Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Martin Kusche

Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Martin Kusche

Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller

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. Ralf Wendel

Abgegeben am 11. September 2014

Martin Kusche

Thema der Bachelorthesis

Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller

Stichworte

ARM-Mikrocontroller, Batteriezellen, Batteriezyklen, Zyklierung, Instrumentenverstärker, Analog-Digital-Umsetzer, Ethernet, RS232, Temperaturschrank, E-Mobilität, Neuen Europäischen Fahrzyklus

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit einem Redesign der Analogschaltung und einigen Verbesserung der Schnittstellen des Zykliersystem. Des weiteren wird ein Konzept zur Prüfung von Lithiumzellen, bezogen auf E-Mobilität, entwickelt.

Martin Kusche

Title of the paper

Construction and commissioning of a cycling system for battery cells with a ARM microcontroller

Keywords

ARM microcontroller, battery cells, battery cycles, cycling, analog-digital-converter, instrumentation amplifiers, ethernet, RS232, temperature chamber, e-mobility, new european driving cycle

Abstract

This report deals with a redesign of the analog circuit and some improvements of the interfaces of the cycling system. Additionally a concept is developed for the testing of lithium cells in relation to e-mobility.

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelor-Arbeit unterstützt und motiviert haben. Ganz besonders gilt dieser Dank Herrn Prof. Dr. -Ing. Karl-Ragmar Riemschneider, der meine Arbeit und somit auch mich betreut hat. Nicht nur, dass er immer durch kritisches Hinterfragen wertvolle Hinweise gab, auch seine moralische Unterstützung und Motivation waren unschlagbar. Er hat mich dazu gebracht, über meine Grenzen hinaus zu denken. Vielen Dank für die Geduld und Mühen.

Auch möchte ich mich bei der gesamten BATSEN Gruppe bedanken, für das kritische Hinterfragen und die konstruktive Kritik während des Meetings. Besonders hervorzuheben aus der BATSEN Gruppe ist Herr Dipl. -Ing. Günter Müller und Herr Prof. Dr. -Ing. Jürgen Vollmer für die Erfahrungen, die sie mit mir geteilt haben im Aufbau analoger Schaltungen.

Daneben gilt mein Dank meinen Eltern, Schwiegereltern und meinen beiden Schwestern, die in zahlreichen Stunden Korrektur gelesen haben. Sie wiesen auf Schwächen hin und konnten als Fachfremde zeigen, wo noch Erklärungsbedarf bestand.

Ein besonderer Dank gilt meiner Frau, Claudia Kusche, die mir zuhause viel Last von den Schultern nahm und mich seelisch unterstützt hat.

Inhaltsverzeichnis

1 Einführung

Das „BATSEN - Drahtlose Zellensensoren für Fahrzeugbatterien“ Forschungsprojekt der HAW Hamburg ist ambitioniert, mehr über das Verhalten von derzeit verwendeten Fahrzeugbatterien zu erfahren. Deshalb wird ein Zykliersystem gebraucht, viele Eigenschaften einer Batteriezelle offenbaren sich nach mehrmaligen Laden und Entladen. Aus den gewonnenen Informationen einiger Zyklierungen kann beispielsweise auf die Lebensdauer, das Alter und der Kapazität geschlossen werden.

Aus thematischer Sicht besteht die Thesis aus zwei Teilen. Der erste Teil setzt sich mit der Verbesserung des Zykliersystems auseinander. Dazu gehört die Verbesserung des störanfälligen ADCs sowie die Inbetriebnahme der Schnittstellen für eine einfachere Handhabung. Der zweite Teil der Thesis beschäftigt sich damit, eine Laborumgebung zu schaffen, die der wirklichen Umgebung einer Lithium-Ionen-Batterie für Fahrzeuge mit Hybrid- oder Elektroantrieb nahe kommt. Dazu steht ein Temperaturschrank, eine elektronische Last und ein Netzteil zu Verfügung.

Des weiteren soll am Ende der Thesis eine Dokumentation erstellt werden, die allen Mitgliedern des BATSEN Forschungsprojekts einen Einstieg in die Bedienung des Zykliersystems verschafft.

1.1 Vorarbeiten

Die Thesis knüpft an vorhandene Thesen an, da bereits ein Zykliersystem im BATSEN Forschungsprojekt entwickelt wurde. Herr Wisniewski hat im Sommersemester 2013 seine Thesis [1] eingereicht. Im Laufe dieser Thesis entwickelte er den Prototypen eines Zykliersystems. Dabei orientierte er sich bei der Dimensionierung an gewonnene Messwerte von Starterbatterien auf $LiFePO_4$ Basis. Diese Messwerte sind das Ergebnis zweier Thesen von Lars Hillermann [2] und Rico Loschwitz [3]. Des Weiteren übernahm Herr Wisniewski den Debugger für den LM3S9B92 [4], µC von Texas Instrument, aus der Thesis von Herrn Schlüter [5] sowie die LwIP Schnittstelle, eine abgespeckte Version des TCP/IP Protokolls, aus Herrn Steinmanns [6] Thesis.

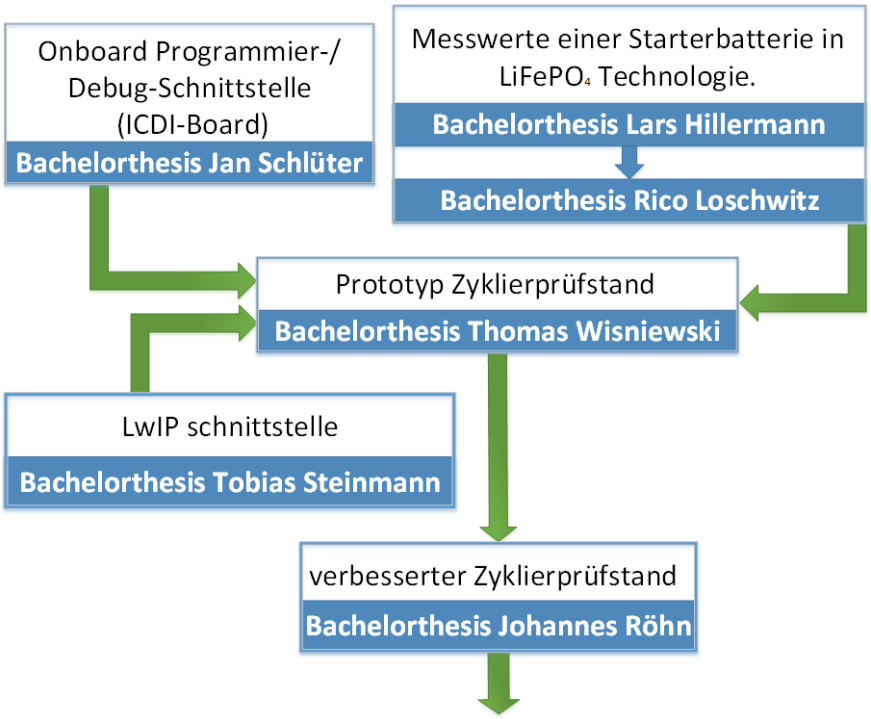


Abbildung 1.1: Das Fundament meiner Thesis

Bei Arbeiten mit dem Prototyp wurden Schwächen in der Strombegrenzung des Lade- und Entlade Vorgangs festgestellt. Aus diesem Grund bekam Herr Röhn die Aufgabe, sich in seiner Thesis [7] mit der Verbesserung des Zykliersystems auseinander zu setzen. Er entwickelte ein neues Zykliersystem auf der Grundlage des Prototypen, dass Lade- und Entladeströme von bis zu 500A zulässt. Des Weiteren löste er noch Probleme in der Bedienung und der Laufzeit einer Messung. Auf diesem Fundament, Abbildung 1.1, werde ich meine Thesis errichten.

1.2 Zykliersystem

Abbildung 1.2 ist ein Auszug aus der aktuellen Kurzanleitung zur Bedienung des Zykliersystems [8, S.6]. Es zeigt einen schematischen Aufbau des Systems, das im Laufe mehrerer Bachelorthesen entwickelt wurde.

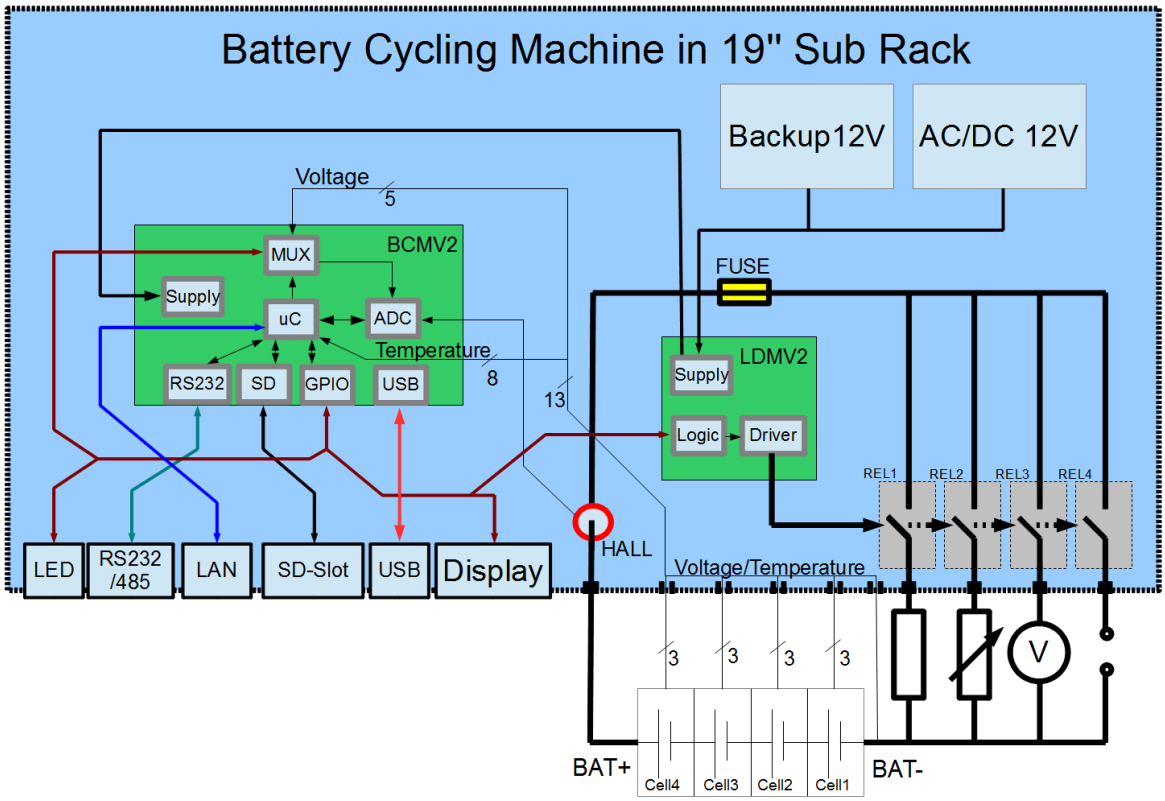


Abbildung 1.2: Schematischer Aufbau einer Zyklierung

Ausgangsfunktionen im Überblick:

- Bis zu 12 Zellen lassen sich gleichzeitig zyklieren, dabei kann von jeder einzelnen Zelle die Spannung und Temperatur bestimmt werden.
- Der Gesamtstrom der Batterie wird über eine HALL-Sonde bestimmt.
- Bis zu 4 Geräte wie z.B. Spannungsmessgeräte, Verbraucher und Netzteile lassen sich automatisch über Relais auf die Gesamtbatterie aufschalten.
- Ein Relais kann bis zu 500A, schalten die anderen 3 bis zu 30A.
- Sicherheitsabschaltung bei zu viel Strom oder einer zu hohen Spannung.

- Integrierter Debugger zum einfachen Programmieren über die USB-Schnittstelle.
- UART-Kommunikation über die USB-Schnittstelle.
- Speicherung aller Messwerte auf SD-Karte.
- LEDs und DOTMATRIX LED-Display zur Statusanzeige.
- 3 entprellte Buttons als Bedienung.

Die Einstellungen einer Zyklierung befinden sich auf der SD-Karte in der Konfigurationsdatei. Sie wird beim Start des Gerätes geladen. Die Konfigurationsdatei enthält Angaben zur Anzahl der zu messenden Zellen, ADC Kalibrierungswerte, IP-Adresse, Dauer der Messung und die Zeitumschaltpunkte für die Relais. Den Start einer Messung kann durch die Buttons und das Display veranlassen oder über die UART-Kommunikation. Eine Messdatenreihe wird aufgenommen, in dem REED-Relais Zelle für Zelle der Batterie auf den differentiellen ADC-Eingang schalten und der ADC bei jeder Zelle die Spannung ermittelt. Anschließend wird über eine HALL-Sonde und dem zweiten Eingang des ADCs der Strom der Batterie gemessen. Am Ende wird die Temperatur aus den Temperatursensoren gelesen. Der fertige Datensatz wird auf die SD-Karte geschrieben und per UART ausgegeben. Während der Zyklierung wird ca. jede Sekunde ein Datensatz erstellt. Die Auswertung dieser Daten erfolgt mit MATLAB.

1.3 Motivation

Auszug aus einer Kienbaum-Studie [9]:

*„**Gummersbach, 14. März 2014** Im Jahr 2025 wird mehr als jeder vierte Neuwagen in Deutschland ein Elektroauto sein. ... Dies belegt eine Studie der Managementberatung Kienbaum, für die mehr als 350 Topmanager, Branchenexperten und Wissenschaftler in Europa, Nordamerika und Asien interviewt wurden. Im Jahr 2020 werden rund 800.000 Elektroautos über Deutschlands Straßen rollen, prognostiziert die Kienbaum-Studie.“*

Diese Studie basiert auf den aktuellen Erkenntnissen einer sorgfältig durchgeführten Marktanalyse. Obwohl man das Gefühl hat, dass sich die Automobilindustrie nicht genügend für die E-Mobilität einsetzt und weltweit immer mehr Erdölreserven entdeckt werden, wird in den nächsten Jahren der Trend zur E-Mobilität gehen.

