-Samples
66 unsigned long g_pulData0[8];
67 unsigned long g_pulData1[4];
68
69 /*****************************************************************************
70 *
71 * Initialize internal ADCs for Temperature sensors
72 *
73 *****************************************************************************/
74 void init_temperature() {
75
76  UARTprintf("Initializing OneWire...");
77
78  initOneWireUART();
79  g_cOneWireSlaveCount = searchOneWire(g_lOneWireSlaves, 13, 0); //Get all, but at most 13, onewire slave addresses independent of their
         alarm status
80
81  if(g_cOneWireSlaveCount==0){
82   UARTprintf("Failer: Initializing OneWire again...");
83   g_cOneWireSlaveCount = searchOneWire(g_lOneWireSlaves, 13, 0);
84  }
85  //enable Peripheral
86  /*ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
87  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
88
89  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
90  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
91  ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
92
93  ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
94  ROM_GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
95  ROM_GPIOPinTypeADC(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
96
97  // Set Speed to 500.000 Samples per Second (up to 1MSPS)
98     ROM_SysCtlADCSpeedSet(SYSCTL_ADCSPEED_125KSPS);
99
100    //
101    // Configure the ADCs.
102    //
103    ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
104    ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH0);
105    ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH1);
106    ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH2);
107    ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH3);
108    ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH8);
109    ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ADC_CTL_CH9);
110    ADCSequenceStepConfigure(ADC0_BASE, 0, 6, ADC_CTL_CH11);
111    ADCSequenceStepConfigure(ADC0_BASE, 0, 7, ADC_CTL_CH10 | ADC_CTL_IE | ADC_CTL_END);
112
113    ADCSequenceConfigure(ADC1_BASE, 1, ADC_TRIGGER_TIMER, 1);
114    ADCSequenceStepConfigure(ADC1_BASE, 1, 0, ADC_CTL_CH4);
115    ADCSequenceStepConfigure(ADC1_BASE, 1, 1, ADC_CTL_CH5);
116    ADCSequenceStepConfigure(ADC1_BASE, 1, 2, ADC_CTL_CH6);
117    ADCSequenceStepConfigure(ADC1_BASE, 1, 3, ADC_CTL_CH7 | ADC_CTL_IE | ADC_CTL_END);
118
119 ADCSequenceEnable(ADC0_BASE, 0);
120 ADCSequenceEnable(ADC1_BASE, 1);
121
122    //
123    // Clear out the FIFO (not really important for this exercise)
124    //
125    ROM_ADCSequenceDataGet(ADC0_BASE, 0, g_pulData0);
126    ROM_ADCSequenceDataGet(ADC1_BASE, 1, g_pulData1);
```

```
127
128     //
129     // Clear the interrupt status (again, not too important)
130     //
131     ROM_ADCIntClear(ADC0_BASE, 0);
132     ROM_ADCIntClear(ADC1_BASE, 1);
133
134     //
135     // Allow the ADC sequencer in both ADCs to generate interrupts
136     // for SS0
137     //
138     ROM_ADCIntEnable(ADC0_BASE, 0);
139     ROM_ADCIntEnable(ADC1_BASE, 1);
140
141     //
142     // Get the NVIC to generate the appropriate interrupts for the
143     // interrupt handlers as spec'd in the vector table (see startup_*.c)
144     //
145     ROM_IntEnable(INT_ADC0SS0);
146     ROM_IntEnable(INT_ADC1SS1);*/
147
148  UARTprintf("done, found %d devices\n", g_cOneWireSlaveCount);
149
150 }
151
152
153 /*****************************************************************************
154 *
155 * Get the temperature from global buffer
156 *
157 *****************************************************************************/
158 int get_temperature(int choose)
159 {
160 /* int temperature;
161  unsigned int adc_value;
162
163  if((choose >= 0) && (choose <= 8)){
164   adc_value = g_pulData0[choose - 1];
165
166   if(adc_value<=2){ //Erkennung ob Sensor angeschlossen ist. 2==>Schaltschwelle fürs rauschen um 0
167    temperature = 0;
168   }
169   else
170    temperature =(int)((3988 * 298.15 / ( 3988 + log(1024.0 / adc_value - 1) * 298.15) - 273.15) * 10 + 0.5);  // *10 für Temp ohne
           Komma, +0,5 fürs runden
171  }
172  g_iCellTemperature[choose-1] = temperature-50; //Manuelle Offset Korrektur -5°C
173
174  return temperature;*/
175
176 //Alte Temp. Funktion Problem: Teilt durch 0! Wenn kein Temp Sensor angeschlossen ist!
177 //  unsigned int temperature;
178 //
179 //  if((choose >= 0) && (choose <= 8)){
180 //   temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData0[choose - 1] / 10000 - 1 ) * 298) - 273) * 10;
181 //  }
182 //
183 //  if((choose >= 9) && (choose <= 12)){
184 //   temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData1[choose - 9] / 10000 - 1 ) * 298) - 273) * 10;
185 //  }
186 //
187 // // temperature=0; //olli
188 //
189 //  if(temperature >= 2147483647){ //2^31 -1 für int Temp = 0°
190 //   g_iCellTemperature[choose-1] = 0;
191 //   temperature = 0;
192 //  }else
193 //   g_iCellTemperature[choose-1] = temperature;
194 //
195 //  return temperature;
196
197  int i,temp;
198  char measurement[2] = {0};
199  if (g_cOneWireSlaveCount) {
200   if (matchOneWireDevice(&(g_lOneWireSlaves[g_cOneWireActiveSlave])) == -1) { //TODO: What do we do when no sensor answers
201    g_iCellTemperature[g_cOneWireActiveSlave] = -560;
202   } else {
203    writeOneWire(0xBE, 1); //Reads the scratchpad from the sensor
204
205    // read 8 Bytes + 1 CRC Byte
206    for (i=0; i<2; i++)
207     measurement[i] = readOneWire(1);
208
209    if (((uint16_t*)measurement)[0] == 0xFFFF)  {
210     g_iCellTemperature[g_cOneWireActiveSlave] = -560;
```

```
211     } else {
212      for (i=0; i<7; i++)
213       readOneWire(1);
214
215      temp = *((short *)measurement) * 0.625; //Results in an int giving the temperature * 10
216
217      if(temp==850)   //Olli  85°C unschön passiert max Value
218       UARTprintf("Temp Messung ausgelassen temp: %d >= 850\n",temp);
219      else
220       g_iCellTemperature[g_cOneWireActiveSlave] = temp;
221     }
222    //if (g_iCellTemperature[g_cOneWireActiveSlave - 1] == 850) //Discard ROM default value for the temperature
223    // g_iCellTemperature[g_cOneWireActiveSlave - 1] = 0;
224    }
225  }
226
227  //OLLI matchOneWireDevice(&(g_lOneWireSlaves[++g_cOneWireActiveSlave > g_cOneWireSlaveCount ? 0 : (g_cOneWireActiveSlave-1)])); //
         Addresses the next sensor in a circular manner
228
229  if(g_cOneWireActiveSlave == g_cOneWireSlaveCount-1){
230   g_cOneWireActiveSlave = 0;
231  }
232  else{
233   g_cOneWireActiveSlave++;
234  }
235
236  matchOneWireDevice(&(g_lOneWireSlaves[g_cOneWireActiveSlave]));
237
238  writeOneWire(0x44, 1); //Trigger measurement
239
240  if (choose)
241   return g_iCellTemperature[choose - 1];
242  else
243   return 0;
244 }
245
246
247 //
248 // The ADC unit 0 interrupt hander
249 //
250 void
251 ADC0Handler(void)
252 {
253     ROM_ADCIntClear(ADC0_BASE, 0);
254     //write adc-values to global buffer
255     ROM_ADCSequenceDataGet(ADC0_BASE, 0, g_pulData0);
256 }
257
258
259 //
260 // The ADC unit 1 interrupt handler
261 //
262 void
263 ADC1Handler(void)
264 {
265     ROM_ADCIntClear(ADC1_BASE, 1);
266     //write adc-values to global buffer
267     ROM_ADCSequenceDataGet(ADC1_BASE, 1, g_pulData1);
268 }
```

Quellcode E.20: Anhang/c/watchdog.c

```
1  /*
2  Project:        battery_cycling_sw_v4
3  File:          watchdog.c
4
5  Auhtor:         Johannes Roehn
6
7
8  last modified:       2013/12/17
9
10 Project Status       Under Construction
11 Status:         running
12
13 CCS:         5.5.1.00031
14 Stellarisware:       8555
15
16 Hardware:        Stellaris EKS-LM3S9B92 on Extension Board with
17          16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
18          NTC-Connectors, MAX3232 and Suplly Circuits
19
20 Description:       Code for initializing watchdog and interrupt handler
21 */
```

```
22
23 /*****************************************************************************
24 *
25 * Own Includings
26 *
27 *****************************************************************************/
28 #include "header/watchdog.h"
29 #include "driverlib/rom.h"
30 #include "driverlib/sysctl.h"
31 #include "inc/hw_memmap.h"
32 #include "inc/hw_ints.h"
33 #include "header/relais.h"
34 #include "utils/uartstdio.h"
35 #include "header/mysdcard.h"
36
37 //*****************************************************************************
38 //Extern variable for name of file for measurment saving
39 //*****************************************************************************
40 extern char MEAS_FILE[128];
41
42 extern volatile unsigned int measurement_active;
43
44
45 extern volatile unsigned long clock_msec;
46 extern volatile unsigned long clock_sec;
47 extern volatile unsigned long clock_min;
48 extern volatile unsigned long clock_hour;
49 extern volatile unsigned long clock_day;
50 extern volatile unsigned long clock_month;
51 extern volatile unsigned long clock_year;
52
53
54 int feedCount;
55
56 //*****************************************************************************
57 //
58 // Flag to tell the watchdog interrupt handler whether or not to clear the
59 // interrupt (feed the watchdog).
60 //
61 //*****************************************************************************
62 volatile tBoolean g_bFeedWatchdog = true;
63
64
65
66 void watchdog_init(void){
67
68     /* Enable the peripherals used by this example. */
69     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);
70
71     //
72  // Enable the watchdog interrupt.
73  //
74  ROM_IntEnable(INT_WATCHDOG);
75
76     /* Set the period of the watchdog timer. */
77     ROM_WatchdogReloadSet(WATCHDOG0_BASE, SysCtlClockGet() * 10);//called every 5 seconds
78
79     /* Enable reset generation from the watchdog timer. */
80     ROM_WatchdogResetEnable(WATCHDOG0_BASE);
81
82     /* Enable the watchdog timer. */
83     ROM_WatchdogEnable(WATCHDOG0_BASE);
84
85     ROM_WatchdogLock(WATCHDOG0_BASE);//Lock the watchdog registers
86
87     ROM_WatchdogStallEnable(WATCHDOG0_BASE);
88 }
89
90
91 void watchdog_int_handler(void){
92
93  if(g_bFeedWatchdog){//feed the watchdog if variable is set in main loop
94   g_bFeedWatchdog = false;
95
96       UARTStdioInit(1);
97       feedCount++;
98       UARTprintf("%06d,%04d,%02d,%02d,%02d,%02d,%02d,%03d ",feedCount, clock_year, clock_month, clock_day, clock_hour, clock_min,
             clock_sec, clock_msec);
99       UARTprintf(" Watchdog feed\n");
100      UARTStdioInit(0);
101
102
103
104      ROM_WatchdogIntClear(WATCHDOG0_BASE);
105  }else{
```

```
106   //system will  be restarted, switch off all relays
107   switch_power_relais(0);
108   switch_relais(0);
109   UARTprintf("WATCHDOG FAIL!\nSYSTEM RESET!\n");
110   if(measurement_active)
111    add_to_file(MEAS_FILE, "\nWATCHDOG FAIL!\nSYSTEM RESET!\n");
112  }
113 }
```

Quellcode E.21: Anhang/h/clocktimer.h

```
1  /*
2   * clocktimer.h
3   *
4   *  Created on: 04.03.2013
5   *      Author: Thomas W., Johannes R.
6   *      modified 13.01.2014
7   */
8
9  #ifndef TIMER_H_
10 #define TIMER_H_
11
12 /*****************************************************************************
13 *
14 * Function Declarations
15 *
16 *****************************************************************************/
17 void init_clock_timer(void);
18 void TimerClock(void);
19 void reinit_timer0(void);
20 void stop_cycling(void);
21
22 int Cmd_start(int argc, char *argv[]);
23 int Cmd_stop(int argc, char *argv[]);
24
25 // LOOPDEPTH for the relsel loop
26 #define LOOPDEPTH 5
27
28 #endif /* TIMER_H_ */
```

Quellcode E.22: Anhang/h/config.h

```
1  /*
2   * config.h
3   *
4   *  Created on: 11.12.2013
5   *      Author: Thomas W.
6   *        Johannes R.
7   *      modified 13.01.2014
8   */
9
10
11 #ifndef CONFIG_H_
12 #define CONFIG_H_
13
14 #include "config.h"
15
16
17 // defines, which ADC will be used
18
19 #define AD7798  0
20 #define AD7799  1
21
22 #define ADC_TYPE AD7799
23
24 #ifndef ADC_TYPE
25 # define ADC_TYPE AD7798
26 #endif
27
28
29
30 //defines for all the available step types
31 //#define WAIT  1
32 //#define CHARGE_CELLV 2
33 //#define DISCHARGE_CELLV 3
34 //#define CHARGE_BATV 4
35 //#define DISCHARGE_BATV 5
36
37 typedef
38 enum steptypes{
39  WAIT,
```

```
40  CHARGE_CELLV,
41  DISCHARGE_CELLV,
42  CHARGE_BATV,
43  DISCHARGE_BATV,
44  LOOP_START,
45  LOOP_END
46 }steptypes_t;
47
48 #define MAX_STEP_TYPES 7
49
50
51 //define for config init
52 #define MAX_CYCLE_STEPS 250
53 //*****************************************************************************
54 //
55 // Defines the size of the buffer that holds the input data.
56 //
57 //*****************************************************************************
58 #define CONF_BUF_SIZE 6144
59 //#define CONF_BUF_SIZE 21000
60
61
62 //defines for calibration
63 #define CALIBRATION_THRESHOLD 1000L // Schaltschwelle liegt bei ca. 0,2V da: 5/2^16*1000 * 2 =0,152V +einen gewissen Offset, durch die
       Mittelwert bildung.
64        // Der Faktor 2 wird in der Auswertung wieder raus gerechnet
65 #define NO_MEASUREMENTS  16
66 //#define NO_MEASUREMENTS_EXP 4 // n=4 entspricht 2^n = 16 Messungen
67 #define MAX_NO_CHANNELS 12
68 #define CALIBRATION_STEPS 20
69 //scruct defining a single cycle step
70 typedef struct
71 {
72  int type;
73  int param;
74  int relay;
75
76 } cycle_step_t;
77
78 // sizeof(config_t) must be an even number! Otherwise the program
79 // will fail to save and/or reload the configuration!
80 typedef struct
81 {
82  long quantity_cells;
83  long cyclestep;
84  float A[13];
85  float B[13];
86  float C[13];
87  int ip_a;
88  int ip_b;
89  int ip_c;
90  int ip_d;
91  int zerocurrentch1;
92  int zerocurrentch2;
93
94  int measuretime;
95  int max_cycle_time;
96
97  int step_quantity;
98  //includes all variables for cylcing
99  cycle_step_t cycle_steps[MAX_CYCLE_STEPS+1];
100
101  //LIMIT values
102  int max_current;
103  int min_voltage;
104  int max_voltage;
105  int max_temp;
106  int max_charge;
107
108 } config_t;
109
110
111 extern config_t config;
112
113 /*****************************************************************************
114 *
115 * Function Declarations
116 *
117 *****************************************************************************/
118 void config_read(void);
119 int config_write(void);
120 void init_config(void);
121 void calibrate_adc_quadratisch(void);
122
123 int config_cmd_line(int argc, char *argv[]);
```