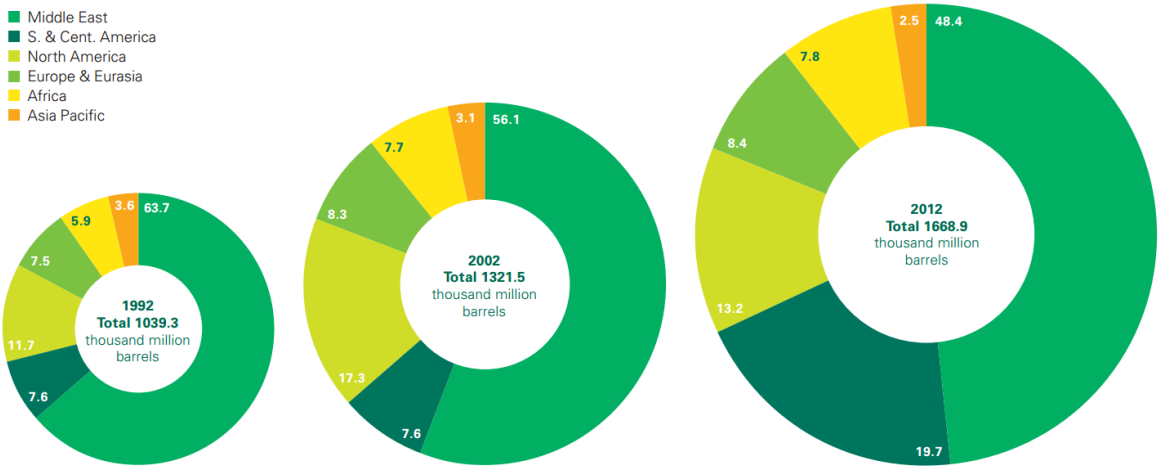


Abbildung 1.3: Entwicklung der gesicherten Erdölreserven

Abbildung 1.3 ist ein Auszug aus „Statistical Review of World Energy June 2013“ von BP [10, S.7]. Es zeigt die Entwicklung der gesicherten Erdölreserven. Die Reichweite der Erdölreserven beträgt nach Schätzungen 40 Jahre, dieser Wert blieb seit 20 Jahren unverändert da die steigende Nachfrage durch die neu entdeckten Reserven kompensiert wurde. Warum soll man sich um die E-Mobilität bemühen, wenn das Ende des Erdöl in weite Ferne rückt. Wie kann die Kienbaum-Studie zu diesem Schluss kommen?

Der Grund hierfür ist der steigende Preis des Erdöls. Die neu erschlossenen Erdölquellen sind nur schwer erreichbar, dadurch steigt der Preis für die Förderung. Dies verschafft der E-Mobilität den nötigen Platz auf dem Markt. Zurzeit hält sich die Automobilindustrie zurück, da es noch zu viele Vorurteile gegenüber der E-Mobilität gibt. Es gibt noch Probleme mit

der Reichweite, der Sicherheit und der Ladedauer. Die Kienbaum-Studie besagt, dass es in Zukunft der Forschung und Entwicklung gelingt, diese Probleme zu lösen.

Dieses Jahr hat zum Beispiel die Firma „StoreDot“ aus Tel Aviv einen Akku für Mobilgeräte entwickelt, dass sie „Speed-Charger“ nennen, da er innerhalb von 30 Sekunden aufgeladen wird. Es soll der erste Akku sein, der Bioorgannische-Komponenten enthält. Die Elektrode wird mit Peptiden beschichtet. Das sind natürliche Nanopartikel, die ursprünglich aus der Medizin kommen und bekannt sind, in kürzester Zeit viel Energie aufzunehmen. Spätestens 2016 will „StoreDot“ die ersten Hochleistungsbatterien für die E-Mobility-Branche auf den Markt bringen.

Da es auf dem Markt immer neue Batterietypen durch Forschung und Entwicklung geben wird, ist ein Zykliersystem unabdingbar. Das BATSEN Forschungsprojekt geht mit der Entwicklung des Zykliersystem einen Schritt in die Zukunft, weil in naher Zukunft das Interesse der Automobilindustrie für Zykliersysteme wachsen wird.

2 Voruntersuchung und Analyse

Zunächst wird sich ein Überblick über bereits existierende Messverfahren in der E-Mobilität Branche verschafft, speziell über die Überprüfung der Batterien. Des Weiteren werden beim bereits bestehenden Zykliersystem die Kommunikationsschnittstellen und die ADC-Schaltung untersucht. Diese Voruntersuchung ist die Grundlage für die spätere Entwicklung und ist entscheidend für den weiteren Verlauf der Thesis.

2.1 Testverfahren von Lithiumzellen in der Automobilindustrie

Die meisten Hersteller von Elektroautos beziehen ihre Lithiumbatterien aus Japan und China von Firmen wie Sanyo, NEC-AESC und Sony. Diese Batteriehersteller sind verpflichtet, gewisse Normen einzuhalten um sie sicher zu transportieren und um eine sichere Benutzung zu gewährleisten. In Europa gelten Lithiumbatterien als Gefahrengut Klasse 9. Deshalb müssen die Hersteller die Norm UN 3480 einhalten. Diese Norm bezieht sich nur auf den Transport und die Lagerung der Zellen. Für die Benutzung von Lithiumzellen gibt es zwar internationale Normen von „UL - Underwriters Laboratories“ und „IEC - International Electrotechnical Commission“, diese sind aber nicht verbindlich. Somit gibt es keine genauen Richtlinien für die Anwendung von Lithiumzellen.

Hersteller	Zulieferer	Batterietyp
Audi	Sanvo (Japan)	Lithium-Ionen
BMW	Conti (D), Johnson Controls-SAFT (F), Cobasvs (US)	Lithium-Ionen, Nickel-Metallhydrid
BMW-Mini	AC Propulsion (US)	Lithium-Ionen
Bosch	Samsung SB LiMotive (Korea)	Lithium-Ionen
Continental	Enax (Japan)	Lithium-Ionen
GM	Conti (D),/A123 (US), LG (Korea)/Chem (US), Cobasvs (US)	Lithium-Ionen, Nickel-Metallhydrid
Honda	Sony, Yuasa (Japan)	Lithium-Ionen
Mercedes-Benz	Conti (D), Johnson Controls-SAFT (F), Hitachi (Japan), LiTec (D), Cobasvs (US)	Lithium-Ionen, Nickel-Metallhydrid
Mercedes-Smart	AC Propulsion (US), LiTec (D)	Natrium-Nickelchlorid, Lithium-Ionen
Mitsubishi	Yuasa (Japan)	Lithium-Ionen
Nissan	NEC-AESC (Japan)	Lithium-Ionen
PSA	Yuasa (Japan)	Lithium-Ionen
Renault	NEC-AESC (Japan)	Lithium-Ionen
Tesla	AC Propulsion (US)	Lithium-Ionen
Toyota	Panasonic (Japan)	Nickel-Metallhydrid, Lithium-Ionen
Volkswagen	Sanvo. Toshiba (Japan)	Lithium-Ionen

Tabelle 2.1: Kooperationen zwischen Autohersteller und Batteriehersteller, stand 2009

Tabelle 2.1 zeigt die bestehenden Kooperationen zwischen Batterieherstellern und den Automobilherstellern. Es zeigt einen Auszug aus einem Artikel von „Auto Motor und Sport“ [11]. Über die Testverfahren der Zulieferer der Automobilindustrie ist wenig bekannt.

Außer den Automobilherstellern und deren Zulieferern gibt es noch die externen Dienstleistungsunternehmen. Das sind Firmen wie Intertek, TÜV NORD zusammen mit CETECOM, TÜV SÜD und SGS Germany GmbH. Drei dieser Firmen haben erst vor Kurzem in ein neues Batterieprüflabor investiert. Aus deren Werbung ist ersichtlich, welche Tests sie an Batterien durchführen.

Leistung	Beschreibung
Lebensdauerprüfung	• zyklische Alterung • kalendarische Alterung • Alterung von Batterien unter wechselnden Umwelteinflüssen
Abuse-Prüfungungen Beschreibung: Simulation extremer Umwelteinflüsse die bis an die Grenzen der Belastbarkeit gehen	• Nageltest • Elektrische Tests, z.B. Überladung bzw. Tiefentladung • Kurzschlusstests • Thermische Belastungstests • Zyklische Belastungstests ohne aktive Kühlung • Fallprüfungen • Brandtests (Kraftstofffeuer) • Immersionstests • Sicherheitstests mit Gasanalyse • Höhenlage • Vibration
Leistungsprüfung	• Erstellen der Datenblätter • Belastungsgrenzen ermitteln • Nennwerte bestimmen
Umweltprüfungen/mechanische Belastungsprüfungen	• EMV Prüfungen • Schwingungs-, Stoß- und Temperaturwechselprüfungen • Beständigkeit gegen Korrosion, Staub, Salz und Feuchtigkeit
Crashtest für die Elektromobilität Beschreibung: Simulation realer Unfallszenarien	• Stoßkörper wird mit Hilfe eines Trägerfahrzeugs gegen die starr befestigte Hochvolt-Batterie gefahren • Batterie wird mit Hilfe eines Trägerfahrzeugs gegen den starr befestigten Stoßkörper gefahren • Quetschversuche am Prüfling • Batterie wird auf eine Schlittenanlage montiert und gegen eine Barriere gecrasht • Impaktor wird mit Hilfe eines Fallturms(10m) auf die Batterie gestoßen
Benutzungs- und Transport-Prüfungen nach internationalen Normen	• BATSO 01 • ECE R100 • DOE/ID-11069 • DOE/ID-11173 • IEC 1725 • IEC 61508 • IEC 62660-1/2 • IEC 62133 • IEEE 1625 • ISO 12405-1/2 • ISO 26262 • JIS D5305 • SAE J2464 • SAE J2929 • SAND 2005-3123 • SBA S1101 • UL 1642 • UL 2054 • UN 38.3 • QC/T 743

Tabelle 2.2: Dienstleistungen der Batterie-Prüflabore

Tabelle 2.2 zeigt eine Zusammenfassung aller durchgeführten Test von TÜV SÜD [12] und SGS Germany GmbH [13]. Viele dieser Tests sind irreversibel und bedeuten das Ende der Zelle.

2.1.1 NEFZ

Der „NEFZ - Neuer Europäische Fahrzyklus“ dient in Europa zur Ermittlung des Kraftstoffverbrauchs und der CO_2-Emission. Er besteht seit 1992, als die Richtlinie 70/220/EWG, durch die Richtlinie 91/441/EWG [14] ergänzt wurde. Diesen Richtlinien unterliegen alle PKWs in Europa, unabhängig vom Antrieb (Verbrennungsmotor, Elektromotor, Hybrid).

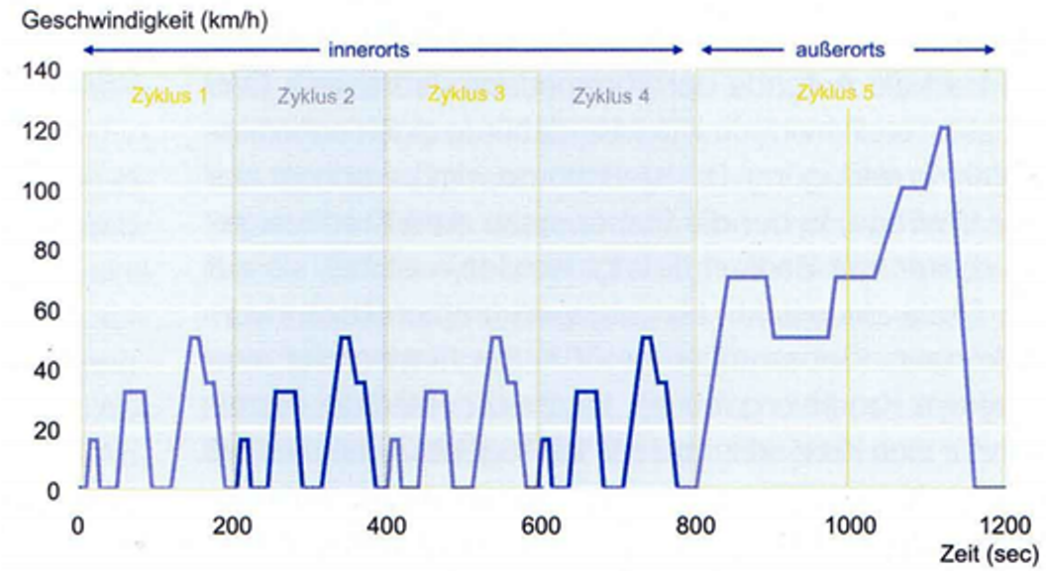


Abbildung 2.1: der Neue Europäische Fahrzyklus

In Abbildung 2.1 ist der NEFZ abgebildet. Er besteht aus vier Zyklen innerorts und einem Zyklus außerorts. Die Richtlinie 70/220/EWG wird ständig ergänzt und angepasst. Sie enthält nicht nur den Fahrzyklus, sondern auch die Prüfungsbedingungen und die Berechnungsvorschriften für die zu ermittelnden Werte. Diese Prüfung dürfen nur zertifizierte EG-Prüflaboratorien durchführen. Die Richtlinie lässt der Automobilindustrie einen gewissen Spielraum in der Durchführung der Messung zu. Eine Studie von T&E [15] hat dazu einige Nachforschungen angestellt.