```
124 int Cmd_config_exit(int argc, char *argv[]);
125 int Cmd_set_coeff   (int argc, char *argv[]);
126 int Cmd_calibrate   (int argc, char *argv[]);
127 int Cmd_calVolt     (int argc, char *argv[]);
128 int Cmd_print       (int argc, char *argv[]);
129 int Cmd_quantity    (int argc, char *argv[]);
130 int Cmd_cyclestep   (int argc, char *argv[]);
131 int Cmd_relsel      (int argc, char *argv[]);
132 int Cmd_relqua      (int argc, char *argv[]);
133 int Cmd_meastime    (int argc, char *argv[]);
134 int Cmd_save        (int argc, char *argv[]);
135 int Cmd_ip          (int argc, char *argv[]);
136 int Cmd_set_clock   (int argc, char *argv[]);
137 int Cmd_set_limits  (int argc, char *argv[]);
138 int Cmd_set_current(int argc, char *argv[]);
139 int Cmd_max_time    (int argc, char *argv[]);
140 int Cmd_max_temp    (int argc, char *argv[]);
141 int Cmd_max_charge  (int argc, char *argv[]);
142 int Cmd_load        (int argc, char *argv[]);
143
144 // Für die interne Nutzung
145 void config_erase(void);
146 void config_print_uart(void);
147 void config_write_struct(const char *conf_name, const char *conf_val);
148
149 void calibrate_adc(void);
150
151 #endif /* CONFIG_H_ */
```

Quellcode E.23: Anhang/h/control.h

```
1  /*
2   * control.h
3   *
4   *  Created on: 25.12.2011
5   *      Author: Fabian S.
6   *      Credits: Thomas W.
7   */
8
9  #ifndef CONTROL_H_
10 #define CONTROL_H_
11
12 #include "lwip/tcp.h"
13
14 /*****************************************************************************
15 *
16 * Function Declarations
17 *
18 *****************************************************************************/
19 void control_init(void);
20 err_t control_new_con(void* arg, struct tcp_pcb* newpcb, err_t err);
21 err_t control_rx(void* arg, struct tcp_pcb* tpcb, struct pbuf* p, err_t err);
22 void control_tick(unsigned char *data_check, unsigned long data_length);
23 void telnet_write(const void *data);
24 void telnet_immediate_write(const void *data);
25
26
27 typedef enum
28 {
29  MAIN,
30  BROWSER,
31  CONFIG
32
33 } control_state_t;
34
35
36
37 #endif /* CONTROL_H_ */
```

Quellcode E.24: Anhang/h/discover.h

```
1  /*
2   * discover.h
3   *
4   *  Created on: 16.07.2012
5   *      Author: Thomas W. / Fabian S. / Tobias S.
6   */
7
8  #ifndef DISCOVER_H_
9  #define DISCOVER_H_
10
```

```
/*****************************************************************************
*
* Function Declarations
*
*****************************************************************************/
void discover_init(void);
void discover_rx(void* arg, struct udp_pcb* upcb, struct pbuf* p, struct ip_addr* addr, u16_t port);
void ipToString(char* string, unsigned long ip, int maxLen);

#endif /* DISCOVER_H_ */
```

Quellcode E.25: Anhang/h/display.h

```
/*
 * display.h
 *
 *  Created on: 25.03.2013
 *      Author: Thomas W.
 *              Johannes R.
 *      modified: 13.01.2014
 */

#ifndef DISPLAY_H_
#define DISPLAY_H_

#define LEDPORTCTRL      GPIO_PORTH_BASE
#define LEDPORTDATA     GPIO_PORTJ_BASE

#define RS          GPIO_PIN_0
#define RW       GPIO_PIN_1
#define E        GPIO_PIN_2


#define D4        GPIO_PIN_0
#define D5        GPIO_PIN_1
#define D6        GPIO_PIN_2
#define D7        GPIO_PIN_3


// Displaymodes
typedef enum
{
 ERROR,
 ERRCURR,
 ERRCHARGE,
 ERRVOLT,
 ERRTIME,
 ERRTEMP,
 START,
 CLOCK,
 MEAS,
 MEASSTOP,
 CELLA,
 CELLB,
 CELLC,
 CELLD,
 CURRENT,
 IP,
 CONF,
 CURRSTEP,
 CALPLUG,
 CALMEAS,
 CALPULL
} display_handler_t;

// Instance of display mode
volatile display_handler_t handlerState;

/*****************************************************************************
*
* Function Declarations
*
*****************************************************************************/

void display_init(void);
void display_reinit(void);
void buttons_init(void);
void clear_display(void);
void write_to_display(char* inputText, unsigned char row, unsigned char col);
void set_position(unsigned char address);
void write_char(unsigned char inputData);
void LEDcommand(unsigned char command);

```

```
71 void GPIOD01Handler(void);
72 void GPIOF00Handler(void);
73 void GPIOC04Handler(void);
74 void GPIOH05Handler(void);
75 void Timer1IntHandler(void);
76
77
78 #endif /* DISPLAY_H_ */
```

Quellcode E.26: Anhang/h/ethernet.h

```
1  /*
2   * ethernet.h
3   *
4   *  Created on: 13.01.2014
5   *      Author: Thomas W. / Tobias S. / Fabian S./ Johannes R.
6   */
7
8  #ifndef ETHERNET_H_
9  #define ETHERNET_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 int init_ethernet(void);
17 void lwIPHostTimerHandler(void);
18
19 #define MACADDR 0x0010A7180EF5; //MAC: 00:10:A7:18:0E:F5
20
21
22 #endif /* ETHERNET_H_ */
```

Quellcode E.27: Anhang/h/led.h

```
1  /*
2   * led.h
3   *
4   *  Created on: 03.06.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef LED_H_
9  #define LED_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void led_init(void);
17
18
19
20 #endif /* LED_H_ */
```

Quellcode E.28: Anhang/h/lwipopts.h

```
1  //*****************************************************************************
2  //
3  // lwipopts.h - Configuration file for lwIP
4  //
5  // Copyright (c) 2009-2011 Texas Instruments Incorporated.  All rights reserved.
6  // Software License Agreement
7  //
8  // Texas Instruments (TI) is supplying this software for use solely and
9  // exclusively on TI's microcontroller products. The software is owned by
10 // TI and/or its suppliers, and is protected under applicable copyright
11 // laws. You may not combine this software with "viral" open-source
12 // software in order to form a larger program.
13 //
14 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
15 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
16 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
17 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
18 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
19 // DAMAGES, FOR ANY REASON WHATSOEVER.
```

```
20 //
21 // This is part of revision 8264 of the EK-LM3S9B92 Firmware Package.
22 //
23 //*****************************************************************************
24 //
25 // NOTE:  This file has been derived from the lwIP/src/include/lwip/opt.h
26 // header file.
27 //
28 // For additional details, refer to the original "opt.h" file, and lwIP
29 // documentation.
30 //
31 //*****************************************************************************
32
33 #ifndef __LWIPOPTS_H__
34 #define __LWIPOPTS_H__
35
36 //*****************************************************************************
37 //
38 // ---------- Stellaris / lwIP Port Options ----------
39 //
40 //*****************************************************************************
41 #define HOST_TMR_INTERVAL               100         // default is 0
42 //#define DHCP_EXPIRE_TIMER_MSECS         (60 * 1000)
43 //#define INCLUDE_HTTPD_SSI
44 //#define INCLUDE_HTTPD_CGI
45 #define DYNAMIC_HTTP_HEADERS
46 //#define INCLUDE_HTTPD_DEBUG
47
48 //*****************************************************************************
49 //
50 // ---------- Platform specific locking ----------
51 //
52 //*****************************************************************************
53 #define SYS_LIGHTWEIGHT_PROT            1           // default is 0
54 #define NO_SYS                          1           // default is 0
55 //#define MEMCPY(dst,src,len)             memcpy(dst,src,len)
56 //#define SMEMCPY(dst,src,len)            memcpy(dst,src,len)
57
58 //*****************************************************************************
59 //
60 // ---------- Memory options ----------
61 //
62 //*****************************************************************************
63 //#define MEM_LIBC_MALLOC                 0
64 #define MEM_ALIGNMENT                   4           // default is 1
65 #define MEM_SIZE                        (22 * 1024)  // default is 1600, was 16K
66 //#define MEMP_OVERFLOW_CHECK             0
67 //#define MEMP_SANITY_CHECK               0
68 //#define MEM_USE_POOLS                   0
69 //#define MEMP_USE_CUSTOM_POOLS           0
70
71 //*****************************************************************************
72 //
73 // ---------- Internal Memory Pool Sizes ----------
74 //
75 //*****************************************************************************
76 #define MEMP_NUM_PBUF                   24    // Default 16, was 16
77 //#define MEMP_NUM_RAW_PCB                4
78 //#define MEMP_NUM_UDP_PCB                4
79 #define MEMP_NUM_TCP_PCB                16    // Default 5, was 12
80 //#define MEMP_NUM_TCP_PCB_LISTEN         8
81 //#define MEMP_NUM_TCP_SEG                16
82 //#define MEMP_NUM_REASSDATA              5
83 //#define MEMP_NUM_ARP_QUEUE              30
84 //#define MEMP_NUM_IGMP_GROUP             8
85 //#define MEMP_NUM_SYS_TIMEOUT            3
86 //#define MEMP_NUM_NETBUF                 2
87 //#define MEMP_NUM_NETCONN                4
88 //#define MEMP_NUM_TCPIP_MSG_API          8
89 //#define MEMP_NUM_TCPIP_MSG_INPKT        8
90 #define PBUF_POOL_SIZE                  24    // Default 16, was 36
91
92 //*****************************************************************************
93 //
94 // ---------- ARP options ----------
95 //
96 //*****************************************************************************
97 //#define LWIP_ARP                        1
98 //#define ARP_TABLE_SIZE                  10
99 //#define ARP_QUEUEING                    1
100 //#define ETHARP_TRUST_IP_MAC             1
101
102 //*****************************************************************************
103 //
104 // ---------- IP options ----------
```

```
105 //
106 //*****************************************************************************
107 //#define IP_FORWARD                      0
108 //#define IP_OPTIONS_ALLOWED              1
109 #define IP_REASSEMBLY                   0           // default is 1
110 #define IP_FRAG                         0           // default is 1
111 //#define IP_REASS_MAXAGE                 3
112 //#define IP_REASS_MAX_PBUFS              10
113 //#define IP_FRAG_USES_STATIC_BUF         1
114 //#define IP_FRAG_MAX_MTU                 1500
115 //#define IP_DEFAULT_TTL                  255
116
117 //*****************************************************************************
118 //
119 // ---------- ICMP options ----------
120 //
121 //*****************************************************************************
122 //#define LWIP_ICMP                       1
123 //#define ICMP_TTL                        (IP_DEFAULT_TTL)
124
125 //*****************************************************************************
126 //
127 // ---------- RAW options ----------
128 //
129 //*****************************************************************************
130 //#define LWIP_RAW                        1
131 //#define RAW_TTL                         (IP_DEFAULT_TTL)
132
133 //*****************************************************************************
134 //
135 // ---------- DHCP options ----------
136 //
137 //*****************************************************************************
138 #define LWIP_DHCP                       1           // default is 0
139 //#define DHCP_DOES_ARP_CHECK             ((LWIP_DHCP) && (LWIP_ARP))
140
141 //*****************************************************************************
142 //
143 // ---------- UPNP options ----------
144 //
145 //*****************************************************************************
146 //#define LWIP_UPNP                       0
147
148 //*****************************************************************************
149 //
150 // ---------- PTPD options ----------
151 //
152 //*****************************************************************************
153 //#define LWIP_PTPD                       0
154
155 //*****************************************************************************
156 //
157 // ---------- AUTOIP options ----------
158 //
159 //*****************************************************************************
160 #define LWIP_AUTOIP                     1           // default is 0
161 #define LWIP_DHCP_AUTOIP_COOP           ((LWIP_DHCP) && (LWIP_AUTOIP))
162                                                     // default is 0
163 #define LWIP_DHCP_AUTOIP_COOP_TRIES     5           // default is 9
164
165 //*****************************************************************************
166 //
167 // ---------- SNMP options ----------
168 //
169 //*****************************************************************************
170 //#define LWIP_SNMP                       0
171 //#define SNMP_CONCURRENT_REQUESTS        1
172 //#define SNMP_TRAP_DESTINATIONS          1
173 //#define SNMP_PRIVATE_MIB                0
174 //#define SNMP_SAFE_REQUESTS              1
175
176 //*****************************************************************************
177 //
178 // ---------- IGMP options ----------
179 //
180 //*****************************************************************************
181 //#define LWIP_IGMP                       0
182
183 //*****************************************************************************
184 //
185 // ---------- DNS options ----------
186 //
187 //*****************************************************************************
188 //#define LWIP_DNS                        0
189 //#define DNS_TABLE_SIZE                  4
```