Kritische Punkte bei der Messung nach der Studie von T&E:

- Anpassung der Motorsteuerung
- Abzug der 4%igen Toleranz von den Messwerten
- Verwendung des minimalen Fahrzeuggewichtes
- Abkleben von Kanten
- höherer Reifenluftdruck

Die Kraftstoffverbrauchs- und die CO_2-Emissions Werte sind beim Autoverkauf in Europa mit anzugeben. Unternehmen wie die „DAT - Deutsche Automobil Treuhand GmbH“ sammeln diese Werte und veröffentlichen sie regelmäßig, beispielsweise im „DAT-Leitfaden“ [16].

Modell	Leistung	Kraftstoff	Fahrzeug-masse	Kraftstoff- Stromverbrauch			CO2-Emission	CO2-Effizienz
				innerorts	außerorts	kombiniert	kombiniert	
	[kW]		[kg]	[l/100km] oder [kg/100km] nach (EG) Nr. 715/2007 oder [kWh/100 km]			[g/km]	
Strom								
Renault Twizy 45	4	E	548			5,8	0	A+
Renault Twizy	13	E	562			6,3	0	A+
VW e-up!	60	E	1214			11,7	0	A+
Citroen C-Zero	49	E	1140			12,6	0	A+
Peugeot iOn	49	E	1140			12,6	0	A+
Volkswagen e-Golf	85	E	1585			12,7	0	A+
BMW i3	75	E	1270			12,9	0	A+
Mitsubishi Electric Vehicle (I-MiEV)	49	E	1185			13,5	0	A+
Renault Fluence Z.E.	70	E	1605			14,0	0	A+
Renault Zoe	65	E	1503			14,6	0	A+
Nissan NEW LEAF	80	E	1505			15,0	0	A+

Tabelle 2.3: Auszug aus dem „DAT-Leitfaden“

Tabelle 2.3 ist ein Auszug aus dem aktuellen „DAT-Leitfaden“. Es zeigt alle aufgelisteten Elektroautos, wie man sieht haben sie keine CO_2-Emission. Der „Kraftstoffverbrauch“ wird immer in kWh/100km angegeben. Diesen Verbrauchsangaben wurden mithilfe des NEFZ ermittelt.

2.2 ADC-Problematik

Beim Umgang mit dem Zykliersystem hat sich heraus gestellt, dass der ADC eine Schwachstelle ist. Die Problematik ist hierbei, über einen großen Messbereich genaue Messungen im Millivoltbereich durchzuführen und gleichzeitig den ADC gegen Überspannung, HF-Signale und NF-Signale zu schützen. Bei jeder Reparatur eines defekten ADC leidet das Board und es geht viel Zeit verloren.

2.2.1 Besonderheiten und Charakteristik des ADC

Der verwendete ADC ist ein AD7798 von „Analog Devices“. Er arbeitet nach dem Delta-Sigma Verfahren mit einer Auflösung von 16 Bit und einer maximalen Update Rate von 470 Hz. Bei Delta-Sigma ADCs sind Auflösungen von 12 bis 24 Bit und Update Raten von 2Hz bis 80kHz üblich, somit liegt er im Mittelfeld was Schnelligkeit und Auflösung angeht. Das besondere an diesem IC ist die Peripherie.

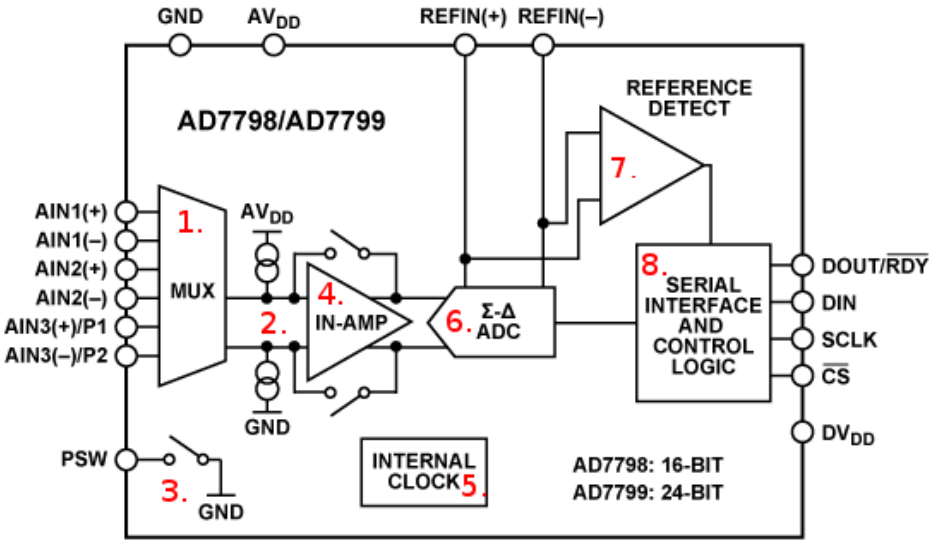


Abbildung 2.2: Funktionsblockschaltbild des AD7798

Abbildung 2.2 zeigt das Blockschaltbild aus dem Datenblatt [17]. An diesem Blockschaltbild kann man gut die Funktionen des ADCs erkennen.

Die Peripherie des AD7798:

1. **Multiplexer** - Er ermöglicht das Messen von drei differentiellen Eingängen, wobei der dritte Eingang auch als Digitalausgang P1 und P2 verwendet werden kann.
2. **zwei 100nA Stromquellen** - Damit lässt sich feststellen ob sich ein Sensor am Eingang des ADCs befindet. Angenommen die Stromquellen werden aktiviert und der ADC führt eine Messung durch. Ergibt sich nun ein Vollausschlag, ist die Leitung offen.
3. **Power Schalter** - Wird GND vom Sensor mit dem Schalter verbunden, so lässt sich die Stromzufuhr zum Sensor durch den Schalter unterbrechen. Damit lässt sich in der Zeit, in der nicht gemessen wird, Strom sparen.
4. **Instrumentenverstärker** - Zur Verstärkung der analogen Eingangsspannung und der Verringerung des Eingangsstroms durch einen höheren Eingangswiderstand.
5. **Interner Oszillator** - Dient als Takt für die digitalen Bauteile. Ein externer Oszillator wird somit überflüssig.
6. **Digitales Filter** - Im Delta-Sigma ADC ist ein digitales Filter enthalten, der zur Rauschunterdrückung dient.
7. **Referenzspannung Detektor** - Um sicher zu stellen, dass eine geeignete Referenzspannung anliegt, wird im Sekundentakt überprüft ob REFIN(+) und REFIN(-) größer ist als 0,3V.
8. **Serielle Schnittstelle** - Zur Kommunikation zwischen µC und ADC. Diese Schnittstelle verfügt über eine eigene Spannungsversorgung (DV_{DD}).

Der Multiplexer, der Power Schalter, der Referenzspannung Detektor und die zwei 100nA Stromquellen lassen sich ohne Einschränkungen benutzen. Sie werden durch die Register einfach ein- und ausgeschaltet. Der interne Oszillator und die serielle Schnittstelle sind ständig im Betrieb. Lediglich der Instrumentenverstärker und das digitale Filter haben ihre Besonderheiten.

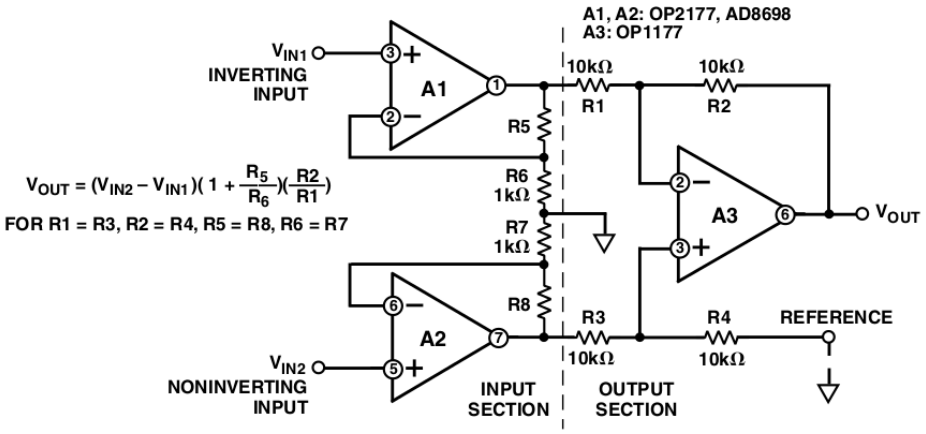


Abbildung 2.3: Allgemeiner Aufbau des internen Instrumentenverstärkers

Im Datenblatt des ADCs erhält man keine Informationen über den Aufbau des Instrumentenverstärkers. Aber im „Designers Guide to Instrumentation Amplifiers - Chapter II“ [18] von Analog Devices erhält man allgemeine Informationen zum Aufbau. Abbildung 2.3 ist ein Auszug aus der genannten „Applikation Note“. Die „INPUT SECTION“ ist der sogenannte Buffer bestehend aus A1 (invertierter Eingang) und A2 (nicht invertierter Eingang). Die „OUTPUT SECTION“ ist ein Subtrahierer, bestehend aus A3. Zusammen bilden sie den Instrumentenverstärker. Mit den Widerständen R5, R6, R7 und R8 lassen sich die Verstärkung einstellen. Der Instrumentenverstärker im AD7798 hat eine programmierbare Verstärkung von 1, 2, 4, 8, 16, 32, 64 oder 128. Bei einer Verstärkung von 1 oder 2 ist die „OUTPUT SECTION“ abgeschaltet. Das heißt, es kann bei einer Verstärkung von 1 oder 2 nur der Buffer ein- oder ausschalten werden. Bei einer Verstärkung von 4 oder höher schaltet sich der gesamte Instrumentenverstärker automatisch ein. Im Funktionsblockschaltbild Abbildung 2.2 kommt man schnell zum Trugschluss, dass sich der Instrumentenverstärker beliebig ein- oder ausschalten lässt.

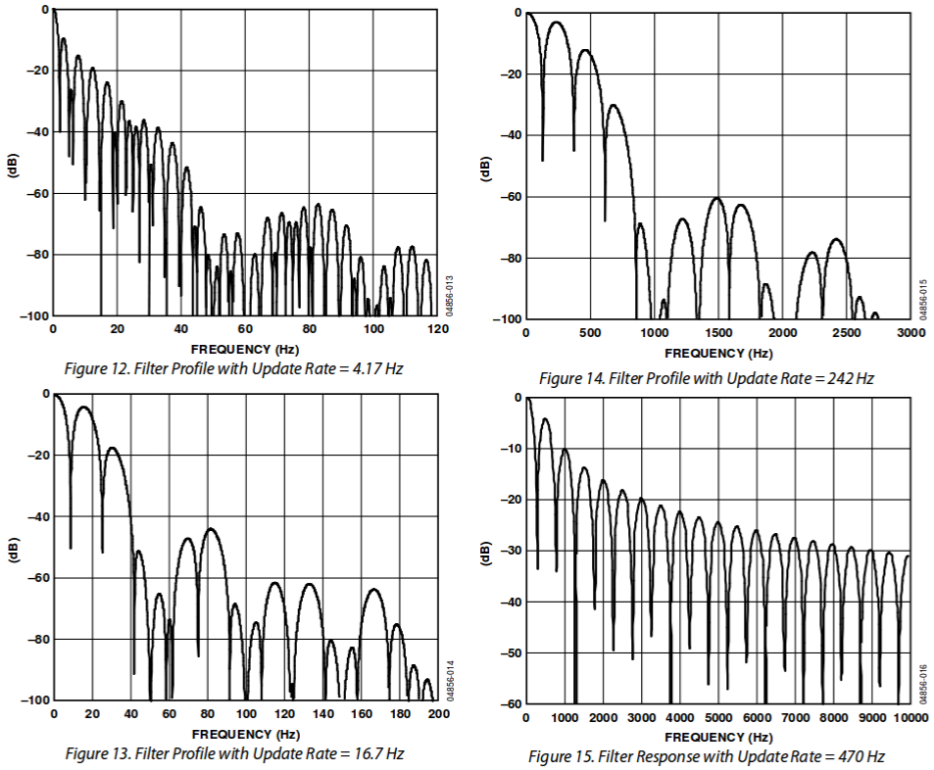


Abbildung 2.4: Zusammenhang zwischen der „Update Rate“ und dem Amplitudengang des digitalen Filters

Beim ADC kann kann zwischen 15 unterschiedlichen Update Raten gewählt werden, von minimal 4,17Hz bis maximal 470Hz. Jede dieser 15 Update Raten hat einen eigenen Amplitudengang. Abbildung 2.4 ist ein Auszug aus dem Datenblatt des ADCs [17] und zeigt für vier Update Raten den dazugehörigen Amplitudengang. Bei den niedrigen Update Raten unterdrückt man zusätzlich das Netzbrummen, beispielsweise hat man bei einer Update Rate von 16,7Hz eine 80dB Unterdrückung bei 50Hz.

2.2.2 Untersuchung der bestehenden ADC Schaltung

Der verwendete ADC ist ein AD7798 mit drei differentiellen Eingängen, die durch einen internen Multiplexer ausgewählt werden können. Der zweite und dritte Eingang wird benutzt, um den Strom der Batterie über eine Hall Sonde zu messen. Dies funktioniert reibungslos. Der erste Eingang misst die Spannung der einzelnen Zellen.