```
190 //#define DNS_MAX_NAME_LENGTH             256
191 //#define DNS_MAX_SERVERS                 2
192 //#define DNS_DOES_NAME_CHECK             1
193 //#define DNS_USES_STATIC_BUF             1
194 //#define DNS_MSG_SIZE                    512
195
196 //----------------------------------------------------------------------------
197 //
198 // ---------- UDP options ----------
199 //
200 //----------------------------------------------------------------------------
201 //#define LWIP_UDP                        1
202 //#define LWIP_UDPLITE                    0
203 //#define UDP_TTL                         (IP_DEFAULT_TTL)
204
205 //----------------------------------------------------------------------------
206 //
207 // ---------- TCP options ----------
208 //
209 //----------------------------------------------------------------------------
210 //#define LWIP_TCP                        1
211 //#define TCP_TTL                         (IP_DEFAULT_TTL)
212 #define TCP_WND                         0xFFFF    // default is 2048
213 //#define TCP_WND                         45 * TCP_MSS    // default is 2048
214 //#define TCP_MAXRTX                      12
215 //#define TCP_SYNMAXRTX                   6
216 //#define TCP_QUEUE_OOSEQ                 1
217 #define TCP_MSS                         1460
218 //#define TCP_MSS                         1400         // default is 128
219 //#define TCP_CALCULATE_EFF_SEND_MSS      1
220 #define TCP_SND_BUF                     TCP_WND /* default is 256, was 6 */
221 //#define TCP_SND_QUEUELEN                48
222 #define TCP_SND_QUEUELEN                (6 * (TCP_SND_BUF/TCP_MSS))
223 //#define TCP_SNDLOWAT                    (TCP_SND_BUF/2)
224 //#define TCP_LISTEN_BACKLOG              0
225 //#define TCP_DEFAULT_LISTEN_BACKLOG      0xff
226 //#define TCP_OVERSIZE TCP_MSS
227
228
229 //----------------------------------------------------------------------------
230 //
231 // ---------- API options ----------
232 //
233 //----------------------------------------------------------------------------
234 //#define LWIP_EVENT_API                  0
235 //#define LWIP_CALLBACK_API               1
236
237 //----------------------------------------------------------------------------
238 //
239 // ---------- Pbuf options ----------
240 //
241 //----------------------------------------------------------------------------
242 #define PBUF_LINK_HLEN                  16           // default is 14
243 //#define PBUF_POOL_BUFSIZE               2048
244 #define PBUF_POOL_BUFSIZE               1024
245                                                      // default is LWIP_MEM_ALIGN_SIZE(TCP_MSS+40+PBUF_LINK_HLEN) //256
246 #define ETH_PAD_SIZE                    2            // default is 0
247
248 //----------------------------------------------------------------------------
249 //
250 // ---------- Network Interfaces options ----------
251 //
252 //----------------------------------------------------------------------------
253 //#define LWIP_NETIF_HOSTNAME             0
254 //#define LWIP_NETIF_API                  0
255 //#define LWIP_NETIF_STATUS_CALLBACK      0
256 //#define LWIP_NETIF_LINK_CALLBACK        0
257 //#define LWIP_NETIF_HWADDRHINT           0
258
259 //----------------------------------------------------------------------------
260 //
261 // ---------- LOOPIF options ----------
262 //
263 //----------------------------------------------------------------------------
264 //#define LWIP_HAVE_LOOPIF                0
265 //#define LWIP_LOOPIF_MULTITHREADING      1
266
267 //----------------------------------------------------------------------------
268 //
269 // ---------- Thread options ----------
270 //
271 //----------------------------------------------------------------------------
272 //#define TCPIP_THREAD_NAME               "tcpip_thread"
273 //#define TCPIP_THREAD_STACKSIZE          0
274 //#define TCPIP_THREAD_PRIO               1
```

```
275 //#define TCPIP_MBOX_SIZE                 0
276 //#define SLIPIF_THREAD_NAME              "slipif_loop"
277 //#define SLIPIF_THREAD_STACKSIZE         0
278 //#define SLIPIF_THREAD_PRIO              1
279 //#define PPP_THREAD_NAME                 "pppMain"
280 //#define PPP_THREAD_STACKSIZE            0
281 //#define PPP_THREAD_PRIO                 1
282 //#define DEFAULT_THREAD_NAME             "lwIP"
283 //#define DEFAULT_THREAD_STACKSIZE        0
284 //#define DEFAULT_THREAD_PRIO             1
285 //#define DEFAULT_RAW_RECVMBOX_SIZE       0
286 //#define DEFAULT_UDP_RECVMBOX_SIZE       0
287 //#define DEFAULT_TCP_RECVMBOX_SIZE       0
288 //#define DEFAULT_ACCEPTMBOX_SIZE         0
289
290 //----------------------------------------------------------------------------
291 //
292 // ---------- Sequential layer options ----------
293 //
294 //----------------------------------------------------------------------------
295 //#define LWIP_TCPIP_CORE_LOCKING         0
296 #define LWIP_NETCONN                    0           // default is 1
297
298 //----------------------------------------------------------------------------
299 //
300 // ---------- Socket Options ----------
301 //
302 //----------------------------------------------------------------------------
303 #define LWIP_SOCKET                     0           // default is 1
304 //#define LWIP_COMPAT_SOCKETS             1
305 //#define LWIP_POSIX_SOCKETS_IO_NAMES     1
306 //#define LWIP_TCP_KEEPALIVE              0
307 //#define LWIP_SO_RCVTIMEO                0
308 //#define LWIP_SO_RCVBUF                  0
309 //#define SO_REUSE                        0
310
311 //----------------------------------------------------------------------------
312 //
313 // ---------- Statistics options ----------
314 //
315 //----------------------------------------------------------------------------
316 //#define LWIP_STATS                      1
317 //#define LWIP_STATS_DISPLAY              0
318 //#define LINK_STATS                      1
319 //#define ETHARP_STATS                    (LWIP_ARP)
320 //#define IP_STATS                        1
321 //#define IPFRAG_STATS                    (IP_REASSEMBLY || IP_FRAG)
322 //#define ICMP_STATS                      1
323 //#define IGMP_STATS                      (LWIP_IGMP)
324 //#define UDP_STATS                       (LWIP_UDP)
325 //#define TCP_STATS                       (LWIP_TCP)
326 //#define MEM_STATS                       1
327 //#define MEMP_STATS                      1
328 //#define SYS_STATS                       1
329
330 //----------------------------------------------------------------------------
331 //
332 // ---------- PPP options ----------
333 //
334 //----------------------------------------------------------------------------
335 //#define PPP_SUPPORT                     0
336 //#define PPPOE_SUPPORT                   0
337 //#define PPPOS_SUPPORT                   PPP_SUPPORT
338
339 #if PPP_SUPPORT
340 //#define NUM_PPP                         1
341 //#define PAP_SUPPORT                     0
342 //#define CHAP_SUPPORT                    0
343 //#define MSCHAP_SUPPORT                  0
344 //#define CBCP_SUPPORT                    0
345 //#define CCP_SUPPORT                     0
346 //#define VJ_SUPPORT                      0
347 //#define MD5_SUPPORT                     0
348 //#define FSM_DEFTIMEOUT                  6
349 //#define FSM_DEFMAXTERMREQS              2
350 //#define FSM_DEFMAXCONFREQS              10
351 //#define FSM_DEFMAXNAKLOOPS              5
352 //#define UPAP_DEFTIMEOUT                 6
353 //#define UPAP_DEFREQTIME                 30
354 //#define CHAP_DEFTIMEOUT                 6
355 //#define CHAP_DEFTRANSMITS               10
356 //#define LCP_ECHOINTERVAL                0
357 //#define LCP_MAXECHOFAILS                3
358 //#define PPP_MAXIDLEFLAG                 100
359
```