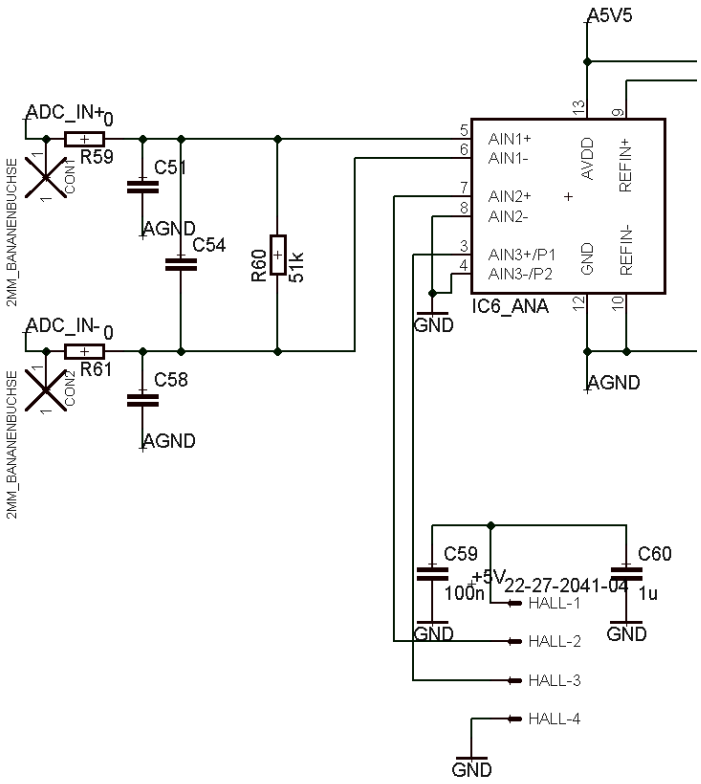


Abbildung 2.5: Auszug aus dem EAGLE Schaltplan - ADC Eingänge

Vergleicht man den Schaltplan von EAGLE, in Abbildung 2.5, mit der bestehenden Platine, dann fällt auf, dass die Kondensatoren C51, C54 und C58 am Eingang 1 fehlen. Wahrscheinlich handelt es sich hierbei um einen optionalen analogen Tiefpass. Da er nicht gebraucht wird, sind die Kondensatoren entfernt worden und die Widerstände R59 und R61 überbrückt.

Um die verwendeten Einstellungen der bestehenden ADC Schaltung zu finden, werden die Register und deren Konfiguration untersucht.

RS2	RS1	RS0	Register	Register Size
0	0	0	Communication register during a write operation	8 bits
0	0	0	Status register during a read operation	8 bits
0	0	1	Mode register	16 bits
0	1	0	Configuration register	16 bits
0	1	1	Data register	16 bits (AD7798)/24 bits (AD7799)
1	0	0	ID register	8 bits
1	0	1	IO register	8 bits
1	1	0	Offset register	16 bits (AD7798)/24 bits (AD7799)
1	1	1	Full-scale register	16 bits (AD7798)/24 bits (AD7799)

Tabelle 2.4: Registertabelle des ADCs

Tabelle 2.4 ist ein Auszug aus dem Datenblatt des AD7798 [17, S.13]. Diese Tabelle bietet eine Übersicht über alle verfügbaren Register. Zur Einstellung einer Messung sind nur das Mode- und Konfigurationsregister von Interesse. Bei den anderen sieben Registern wird nur kurz deren Aufgabe erklärt:

- **Kommunikationsregister** - dient zur Ansteuerung der einzelnen Register.
- **Statusregister** - gibt den Status des ADCs an, z.B. welcher der drei Eingänge gewandelt wird oder ob ein Fehler vorliegt.
- **Datenregister** - enthält die Daten einer Messung.
- **ID Register** - enthält die Seriennummer des ADCs.
- **IO Register** - Eingang 3 lässt sich als digitaler Ausgang verwenden.
- **Offset und Full-Scale Register** - dient zur Kalibrierung des ADCs.

Mit dem Moderegister lässt sich die Update Rate und der Arbeitsmodus einstellen und der „Power Schalter“ schalten. Als Arbeitsmodi existieren die kontinuierliche Umwandlung, die einzelne Umwandlung, ein Idle Modus, ein Energiesparmodus und drei weitere Kalibrierungsmodi. Die Ausgangseinstellung benutzen als Modus die einzelne Umwandlung von analogen Werten bei einer Update Rate von 123 Hz.

Im Konfigurationsregister lässt sich der „buffered mode“, die zwei Stromquellen und der Referenzspannungsdetektor ein- und ausschalten. Des Weiteren kann vom Unipolarbetrieb in den Bipolarbetrieb gewechselt werden, die Verstärkung bestimmt werden und der Multiplexer geschaltet werden. In unserem Fall arbeiten wir im Unipolarbetrieb mit einer Verstärkung von eins im „unbuffered mode“. Die zwei Stromquellen und der Referenzspannungsdetektor werden nicht benutzt.

Zusammenfassung:
Es wird im Einzel-Umwandlungsmodus mit einer Update Rate von 123Hz gearbeitet. Das bedeutet, dass alle 8,13 ms eine neue Umwandlung stattfindet. Die analogen Werte werden mit einer Verstärkung von eins ohne Buffer oder Instrumentenverstärker auf den Delta-Sigma Wandler gegeben. Funktion wie die zwei Stromquellen und der Referenzspannung Detektor sind ausgeschaltet. Die Anforderungen an das analoge Eingangssignal sind durch den Unipolarbetrieb und AV_{DD} festgelegt. Es muss eine Spannung von -30mV und 5,43V anliegen.

2.2.3 Fehlerursache

Der häufige Ausfall des ADCs muss Ursachen haben. Diesen gilt es auf den Grund zu gehen. Zwei mögliche Ursachen werden vorgestellt.

1. **Instabile Referenzspannung** - Der ADC erhält seine Referenzspannung über den ADR4550 [19]. Dies ist eine hoch präzise Referenzspannungsreglung mit einer Ausgangsspannung von 5V. Als Eingangsspannung benötigt er die 5V + die „Dropout Voltage“ von 300mV. Das heißt, dass die Eingangsspannung immer größer als 5,3V sein muss. Der Referenzspannungsregler wird in unserem Fall mit gemessenen 5,4V versorgt. Wenn nun kurzzeitig viel Strom benötigt wird, wie zum Beispiel von den Lastrelais, dann bricht die Spannung kurz ein und hat zur Folge, dass die Referenzspannung ebenfalls davon beeinflusst wird.
2. **Überspannung** - Der ADC hat eine klar definierte Spannungsgrenze von 5,43V. Durch das überschreiten des Wertes zerstört man den ADC. Der Grund dafür kann eine defekte Zelle sein, falsch angeschlossene Zellen oder auch eine zu hoch eingestellte Ladespannung.

2.3 Schnittstellen und Dokumentation

Ein Teil dieser Thesis beschäftigt sich mit den Schnittstellen des Zykliersystems und der Aufbereitung der Dokumentation. Da es vor dieser Arbeit schon zwei Thesen gab, die sich mit dem Zykliersystem beschäftigten, wird eine schlichte Bestandsaufnahme durchgeführt.

Eine umfangreiche Dokumentation zum Zykliersystem existiert bereits in Form von zwei Bachelorthesen von Herrn Wisniewski [1] und Herrn Röhn [7]. Diese sind aber sehr umfangreich und bieten keine schnelle Hilfe. Zudem gibt es eine Kurzanleitung zur Bedienung des Zykliersystems [8].

Die LAN Schnittstelle ist in einem einfachen privaten lokalen Netzwerk einsatzbereit, nachdem man manuell dem Gerät eine IP-Adresse zugewiesen hat. Doch im lokalen HAW-Netzwerk funktioniert das Gerät mit DHCP noch nicht. Das muss untersucht werden.

Die RS232 Schnittstelle wurde noch nie benutzt und deshalb wurde sie in der Programmierung auch noch nie berücksichtigt. Sie wurde durch das Entfernen des Jumpers für die Stromversorgung des Leitungstreibers deaktiviert. Des Weiteren birgt sie die Gefahr, die galvanische Trennung des Zykliersystems aufzuheben, da Host und Device der Schnittstelle eine gemeinsame Masse haben.

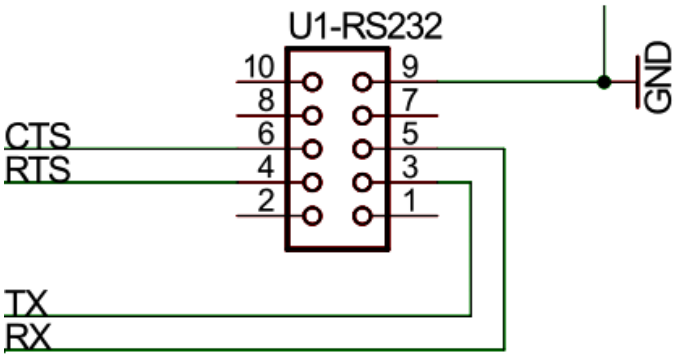


Abbildung 2.6: Auszug aus dem EAGLE Schaltplan - RS232

Die Pinbelegung des 10-Pin-Wannenstecker entspricht nicht der RS232 Norm, siehe Abbildung 2.6. Die zwei existierenden Pinbelegung für die RS232 Norm sind DTE für die Host Seite und DCE für die Device Seite. Dabei liegen TX/RX immer auf Pin 2 und 3 , GND auf 5 und CTS/RTS auf Pin 7 und 8.

3 Redesign der ADC-Schaltung

Da die Ursachen des häufigen Ausfalls des ADCs aus Kapitel 2.2.3 bekannt sind, können diese behoben werden. Es wird eine separate Platine entworfen, auf der der ADC mit Präzisionsspannungsregler und die ADC-Schutzschaltung Platz finden. Diese Platine wird durch ein 10 poliges Flachbandkabel mit der Hauptplatine verbunden. Sie löst die bestehende ADC Schaltung auf der Hauptplatine ab. Die neue Schaltung soll die gefundenen Ursachen beheben, einen leichten Austausch des ADCs ermöglichen und Messungen, mit einer Genauigkeit von 1mV erlauben.

Der Ablauf sieht so aus, dass zunächst verschiedene Schutzschaltungen auf einem Testboard ausprobiert und anschließend auf ihre Genauigkeit hin überprüft werden. Am Ende dieser Versuche soll die optimale Schutzschaltung gefunden sein und es geht mit der Platinenentwicklung weiter.

3.1 Schutzschaltung

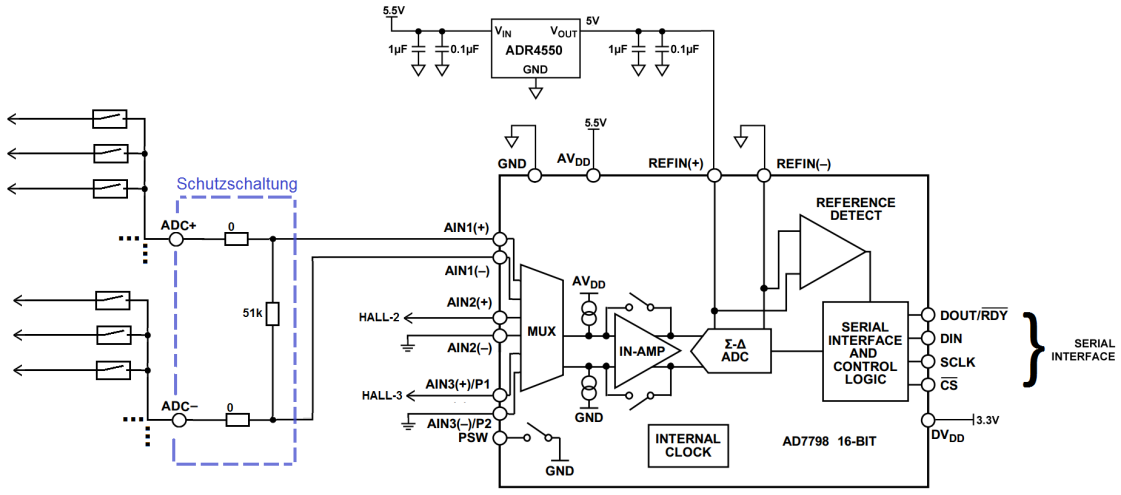


Abbildung 3.1: Blockschaltbild der ADC-Ausgangsschaltung

Abbildung 3.1 zeigt die bestehende ADC-Schaltung. Das gestrichelte Kästchen deutet den zukünftigen Platz der ADC-Schutzschaltung an. An dieser Stelle werden jetzt verschiedene Schutzschaltungen ausprobiert.

3.1.1 Supressordiode

Supressordioden werden zum Abfangen von Spannungsimpulsen eingesetzt. Sie beginnen bei einer bestimmten Schwelle leitend zu werden bis sie schmelzen und einen Kurzschluss herbeiführen. Für unsere Schaltung benötigen wir eine unipolare Supressordiode, die ab 5V anfängt, leitend zu werden. Die Entscheidung fiel hier auf die „SA5.0A“ [20] von Littelfuse. Sie kann bis zu 500W aufnehmen, bevor sie irreversibel schmilzt.

Des Weiteren wird R60, der $51 k\Omega$ Widerstand, der den Eingangswiderstand des ADC mindert, entfernt. Hinzu kommen zwei 56Ω Eingangswiderstände, die im Falle eines Kurzschlusses durch die Supressordiode den Strom begrenzen. Diese zwei Widerstände verursachen nur einen kleinen Offset-Fehler, den man durch die Kalibrierung ausgleichen kann.