```
360 //#define PPP_MAXMTU                      1500
361 //#define PPP_DEFMRU                      296
362 #endif
363
364 //*****************************************************************************
365 //
366 // ---------- checksum options ----------
367 //
368 //*****************************************************************************
369 //#define CHECKSUM_GEN_IP                 1
370 //#define CHECKSUM_GEN_UDP                1
371 //#define CHECKSUM_GEN_TCP                1
372 //#define CHECKSUM_CHECK_IP               1
373 //#define CHECKSUM_CHECK_UDP              1
374 //#define CHECKSUM_CHECK_TCP              1
375
376 //*****************************************************************************
377 //
378 // ---------- Debugging options ----------
379 //
380 //*****************************************************************************
381 #if 0
382 #define U8_F "c"
383 #define S8_F "c"
384 #define X8_F "x"
385 #define U16_F "u"
386 #define S16_F "d"
387 #define X16_F "x"
388 #define U32_F "u"
389 #define S32_F "d"
390 #define X32_F "x"
391 extern void UARTprintf(const char *pcString, ...);
392 #define LWIP_DEBUG
393 #endif
394
395 //#define LWIP_DBG_MIN_LEVEL              LWIP_DBG_LEVEL_OFF
396 #define LWIP_DBG_MIN_LEVEL              LWIP_DBG_LEVEL_OFF
397 //#define LWIP_DBG_MIN_LEVEL              LWIP_DBG_LEVEL_WARNING
398 //#define LWIP_DBG_MIN_LEVEL              LWIP_DBG_LEVEL_SERIOUS
399 //#define LWIP_DBG_MIN_LEVEL              LWIP_DBG_LEVEL_SEVERE
400
401 //#define LWIP_DBG_TYPES_ON               LWIP_DBG_ON
402 #define LWIP_DBG_TYPES_ON               (LWIP_DBG_ON|LWIP_DBG_TRACE|LWIP_DBG_STATE|LWIP_DBG_FRESH)
403
404 //#define ETHARP_DEBUG                    LWIP_DBG_ON     // default is OFF
405 //#define NETIF_DEBUG                     LWIP_DBG_ON     // default is OFF
406 //#define PBUF_DEBUG                      LWIP_DBG_OFF
407 //#define API_LIB_DEBUG                   LWIP_DBG_OFF
408 //#define API_MSG_DEBUG                   LWIP_DBG_OFF
409 //#define SOCKETS_DEBUG                   LWIP_DBG_OFF
410 //#define ICMP_DEBUG                      LWIP_DBG_OFF
411 //#define IGMP_DEBUG                      LWIP_DBG_OFF
412 //#define INET_DEBUG                      LWIP_DBG_OFF
413 //#define IP_DEBUG                        LWIP_DBG_ON     // default is OFF
414 //#define IP_REASS_DEBUG                  LWIP_DBG_OFF
415 //#define RAW_DEBUG                       LWIP_DBG_OFF
416 //#define MEM_DEBUG                       LWIP_DBG_OFF
417 //#define MEMP_DEBUG                      LWIP_DBG_OFF
418 //#define SYS_DEBUG                       LWIP_DBG_OFF
419 //#define TCP_DEBUG                       LWIP_DBG_OFF
420 //#define TCP_INPUT_DEBUG                 LWIP_DBG_OFF
421 //#define TCP_FR_DEBUG                    LWIP_DBG_OFF
422 //#define TCP_RTO_DEBUG                   LWIP_DBG_OFF
423 //#define TCP_CWND_DEBUG                  LWIP_DBG_OFF
424 //#define TCP_WND_DEBUG                   LWIP_DBG_OFF
425 //#define TCP_OUTPUT_DEBUG                LWIP_DBG_OFF
426 //#define TCP_RST_DEBUG                   LWIP_DBG_OFF
427 //#define TCP_QLEN_DEBUG                  LWIP_DBG_OFF
428 //#define UDP_DEBUG                       LWIP_DBG_ON     // default is OFF
429 //#define TCPIP_DEBUG                     LWIP_DBG_OFF
430 //#define PPP_DEBUG                       LWIP_DBG_OFF
431 //#define SLIP_DEBUG                      LWIP_DBG_OFF
432 //#define DHCP_DEBUG                      LWIP_DBG_ON     // default is OFF
433 //#define AUTOIP_DEBUG                    LWIP_DBG_OFF
434 //#define SNMP_MSG_DEBUG                  LWIP_DBG_OFF
435 //#define SNMP_MIB_DEBUG                  LWIP_DBG_OFF
436 //#define DNS_DEBUG                       LWIP_DBG_OFF
437
438 #endif /* __LWIPOPTS_H__ */
```

Quellcode E.29: Anhang/h/myadc.h

```
1 /*
```

```
2  * myadc.h
3  *
4  *  Created on: 11.02.2013
5  *      Author: Thomas W.
6  *        Oliver Schmidt
7  */
8
9  #ifndef MAYADC_H_
10 #define MAYADC_H_
11
12
13
14
15
16 /************************************
17  * ADC registers and bits           *
18  ************************************/
19 //Configuration register
20 #define CONFIG_UB         0x1000
21 #define CONFIG_GAIN_1     0x0000
22 #define CONFIG_GAIN_2     0x0100
23 #define CONFIG_GAIN_4     0x0200
24 #define CONFIG_GAIN_8     0x0300
25 #define CONFIG_GAIN_16    0x0400
26 #define CONFIG_GAIN_32    0x0500
27 #define CONFIG_GAIN_64    0x0600
28 #define CONFIG_GAIN_128   0x0700
29 #define CONFIG_REFDET     0x0020
30 #define CONFIG_BUF        0x0010
31 #define CONFIG_CH1        0x0000
32 #define CONFIG_CH2        0x0001
33 #define CONFIG_CH3        0x0002
34 #define CONFIG_CH_CLR     0x0003
35
36 //Comunication register
37 #define COM_nWEN    0x80
38 #define COM_RnW     0x40
39 #define COM_COM     0x00
40 #define COM_STATUS    0x00
41 #define COM_MODE    0x08
42 #define COM_CONFIG    0x10
43 #define COM_DATA    0x18
44 #define COM_ID      0x20
45 #define COM_IO      0x28
46 #define COM_OFFSET    0x30
47 #define COM_FULLSCALE 0x38
48 #define COM_CREAD    0x04
49
50 #define MODE_SEL_CONCONV    0x0000
51 #define MODE_SEL_SINCONV    0x2000
52 #define MODE_SEL_IDLE       0x4000
53 #define MODE_SEL_PWRDOWN    0x6000
54 #define MODE_SEL_INTZSCAL    0x8000
55 #define MODE_SEL_INTFSCAL    0xA000
56 #define MODE_SEL_SYSZSCAL    0xC000
57 #define MODE_SEL_SYSFSCAL    0xE000
58 #define MODE_PSW       0x1000
59 #define MODE_FILTER_0     0x0000
60 #define MODE_FILTER_1     0x0001
61 #define MODE_FILTER_2     0x0002
62 #define MODE_FILTER_3     0x0003
63 #define MODE_FILTER_4     0x0004
64 #define MODE_FILTER_5     0x0005
65 #define MODE_FILTER_6     0x0006
66 #define MODE_FILTER_7     0x0007
67 #define MODE_FILTER_8     0x0008
68 #define MODE_FILTER_9     0x0009
69 #define MODE_FILTER_A     0x000A
70 #define MODE_FILTER_B     0x000B
71 #define MODE_FILTER_C     0x000C
72 #define MODE_FILTER_D     0x000D
73 #define MODE_FILTER_E     0x000E
74 #define MODE_FILTER_F     0x000F
75
76 #define STATUS_nRDY      0x80
77 #define STATUS_ERR      0x40
78 #define STATUS_NOREF     0x20
79 #define STATUS_CH0      0x00
80 #define STATUS_CH1      0x02
81 #define STATUS_CH2      0x04
82
83 #define EX_STATUS_ADC_COM_ERR 0x10 // falls 0x00 als der Wert des Statusregisters vom ADC angekommen ist
84
85 typedef enum adc_channel{
86  channel1=1,
```