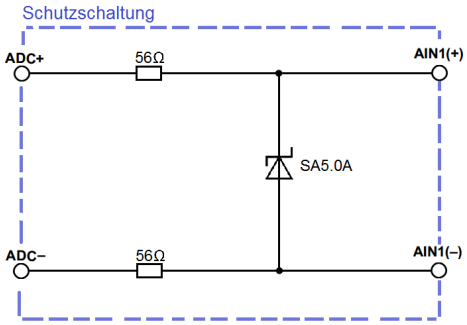


Abbildung 3.2: Schutzschaltung mit einer Supressordiode

In Abbildung 3.2 ist die beschriebene Schutzschaltung dargestellt. Sie besitzt einen Überspannungsschutz und sie hat einen höheren Eingangswiderstand, der den Einfluss der Messleitung minimiert.

3.1.2 Externe Instrumentenverstärker

Instrumentenverstärker werden auch als In-Amp bezeichnet, da sie im Englischen „Instrumentation Amplifier“ genannt werden. Die Supressordiode bietet keinen vollständigen Überspannungsschutz, da in Sperrsichtung die Durchbruchspannung bei 0,5V liegt und in Durchlassrichtung die Begrenzungsspannung „clamping voltage“ 9,2V beträgt. Das bedeutet, der analoge ADC Eingang kann Werte von -0,5V bis 9,2V annehmen. Zur genauen Begrenzung von 0V bis 5V ist ein Instrumentenverstärker ideal, da er hohe Spannungen am Eingang auf +VS und -VS begrenzt. Zur Erklärung +VS und -VS sind die positive und negative Versorgungsspannung des Instrumentenverstärker. Da der Eingangswiderstand des Instrumentenverstärkers im GΩ Bereich liegt, fließt ein sehr geringer Strom. Dieser wird durch äußere Störungen schnell verfälscht, da wir oft einen langen Weg zwischen der Zelle und dem Instrumentenverstärker haben. Deshalb wird vor dem Eingang des Instrumentenverstärker ein 4MΩ Widerstand geschaltet. Bei den Instrumentenverstärker fiel meine Wahl auf den AD623, AD8226 und den AD624 von „Analog Devices“.

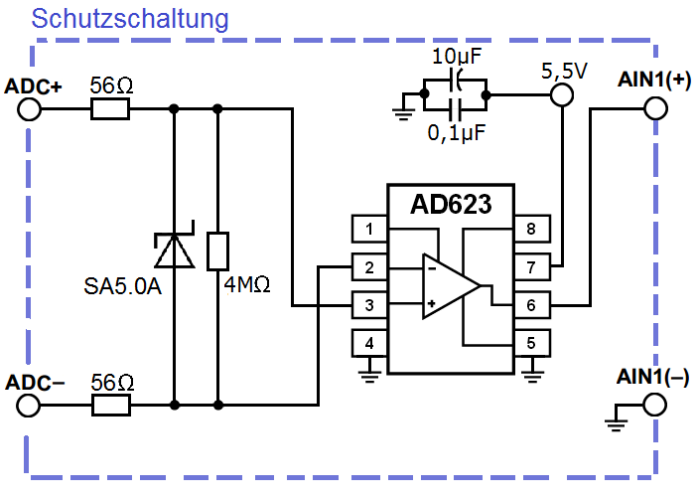


Abbildung 3.3: Schutzschaltung mit dem AD623

Abbildung 3.3 zeigt die Schutzschaltung mit dem AD623 [21]. Hierbei handelt es sich um ein Low Cost In-Amp. Er besitzt eine Bandbreite von 700 kHz und eine Gleichtaktunterdrückung (CMRR) von 70dB.

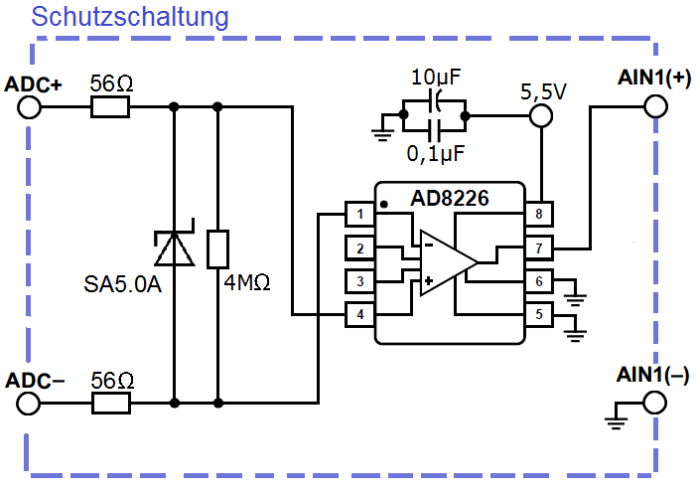


Abbildung 3.4: Schutzschaltung mit dem AD8226

Die aufgebaute Schutzschaltung mit dem AD8226 [22] ist in Abbildung 3.4 dargestellt. Es ist ein weiterer Low Cost In-Amp. Im Gegensatz zum AD623 besitzt der AD8226 eine um 20dB höhere Gleichtaktunterdrückung (CMRR) und sowie eine um 700kHz höhere Bandbreite. Zusätzlich sind die Eingänge vom AD8226 gegen elektrostatische Aufladungen (ESD) und Überspannung geschützt.

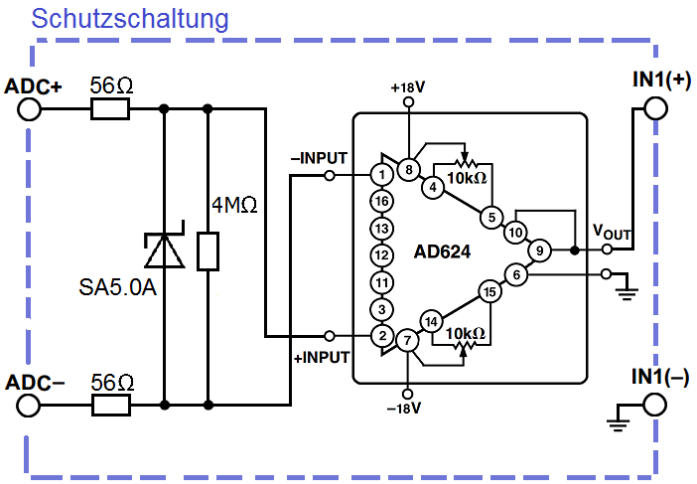


Abbildung 3.5: Schutzschaltung mit dem AD624

In Abbildung 3.5 ist die Schutzschaltung mit dem AD624 [23] dargestellt. Hierbei handelt es sich um einen teuren hoch präzisen In-Amp. Er unterscheidet sich von den Preiswerten Instrumentenverstärkern durch sein geringes Rauschen und seiner exakt linearen Übertragungsfunktion.

3.1.3 Interne Instrumentenverstärker

Da der interne Instrumentenverstärker erst ab einer Verstärkung von vier aktivierbar ist, wird die Spannung am Eingang des ADCs durch einen Spannungsteiler reduziert. Dass sich die Spannung durch thermischen Einfluss verändert, ist nicht zu erwarten. Da das Verhältnis der Widerstände immer gleich bleibt, ist auch ein Drift nicht zu befürchten. Aber durch das Teilen der Spannung und anschließender Verstärkung geht die Genauigkeit verloren. Die Frage die sich stellt ist, ob der interne Instrumentenverstärker trotzdem besser auf den ADC abgestimmt ist und dadurch bessere Ergebnisse liefert.

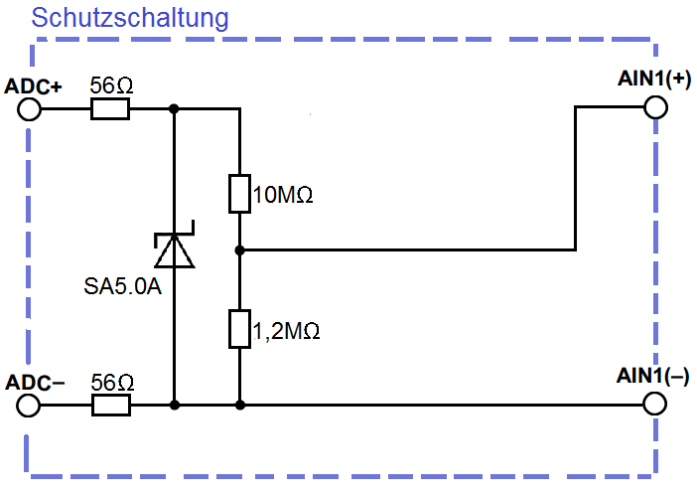


Abbildung 3.6: Schutzschaltung mit einem Spannungsteiler

Abbildung 3.6 zeigt die verwendete Schutzschaltung. Das Verhältnis der Widerstände von einem 10MΩ zu einem 1,2MΩ beträgt:

$$U_{AIN1} = U_{ADC} \cdot \frac{1,2}{11,2} = 0,107 \cdot U_{ADC} \tag{3.1a}$$

Wird das Verhältnis mit 4 verstärkt erhält man:

$$U_{AIN1} = 4 \cdot 0,107 \cdot U_{ADC} = 0,429 \cdot U_{ADC} \tag{3.1b}$$

3.2 Messungen

Das Wichtigste bei den Schutzschaltungen ist, dass nach deren Integration die Genauigkeit von 1mV nicht verloren geht. Deshalb werden die verschiedenen Schutzschaltungen getestet. Zum Verifizieren werden zwei verschiedene Messungen durchgeführt.

Die Aufnahme der Übertragungskennlinie erfolgt mithilfe einer Spannungsquelle, ROHDE & SCHWARZ - NGT 20. Sie nimmt den Platz der Batteriezelle ein.

Die erste Messung ist eine stochastische Messung, die dazu dient, das Rauschen zu erfassen. Bei einem festen Spannungswert von 3,5V und einem festen f_{ADC}=16,7Hz werden 1000 Werte vom ADC aufgenommen.

Bei der zweiten Messung handelt es sich um das Ermitteln der Übertragungsfunktion. Am Eingang wird die Spannungsquelle in 250mV Schritten im Bereich von 0V bis 4V durchlaufen. Bei jedem dieser Schritte führt der ADC 20 Messungen durch und gibt diese über den UART aus. Diese 20 Messungen werden gemittelt und zu einem Wert zusammengefasst, um das mögliche Rauschen zu minimieren.

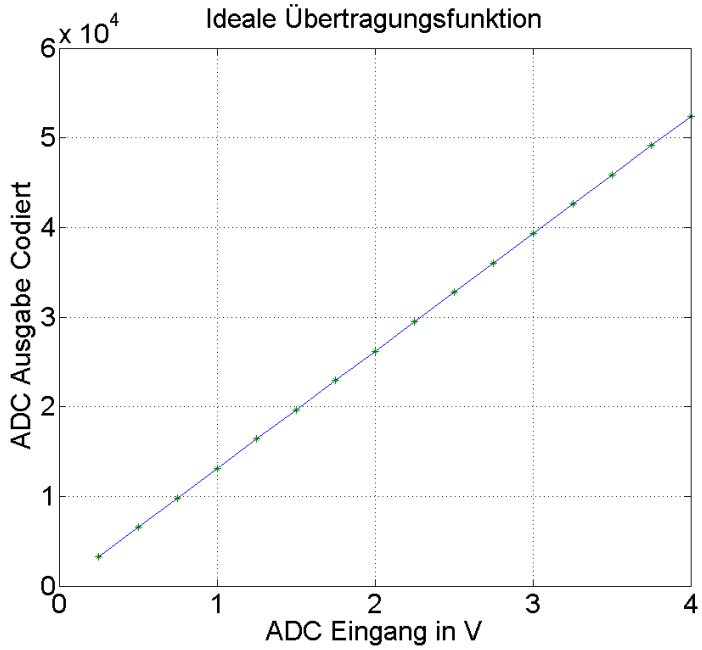


Abbildung 3.7: eine ideale Übertragungsfunktion

Abbildung 3.7 zeigt eine ideale Übertragungskennlinie. Auf der X-Achse ist die Eingangsspannung der Spannungsquelle abgebildet. Die Y-Achse ist die gemittelte UART Ausgabe

des ADCs. Diese ist codiert, die Umrechnung in eine Spannung ist wie folgt:

$$U_{AIN} = Code \cdot V_{REF} \cdot 2^{-Bit} \tag{3.2a}$$

Da wir den AD7798 mit 16 Bit verwenden und ihn mit einer 5V Referenzspannung versorgen ergibt sich:

$$U_{AIN} = Code \cdot 5V \cdot 2^{-16} = Code \cdot 76,29 \cdot \mu V \tag{3.2b}$$

Um aus der ermittelten Übertragungskennlinie auf die resultierende Abweichung zu kommen, muss die aufgenommene Kennlinie quadratisch oder linear kalibriert werden. Anschließend wird der Fehler durch die Differenz der idealen Kennlinie und der kalibrierten Kennlinie ermittelt.

3.2.1 Supressordiode

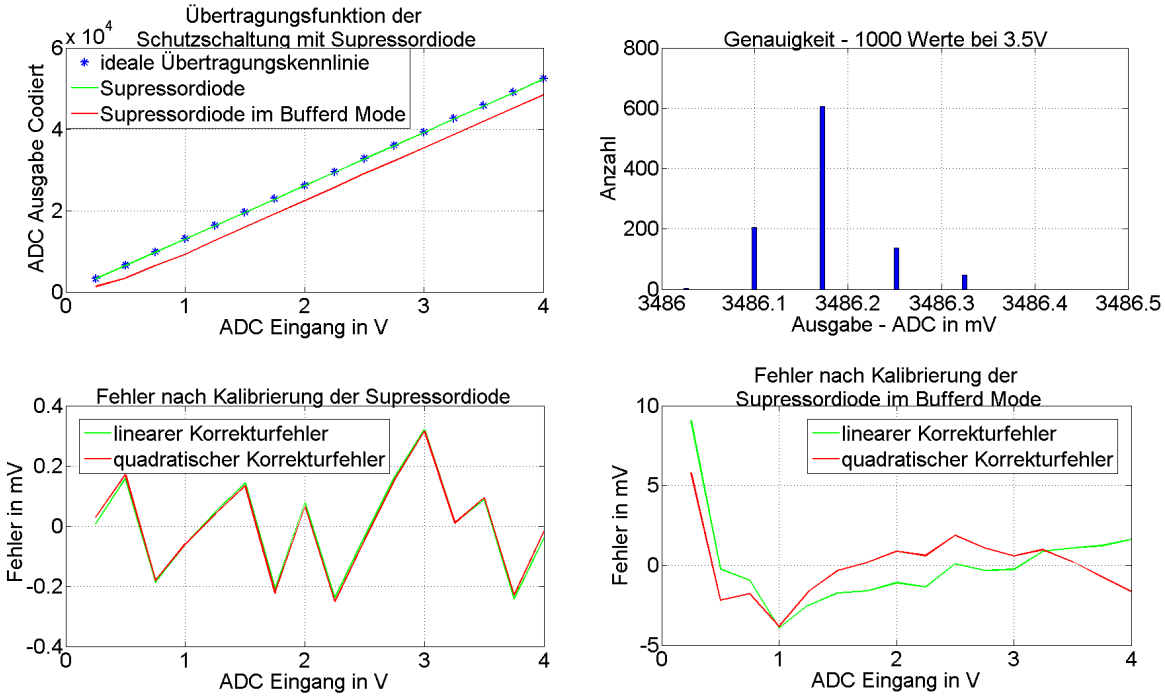


Abbildung 3.8: aufgenommene Messungen der Schutzschaltung mit der Supressordiode

In Abbildung 3.8 sind die Messungen der Schutzschaltung mit der Supressordiode dargestellt. Um den Eingangswiderstand des ADCs zu erhöhen, wurde der „buffered mode“ getestet. Bei der Ermittlung der Übertagungskennlinie und deren Korrektur fällt auf, dass der „buffered mode“ einen zu großen Fehler aufweist. Deshalb ist eine stochastische Auswertung unnötig. Die einfache Supressordiode ohne den „buffered mode“ rauscht nur gering und hat eine nahezu lineare Übertragungskennlinie.

3.2.2 Instrumentenverstärker

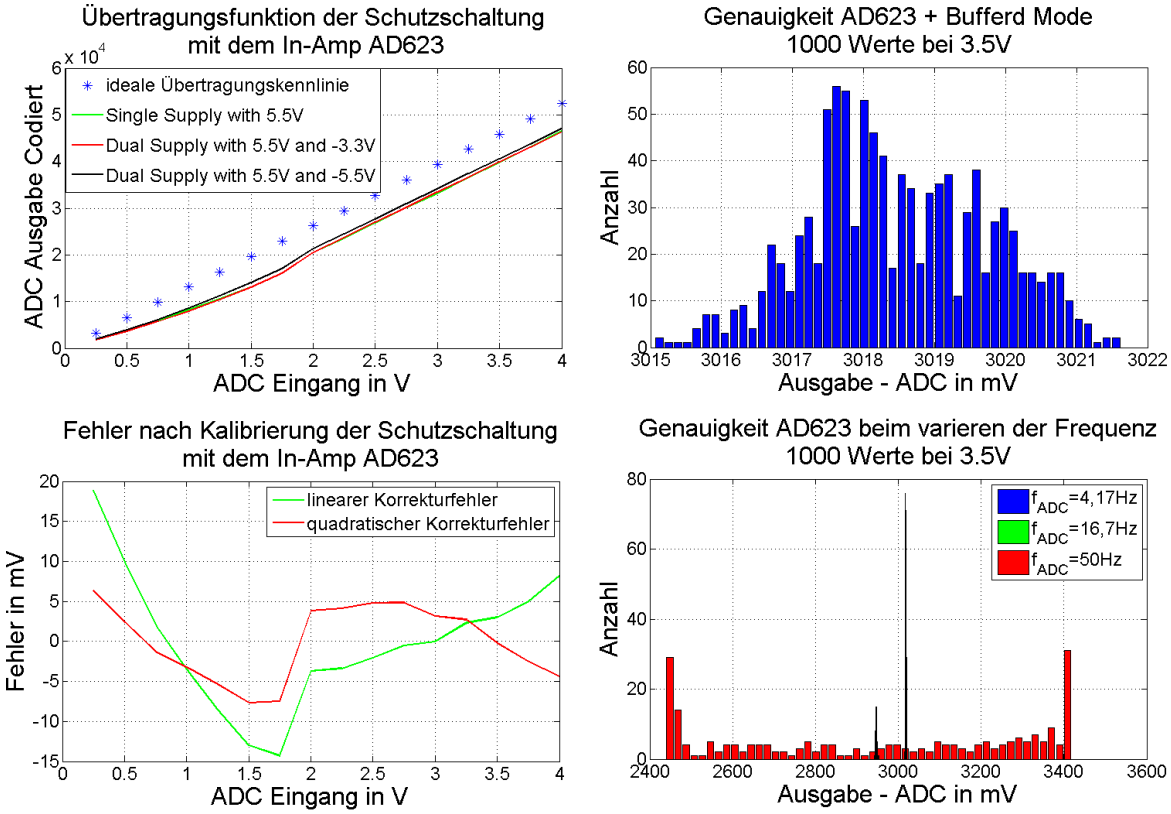


Abbildung 3.9: aufgenommene Messungen der Schutzschaltung mit dem AD623

Der erste externe In-Amp der überprüft wird, ist der AD623. Die Ergebnisse der Messung sind in Abbildung 3.9. Der AD623 kann mit einer Spannung versorgt werden, sprich „Single Supply“, bzw. zwei Spannungen, „Dual Supply“. Nach den Messungen wird klar, dass dies keine genaueren Ergebnisse liefert. Die Kennlinie sowie das Rauschen bleiben unverändert. Die Übertragungskennlinie ist weder linear noch quadratisch kalibriert brauchbar.

Bei den stochastischen Messungen wurde der „buffered mode“ geprüft. Die Messung beweist erneut das der „buffered mode“ ungeeignet ist, da die Messung um 7mV streut. Des Weiteren wurde die Genauigkeit bei veränderter f_{ADC} geprüft. Diese Messung wird kurz erläutert. Die Genauigkeitsmessung bei 4,17Hz (blau) ist der höchste Piek, sie ist um 4mV genau. Der linke schmale Piek ist die Messung bei einer Frequenz von 16,7Hz (grün) mit einer Genauigkeit von 6mV. Die dominante Messung mit 50Hz (rot) hat eine Genauigkeit von 977mV. Diese extreme Ungenauigkeit bei f_{ADC}=50Hz, entsteht, weil das digitale Filter fehlt.

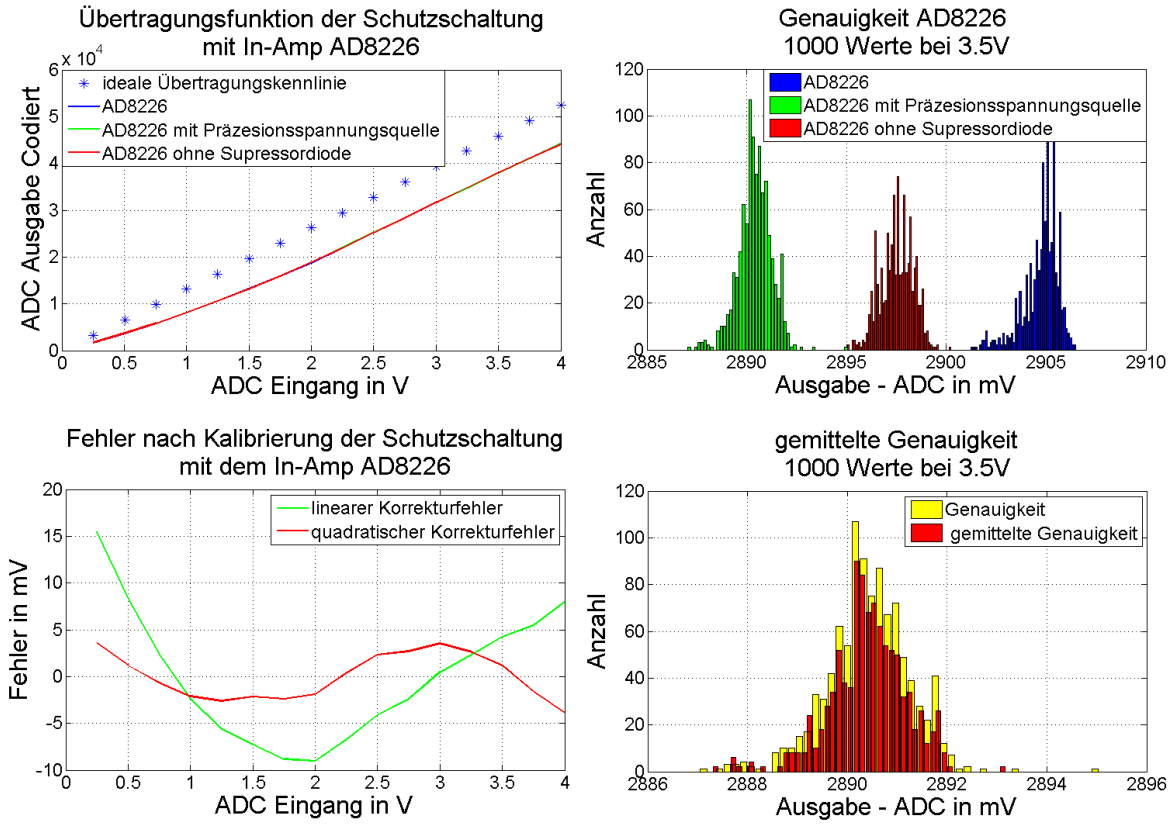


Abbildung 3.10: aufgenommene Messungen der Schutzschaltung mit dem AD8226

Abbildung 3.10 zeigt die Messung mit dem In-Amp AD8226. Bei diesen Messungen gibt es drei Varianten. Die erste Messung ist nur der AD8226 aktiv, bei der zweiten Messung wird der AD8226 mit einer Präzisionsspannungsquelle versorgt und bei der dritten Messung wurde auf die Supressordiode verzichtet.

Bei der Übertragungskennlinie ist zwischen den drei verschiedenen Messungen keine erwähnenswerte Veränderung zu erkennen. Diese Kennlinie lässt sich gut quadratisch kalibrieren. Die beste Genauigkeit wird bei der Messung mit der Präzisionsspannungsquelle erzielt. Diese hat um den Messwert die höchsten Ausschläge. Unten rechts wird versucht, die Genauigkeit durch eine numerische Mittlung zu erhöhen. Das Resultat ist eher ernüchternd. Das Positive an dieser Methode ist, dass die extremen Fehlmessungen verringert werden. Nachteilig ist, dass auch die bereits richtigen Messungen verringert werden.

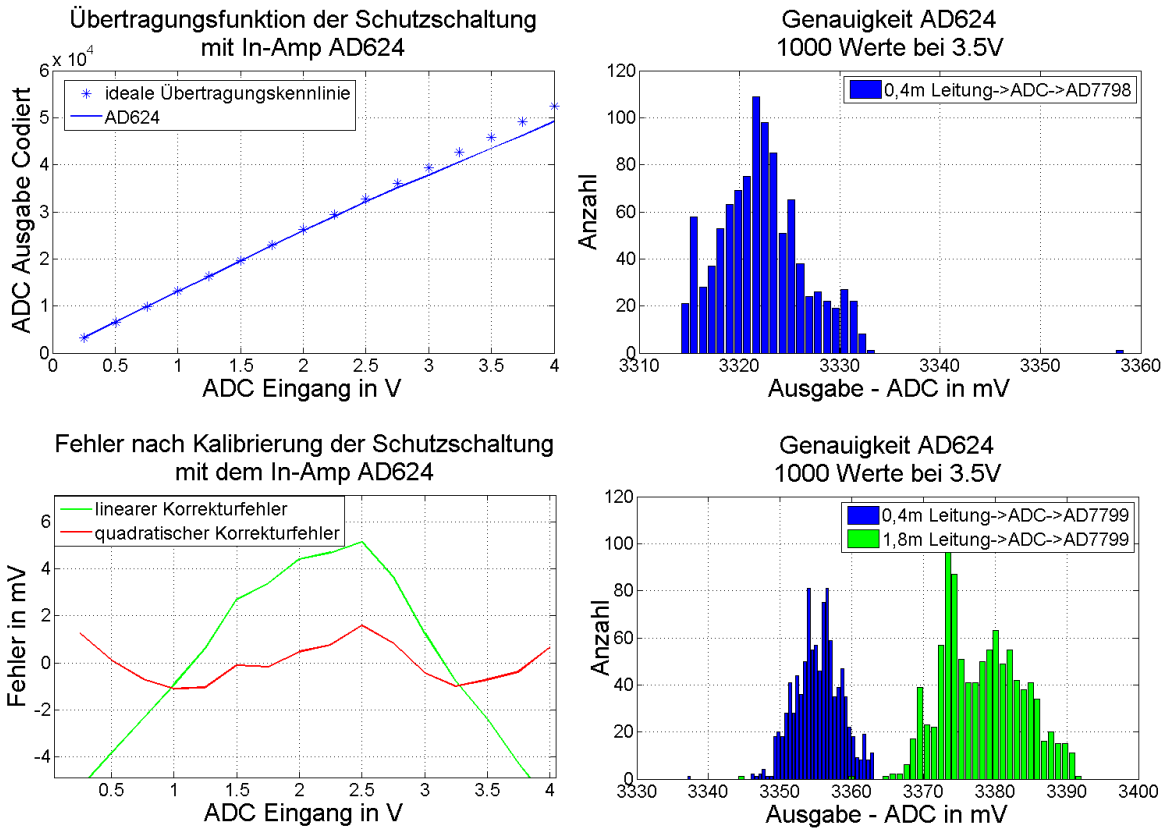


Abbildung 3.11: aufgenommene Messungen der Schutzschaltung mit dem AD624

Der letzte externe In-Amp, der geprüft wird, ist der AD624. Die Messung ist in Abbildung 3.11 dargestellt. Die Besonderheit ist bei diesem In-Amp, dass er über vier 9V-Blöcke seine eigene Stromversorgung hat. Da er ein eigenes Gehäuse besitzt, wird er über vier Messleitungen in die Schaltung integriert.