```
 channel2=2,
 channel3=3
}adc_channel_t;

typedef enum adc_reg8{
 communication,
 status,
 id,
 io
}adc_reg8_t;

typedef enum adc_reg16{
 mode,
 configuration,
 data,
 offset,
 fullscale
}adc_reg16_t;



unsigned char f_adc_error;

#define adc_reg24_t adc_reg16_t


//ADC settings
#define FILTER_MODE    MODE_FILTER_A //fadc = 16,7Hz ; tsettle 120msek ; Rejection 65dB
#define POLARITY_MODE    CONFIG_UB   //Unipolar

#define U_REF   5.0
#define ADC_LSB   (U_REF / 0xffff)
#define G_CH1   26.7
#define G_CH2   4.0

#define KOEFF_CH1   (1.0 / G_CH1)
#define KOEFF_CH2   (1.0 / G_CH2)
#define KOEFF_BG_CH1 (2.5 / G_CH1)
#define KOEFF_BG_CH2 (2.5 / G_CH2)

// Anzahl der Messungen bei der Hall-Effekt Kalibrierung
#define NO_CURRENT_MEASUREMENTS   5

/*******************************************************************************
*
* Function Declarations
*
*******************************************************************************/
int myadc_init(void);
void myadc_reinit(void);
int adc_internal_calibration(adc_channel_t);
unsigned int adc_get_single_value(adc_channel_t);
void adc_calibrate_hall_sensor(void);
int adc_get_current(void);
unsigned int adc_get_voltage(unsigned int);
static unsigned char adc_read_reg8(adc_reg8_t);
static unsigned int adc_read_reg24(adc_reg24_t reg);
static void adc_write_reg8(adc_reg8_t, unsigned char);
static void adc_write_reg24(adc_reg24_t reg, unsigned int value);
static unsigned char adc_write_read(unsigned char);
static void adc_clear_fifo(void);
static void adc_delay_micros(unsigned long);
static void adc_delay_millis(unsigned long);

#endif /* MAYADC_H_ */
```

Quellcode E.30: Anhang/h/mycmdline.h

```
/*
 * mycmdline.h
 *
 *  Created on: 20.02.2013
 *      Author: Thomas W./Stellarisware
 */



//*****************************************************************************
//
// cmdline.h - Prototypes for command line processing functions.
//
// Copyright (c) 2007-2012 Texas Instruments Incorporated.  All rights reserved.
```

```
16 // Software License Agreement
17 //
18 // Texas Instruments (TI) is supplying this software for use solely and
19 // exclusively on TI's microcontroller products. The software is owned by
20 // TI and/or its suppliers, and is protected under applicable copyright
21 // laws. You may not combine this software with "viral" open-source
22 // software in order to form a larger program.
23 //
24 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
25 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
26 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
27 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
28 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
29 // DAMAGES, FOR ANY REASON WHATSOEVER.
30 //
31 // This is part of revision 8555 of the Stellaris Firmware Development Package.
32 //
33 //*****************************************************************************
34
35 #ifndef __CMDLINE_H__
36 #define __CMDLINE_H__
37
38 //*****************************************************************************
39 //
40 // If building with a C++ compiler, make all of the definitions in this header
41 // have a C binding.
42 //
43 //*****************************************************************************
44 #ifdef __cplusplus
45 extern "C"
46 {
47 #endif
48
49 //*****************************************************************************
50 //
51 //! \addtogroup cmdline_api
52 //! @{
53 //
54 //*****************************************************************************
55
56 //*****************************************************************************
57 //
58 //! Defines the value that is returned if the command is not found.
59 //
60 //*****************************************************************************
61 #define CMDLINE_BAD_CMD         (-1)
62
63 //*****************************************************************************
64 //
65 //! Defines the value that is returned if there are too many arguments.
66 //
67 //*****************************************************************************
68 #define CMDLINE_TOO_MANY_ARGS   (-2)
69
70 //*****************************************************************************
71 //
72 // Command line function callback type.
73 //
74 //*****************************************************************************
75 typedef int (*pfnCmdLine)(int argc, char *argv[]);
76
77 //*****************************************************************************
78 //
79 //! Structure for an entry in the command list table.
80 //
81 //*****************************************************************************
82 typedef struct
83 {
84     //
85     //! A pointer to a string containing the name of the command.
86     //
87     const char *pcCmd;
88
89     //
90     //! A function pointer to the implementation of the command.
91     //
92     pfnCmdLine pfnCmd;
93
94     //
95     //! A pointer to a string of brief help text for the command.
96     //
97     const char *pcHelp;
98 }
99 tCmdLineEntry;
100
```

```
101 //*****************************************************************************
102 //
103 //! This is the command table that must be provided by the application.
104 //
105 //*****************************************************************************
106 extern tCmdLineEntry g_sCmdTable[];
107
108 //*****************************************************************************
109 //
110 // Close the Doxygen group.
111 //! @}
112 //
113 //*****************************************************************************
114
115 //*****************************************************************************
116 //
117 // Prototypes for the APIs.
118 //
119 //*****************************************************************************
120 extern int CmdLineProcess(char *pcCmdLine);
121 void Cmd_interprete(char *g_cCmdBuf);
122 int Cmd_alive(int argc, char *argv[]);
123
124 //*****************************************************************************
125 //
126 // Mark the end of the C bindings section for C++ compilers.
127 //
128 //*****************************************************************************
129 #ifdef __cplusplus
130 }
131 #endif
132
133 #endif // __CMDLINE_H__
```

Quellcode E.31: Anhang/h/mypwm.h

```
1  /*
2   * mypwm.h
3   *
4   *  Created on: 22.03.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef MYPWM_H_
9  #define MYPWM_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void pwm_init(void);
17
18 #endif /* MYPWM_H_ */
```

Quellcode E.32: Anhang/h/mysdcard.h

```
1  /*
2   * mysdcard.h
3   *
4   *  Created on: 21.01.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef MYSDCARD_H_
9  #define MYSDCARD_H_
10
11 #include "third_party/fatfs/src/ff.h"
12
13 /*****************************************************************************
14 *
15 * Function Declarations
16 *
17 *****************************************************************************/
18 int Cmd_cat(int argc, char *argv[]);
19 int Cmd_cd(int argc, char *argv[]);
20 int Cmd_help(int argc, char *argv[]);
21 int Cmd_reinit(int argc, char *argv[]);
22 int Cmd_ls(int argc, char *argv[]);
23 int Cmd_pwd(int argc, char *argv[]);
24 const char * StringFromFresult(FRESULT);
```