Die Übertragungskennlinie ist nahezu linear, nur im obigen Bereich hat diese eine leichte Steigungsänderung. Die Genauigkeit auf 1000 gemessenen Werten liegt bei 15mV. Des Weiteren wurde die Genauigkeit mit dem AD7799 verglichen. Er hat eine höhere Auflösung,24 Bit anstatt 16 Bit. Das Variieren des ADCs ist nicht bahnbrechend. Gut zu erkennen ist, der Einfluss von der Länge der Messleitungen auf die Genauigkeit. Bei Verwendung von kurzen Messleitungen gibt es eine Genauigkeit von 10mV.

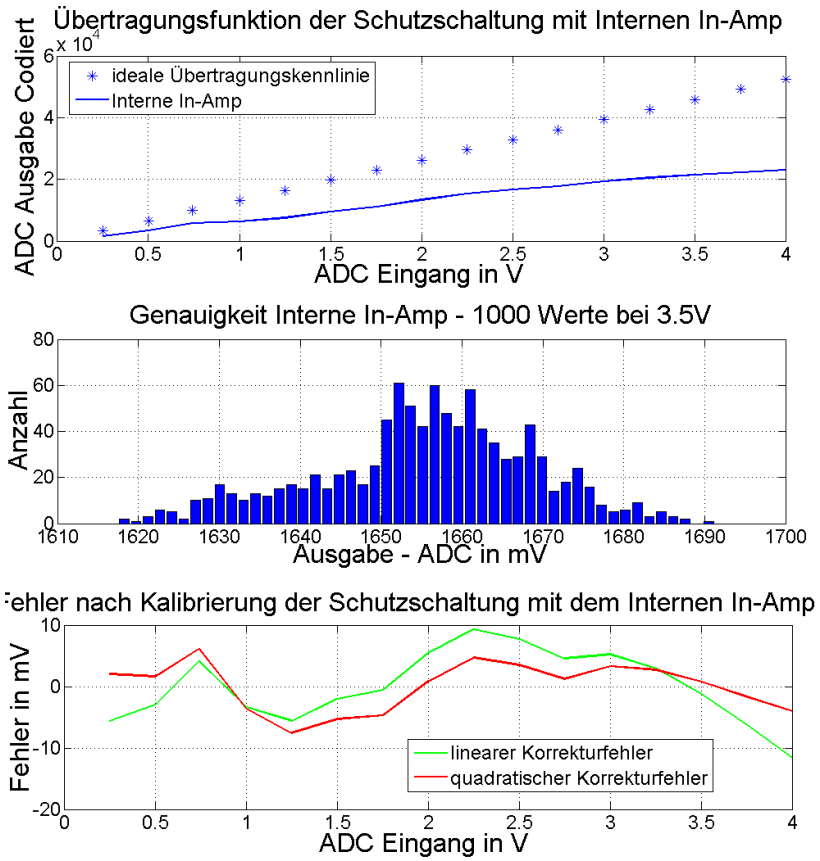


Abbildung 3.12: aufgenommene Messungen der Schutzschaltung mit dem internen In-Amp

Bei den Messergebnissen in Abbildung 3.12 handelt es sich um die des internen Instrumentenverstärkers des AD7798. Diese sind sehr unbefriedigend, da die Genauigkeit und die Übertragungskennlinie sehr schlecht sind.

3.3 Auswertung

Nun wird sich ein Überblick über die Messungen verschafft. Um den Gesamtfehler einer Messung zu erfassen, reicht es nicht aus, den Kalibrierfehler und die Genauigkeitsfehler zu addieren, da die Genauigkeitsfehler schon ein Teil des Kalibrierfehlers sind. Beim Aufzeichnen der Übertragungsfunktion ist die Ungenauigkeit der Messung ein ständiger Begleiter.

Messungen	Genauigkeitsfehler	Kalibrierfehler
Supressordiode	0,3 mV	0,3 mV
In-Amp - AD623	4mV	7mV
In-Amp - AD8226	4mV	4mV
In-Amp - AD624	17mV	1mV
In-Amp - Intern	60mV	7mV

Tabelle 3.1: Zusammenfassung der aufgenommenen Messwerte

In Tabelle 3.1 sind die verschiedenen Messungen mit den geringsten Fehlern dargestellt. Die Supressordiode ist nicht bedenklich und ist die beste Wahl. Alle weiteren Komponenten sind messtechnisch nicht mit der Genauigkeitsvorgabe, von einem Millivolt Auflösung, vereinbar. Bei den Instrumentenverstärkern erweist sich der AD8226, der mit einer Präzisionsspannungsquelle versorgt wird, als der Beste, da er einen relativ geringen Genauigkeitsfehler und Kalibrierfehler hat.

Der „buffered mode“ hat in allen Messungen schlechte Ergebnisse hervorgebracht. Das Varieren von f_{ADC} hingegen ist entscheidend für die Genauigkeit. Da bei hohen Frequenzen der digitale Filter im ADC deaktiviert wird, kann bei den Messfehlern auf äußere Störeinflüsse schließen. Es ist nicht auszuschließen, dass das Schaltnetzteil des Zykliersystem hochfrequente Störungen verursacht. Auch lange Leitungen sind zu vermeiden, da sie wie Antennen wirken.

Da das Ergebnis vom InAmp AD8226 nicht ausreichend genau ist, muss über eine Alternative nachgedacht werden. Auf der zu entwickelten Schutzschaltung muss der In-Amp optional verfügbar sein.

3.4 ADC Schaltungsentwurf

Basierend auf den gewonnen Messergebnissen der aufgebauten Schutzschaltungen wird jetzt die ADC Schutzschaltung entworfen. Um hochfrequente Störungen durch das Schaltnetzteil zu vermeiden, erhält die Schutzschaltung ein eigenes lineares Netzteil. Die Schutzschaltung soll zwei Betriebsmodi haben. Eine mit externen In-Amp, die andere ohne In-Amp. Die Supressordiode soll in beiden Modi enthalten sein. Um die große Bandbreite des In-Amp zu schmälern, kommt vor dem In-Amp noch ein analoger Tiefpass. Um weniger Störungen von der bereits existierenden Schaltung zu bekommen, wird die Schutzschaltung von dem Rest des Zykliersystems galvanisch getrennt.

Da in diesem Kapitel nur ausgewählte Teile der Schaltung behandelt werden, wird auf den Anhang B.1, B.2 und B.3 verwiesen. Dieser enthält den Schaltplan, das Layout und die Teileliste. Ansonsten sind die Eagle Projekt Dateien auch auf der CD zu finden.

3.4.1 Tiefpass

Um den Tiefpass korrekt zu dimensionieren wird eine Simulation durchgeführt. Dazu wird die Software „Qucs“ benutzt. Es ist eine freie Software die einfache und kleine Simulationen ermöglicht. Das Ziel ist eine Grenzfrequenz unter 50Hz zu erhalten, um mögliche Störeinflüsse durch das Netzbrummen zu filtern. Trotzdem sollen die Schaltungen noch auf schnelle Spannungssprünge reagieren können. Deshalb wird nach der Simulation das Bode-Diagramm und die Sprungantwort untersucht.

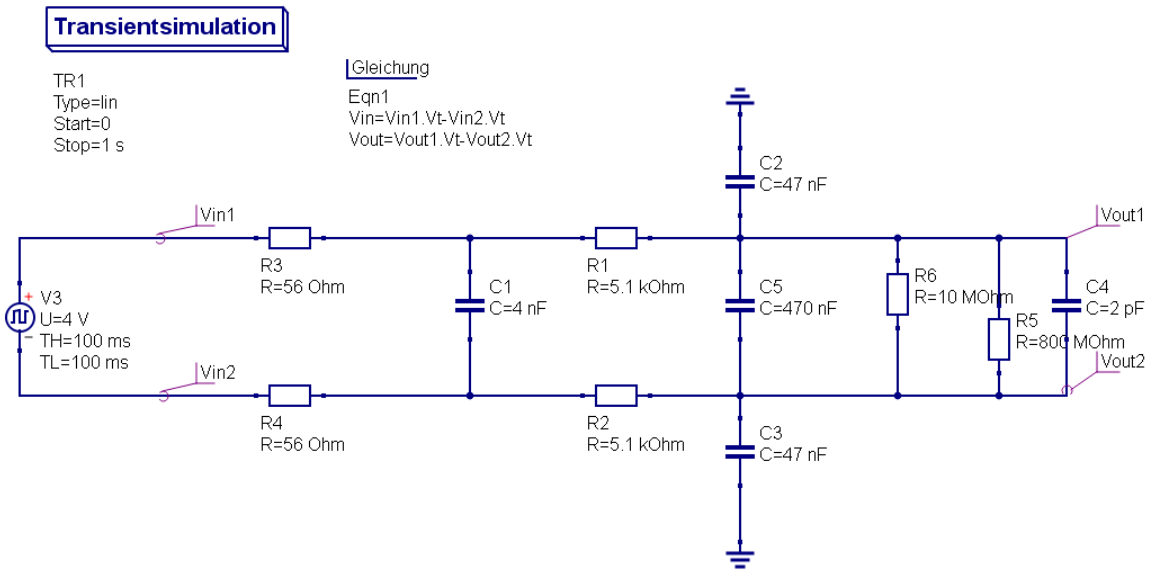


Abbildung 3.13: Tiefpass Simulation

Abbildung 3.13 zeigt die aufgebaute Simulation für die Sprungantworten. Um die Grenzfrequenz des Tiefpasses zu bestimmen wird ein Bodediagramm erstellt. Bei der Simulation für das Bodediagramm wird der Rechteckgenerator durch eine sinusförmige Spannungsquelle ausgetauscht und anschließend ein sweep über die Frequenz durchgeführt. Die Supressordiode hat große kapazitive Eigenschaften, deshalb wird sie als Kapazität C1 dargestellt. R1, R2, C2, C3 und C5 stellen den analogen Tiefpass dar. R5 und C4 bilden den Eingang des In-Amps nach. Da dieser viel zu hochohmig ist, wird parallel am Eingang ein 10MΩ Widerstand platziert.

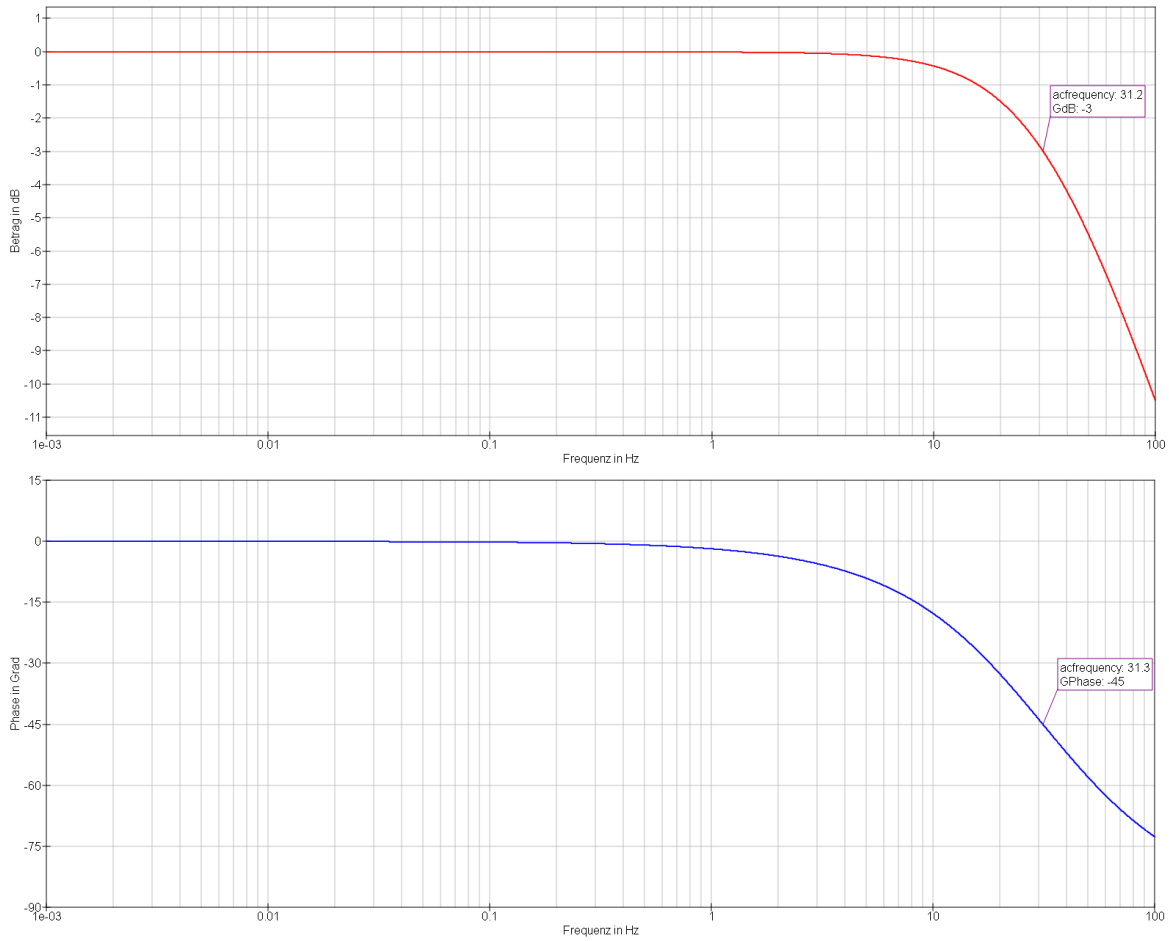


Abbildung 3.14: Tiefpass Simulation - Bode Diagramm

Abbildung 3.14 zeigt das Simulationsergebnis für das Bode-Diagramm. Man sieht ein typisches Tiefpassverhalten in der Amplitude und Phase. Die 3dB Grenzfrequenz liegt bei 31,2Hz.