```
25 int Cmd_sdcard_exit(int argc, char *argv[]);
26
27
28 int add_to_file(const char *filename, const char *write_Buffer);
29 int delete_file(const char *filename);
30 int read_file(const char *filename);
31 int read_into_buffer(const char *filename, char *buffer);
32 int write_to_file(const char *filename, const char *write_Buffer);
33 int create_file(const char *filename);
34 void do_measure();
35
36 void SystickHandler(void);
37
38
39 void mysdcardinit(void);
40 int my_start_cmd_line(int argc, char *argv[]);
41 int Cmd_del(int argc, char *argv[]);
42 int Cmd_cre(int argc, char *argv[]);
43
44
45 //*****************************************************************************
46 //
47 // Defines the size of the buffers that hold the path, or temporary
48 // data from the SD card.  There are two buffers allocated of this size.
49 // The buffer size must be large enough to hold the longest expected
50 // full path name, including the file name, and a trailing null character.
51 //
52 //*****************************************************************************
53 #define PATH_BUF_SIZE   80
54
55 // Define for the max measurement cycles in ERROR-Case
56 #define VOLTCOUNTMAX 3
57 #define CURRENTCOUNTMAX 3
58
59
60 //*****************************************************************************
61 //
62 // This buffer holds the full path to the current working directory.
63 // Initially it is root ("/").
64 //
65 //*****************************************************************************
66 static char g_cCwdBuf[PATH_BUF_SIZE] = "/";
67
68 //*****************************************************************************
69 //
70 // A temporary data buffer used when manipulating file paths, or reading data
71 // from the SD card.
72 //
73 //*****************************************************************************
74 static char g_cTmpBuf[PATH_BUF_SIZE];
75
76
77 //*****************************************************************************
78 //
79 // The following are data structures used by FatFs.
80 //
81 //*****************************************************************************
82 static FATFS g_sFatFs;
83 static DIR g_sDirObject;
84 static FILINFO g_sFileInfo;
85 static FIL g_sFileObject;
86
87
88
89 #endif /* MYSDCARD_H_ */
```

Quellcode E.33: Anhang/h/myssi.h

```
1 /*
2  * myssi.h
3  *
4  *  Created on: 26.05.2013
5  *      Author: Thomas W.
6  */
7
8 #ifndef MYSSI_H_
9 #define MYSSI_H_
10
11
12
13 // SPI chip selects
14 typedef enum
15  {
```

```
  ADC,
  SD,
  NONE
 }
spi_cs_t;

/*****************************************************************************
*
* Function Declarations
*
*****************************************************************************/
void ssi1_init(void);
void spiChipSelect(spi_cs_t chip);

#endif /* MYSSI_H_ */
```

Quellcode E.34: Anhang/h/myuart.h

```
/*
 * myuart.h
 *
 *  Created on: 17.01.2013
 *      Author: Thomas W.
 */

#ifndef MYUART_H_
#define MYUART_H_

#define MYUART_SELECT_0  //To select UART0 (USB)
#define MYUART_SELECT_1  //To select UART1 (RS232)
//#define MYUART_SELECT_2 //To select UART2 (RS232/485)

/*
 * Code to printf a text (TIME Stamp included) by UART1 (RS232) if UART0 (USB) is selected too:
     UARTStdioInit(1);
     UARTprintf("%04d,%02d,%02d,%02d,%02d,%03d ", clock_year, clock_month, clock_day, clock_hour, clock_min, clock_sec, clock_msec
         );
     UARTprintf(" Watchdog feed\n");
     UARTStdioInit(0);
 */

/*****************************************************************************
*
* Function Declarations
*
*****************************************************************************/
void myUARTinit(void);

int myUARTgets(char *pcBuf, unsigned long ulLen);

void UART0IntHandler(void);

void UART1IntHandler(void);
void UART2IntHandler(void);




#endif /* MYUART_H_ */
```

Quellcode E.35: Anhang/h/onewire.h

```
/*
 * onewire.h
 *
 *  Created on: 28.01.2015
 *      Author: f.groth
 */

#ifndef ONEWIRE_H_
#define ONEWIRE_H_

#include <stdbool.h>
#include <stdint.h>

//----------inc----------
//#include "inc/tm4c1294ncpdt.h"
#include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"

```

```
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/pin_map.h"


void initOneWireUART(void);
int initOneWire(char enablePullup);
void writeOneWire(char byte, char enablePullup);
char readOneWire(char enablePullup);
uint8_t searchOneWire(uint64_t* devices, uint8_t maxDevices, uint8_t isAlarm);
int matchOneWireDevice(uint64_t * addr);

#endif /* ONEWIRE_H_ */
```

Quellcode E.36: Anhang/h/relais.h

```
/*
 * relais.h
 *
 *  Created on: 15.07.2012
 *      Author: Thomas W.
 */

#ifndef RELAIS_H_
#define RELAIS_H_

/*******************************************************************************
*
* Function Declarations
*
*******************************************************************************/
void init_relais(void);
void switch_relais(int choose);
int switch_power_relais(int choose);
int Cmd_relais(int argc, char *argv[]);

static void relay_delay_micros(unsigned long);

#endif /* RELAIS_H_ */
```

Quellcode E.37: Anhang/h/rtc.h

```
/*
 * rtc.h
 *
 *  Created on: 03.06.2013
 *      Author: Thomas W.
 */

#ifndef RTC_H_
#define RTC_H_

/*******************************************************************************
*
* Function Declarations
*
*******************************************************************************/
void rtc_init(void);
void set_clock_values(void);
void set_rtc_values(unsigned int year, unsigned int month, unsigned int day, unsigned int hour, unsigned int min, unsigned int sec );

#endif /* RTC_H_ */
```

Quellcode E.38: Anhang/h/temperature.h

```
/*
 * temperature.h
 *
 *  Created on: 06.03.2013
 *      Author: Thomas W.
 */

#ifndef TEMPERATURE_H_
#define TEMPERATURE_H_

/*******************************************************************************
```

```
12 *
13 * Function Declarations
14 *
15 ******************************************************************************/
16 void init_temperature(void);
17 int get_temperature(int chose);
18
19
20 #endif /* TEMPERATURE_H_ */
```

Abkürzungsverzeichnis

App	Applikation
BATSEN	Drahtlose Zellensensoren für Fahrzeugbatterien
BEEDeL	Bewertung des Einsatzes von Elektrobussen mit Dezentraler Ladeinfrastruktur in Metropolen am Beispiel der HOCHBAHN
CC-CV	IU-Ladeverfahren (constant current constant voltage)
CCS	Code Composer Studio ist die Entwicklungsumgebung (IDE) von Texas Instruments.
DOD	Entladetiefe
E_N	Energiegehalt
EDDA	Elektromobilitäts-Demonstration Docking-Anwendung
I_K	Klemmstrom
IDE	integrated development environment
IVI	Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme
$LiFePO_4$	Lithium-Eisenphosphat
NTC	Widerstand mit einen negativen Temperaturkoeffizienten
Q_{Bat}	Kapazität
Q_N	Nennkapazität
R_i	Innenwiderstand
SEI	Solid Electrolyte Interface
SOC	Ladezustand
SOH	Allgemeinzustand der Batterie
U_0	Leerlaufspannung
$U_{dischargeMin}$	Entladeschlussspannung
$U_{dischgeMax}$	Ladeschlussspannung
U_K	Klemmspannung
U_N	Nennspannung
U_R	Ruhespannung
Zykliergerät R	Das von Johannes Röhn weiterentwickelte Zykliergerät.

Zykliergerät W ... Das von Thomas Wisniewski entwickelte Zykliergerät.

Zyklus ..... Kreislauf regelmäßig wiederkehrender Dinge oder Ereignisse.

Zyklus-E ..... Der Zyklus der auf den Messdaten des EDDA-Busses vom Fraunhofer-Institut für Verkehrs- und Infrastruktursysteme basiert

Zyklus-I ..... Der Zyklus der auf den Beschleunigungsdaten der Innovationslinie basiert.

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM 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, 15. Juni 2015	
Ort, Datum	Unterschrift