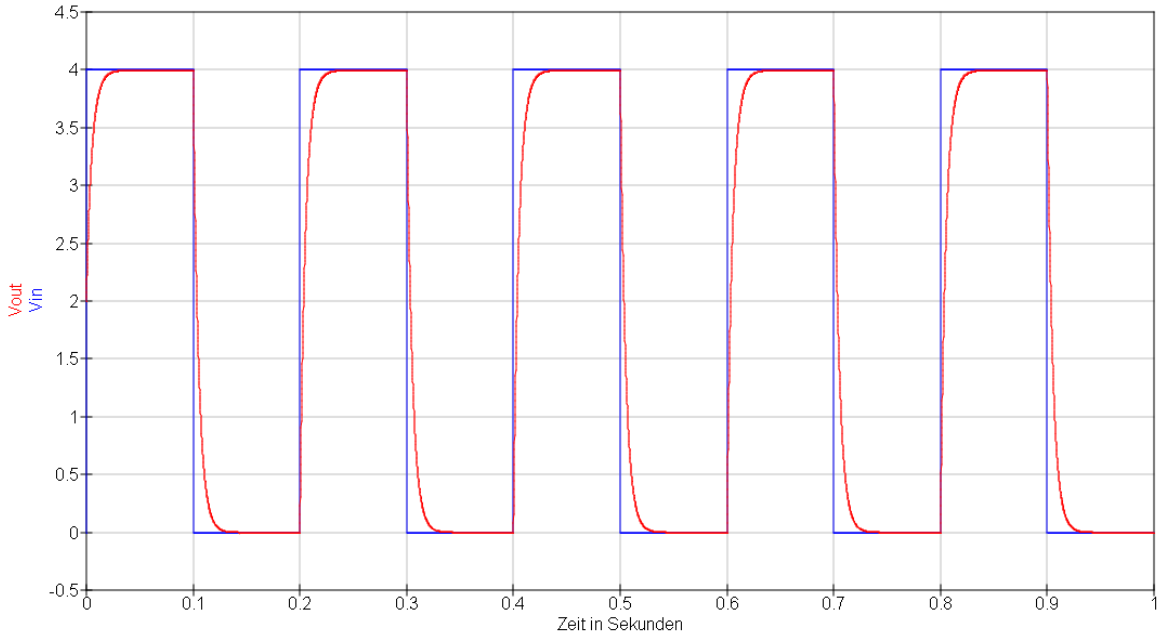


Abbildung 3.15: Tiefpass Simulation - Sprungantwort

Abbildung 3.15 zeigt die Sprungantwort der Simulation. Blau ist die ideale rechteckförmige Eingangsspannung und rot ist die Ausgangsspannung am Instrumentenverstärker. Wichtig ist hierbei, dass der Sprung innerhalb von 50ms seinen stationären Zustand erreicht, damit anschließend die Spannung korrekt am ADC verarbeitet werden kann.

3.4.2 Netzteil

Beim Netzteil fällt die Wahl auf eine „Doppelte Mittelpunktschaltung“, weil sich mit wenig Aufwand eine negative Spannung erzeugen lässt. Die Grundlagen für diese Schaltung sowie die Formel für die Dimensionierung der Siebkondensatoren findet man im Buch „Halbleiter-Schaltungstechnik“ von Tietze & Schenk [24, S.898].

$U_{Br,ss}$: Brummspannung, I_a: maximale Strom des Netzteils, C_L: Siebkondensatoren,
f_N: Frequenz der Wechselspannung, R_i: Innenwiderstand des Transformators,
R_v: Lastwiderstand

$$U_{Br,ss} = \frac{I_a}{2C_L f_N}\left(1 - \sqrt[4]{\frac{R_i}{2R_v}}\right) \tag{3.3a}$$

Da die verwendeten 9V Festspannungsregler eine Dropout Voltage von 2V haben, bedeutet dass die Brummspannung nicht unter 11V sinken darf. Der Transformator liefert eine 18V Spannung. Diese ergibt ideal gleichgerichtet $\frac{18V}{\sqrt{2}} = 12,73V$. Wird davon zwei mal 0,7V für die Dioden im Gleichrichter abgezogen, erhält man eine maximale Brummspannung von 11,33V. Der maximale Strom, den das Netzteil liefern kann, soll $70mA = I_a$ betragen.

$$U_{Br,ss} = 11,33V - 11V = 330mV \tag{3.4a}$$

Stellt man die Formel 3.3a nach dem Siebkondensator um, erhält man:

$$C_L = \frac{I_a}{2U_{Br,ss} f_N}\left(1 - \sqrt[4]{\frac{R_i}{2R_v}}\right) \tag{3.4b}$$

$$C_L = \frac{0,07A}{2 \cdot 0,33V \cdot 50Hz}\left(1 - \sqrt[4]{\frac{0,2\Omega}{2 \cdot 600\Omega}}\right) \tag{3.4c}$$

$$C_L = \frac{0,07A}{2 \cdot 0,33V \cdot 50Hz} \cdot 0,89 = 1888\mu F \tag{3.4d}$$

Da dies die Mindestkapazität für die Siebkondensatoren ist, wird die Kapazität der Kondensator um 10% erhöht. Der nächst mögliche erhältliche Elektrolyt-Kondensator hat 2200μF.

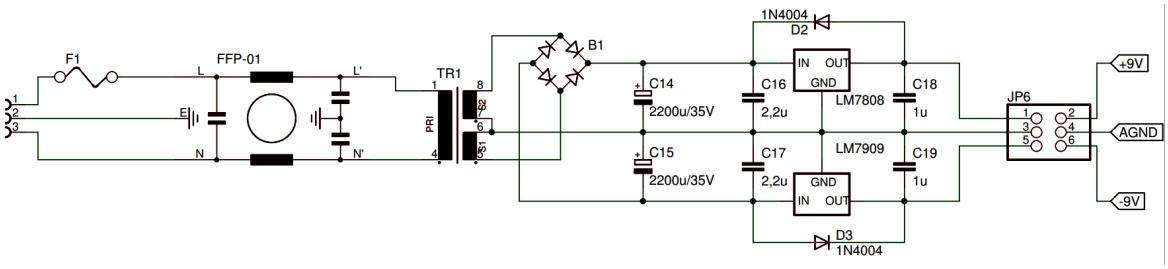


Abbildung 3.16: Netzteil - Aufbau und Dimensionierung

Abbildung 3.16 zeigt den Schaltplan des Netzteils. Außer den üblichen Komponenten eines Netzteils wie Gleichrichter, Transformator und Festspannungsregler verfügt das Netzteil über zusätzliche Elemente. Dazu gehören eine Sicherung F1 als Strombegrenzung, ein Netzfilter FFP-01 gegen Störungen auf der Netzseite und zwei Dioden als Schutz vor Verpolung.

3.4.3 Galvanische Entkopplung

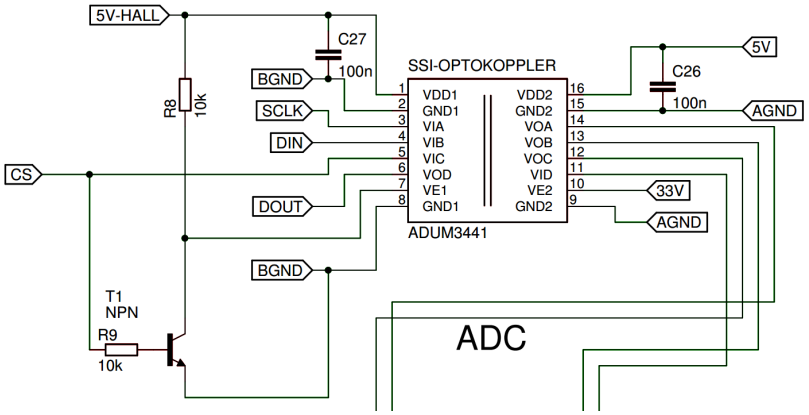


Abbildung 3.17: Optokoppler für die serielle Schnittstelle

Durch das eigene Netzteil mit Transformator ist die ADC Schutzschaltung, außer der SPI Schnittstelle, galvanisch getrennt. Um die galvanische Trennung zu vervollständigen, wird die SPI Schnittstelle durch einen Optokoppler getrennt. Dies ist in Abbildung 3.17 dargestellt. Da beim SPI-Bus die Teilnehmer, die nicht mit dem Host kommunizieren DOUT hochohmig werden lassen, muss dies der Optokoppler ebenso nachahmen. Dazu wird Pin7 des Optokopplers benutzt. Wenn ein High auf diesem Pin anliegt, werden die Ausgänge hochohmig. Dies ist invers zum „CS - Chip Select“ der SPI-Schnittstelle. Mithilfe eines Inverters am CS wird der Optokoppler zur richtigen Zeit hochohmig.

3.4.4 optionale Modifikationen

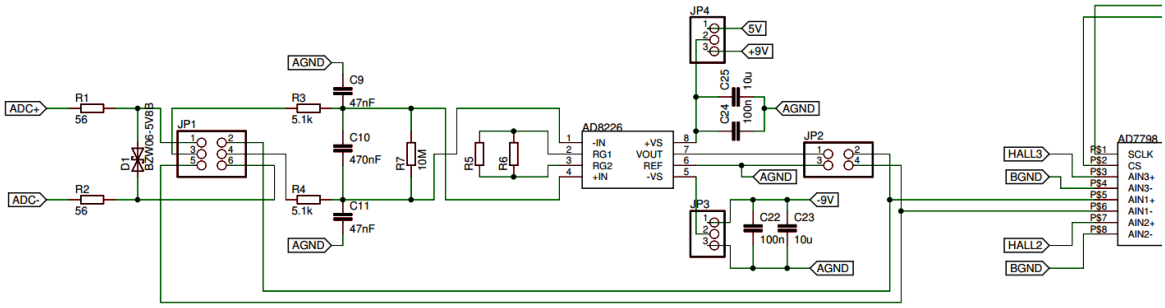


Abbildung 3.18: Einstellmöglichkeiten durch Jumper

Zum Einstellen der verschiedenen Betriebsmöglichkeiten sind „Pin Header“ vorgesehen, die sich mit Jumper konfigurieren lassen. In Abbildung 3.18 sind alle wesentliche Jumper Möglichkeiten zu sehen. „Pin Header“ JP1 und JP2 sind zum Umgehen des In-Amp AD8226. Soll der In-Amp benutzt werden brückt man JP1 1/3 und 4/6, an JP2 ist 1/2 und 3/4 zu brücken.

„Pin Header“ JP3 und JP4 dienen zum Einstellen der Versorgungsspannung des Instrumentenverstärkers. Ist JP3 auf AGND gebrückt, kann der In-Amp im „Single Supply“ betrieben werden. Mit JP4 stellt man die gewünschte Betriebsspannung von 5V oder 9V ein. Soll im „Dual Supply“ gearbeitet werden, sind JP3 auf +9V und JP4 auf -9V zu brücken.

4 Ausarbeitung geeigneter Messkonzepte

Um über das Verhalten der Batteriezelle möglichst viel in Erfahrung zu bringen, ist es wichtig, eine Systematik zu entwickeln. Die Zellen müssen katalogisiert und regelmäßig nach einem streng festgelegten Messplan untersucht werden. Dadurch ist es möglich, für ausgewählte Zellen Daten über einen langen Zeitraum zu gewinnen und zu analysieren.

In diesem Kapitel werden die genauen Messkonzepte, sowie der dazu gehörige Messplan ausgearbeitet.

4.1 Messkonzept nach dem Neuen Europäischen Fahrzyklus

Dieses Messkonzept basiert auf den Herstellerangaben führender Automobilhersteller, sowie dem NEFZ. Der Ansatz basiert auf physikalischen Gesetzen, die benötigte Leistung während eines NEFZ zu berechnen. Da diese errechnete Leistung auf die gesamte Batterie bezogen ist, wird eine Simulation dieser Batterie durchgeführt. Der kleinste Bestandteil dieser Batterie ist ein Batteriemodell der verwendeten Zelle. Somit ist bekannt, welche Ströme während des NEFZ auf eine einzelne Zelle wirken. Damit nun eine Messkonzept nach dem NEFZ entsteht, muss nun mit dem Zykliersystem die Belastung so gut wie möglich nachvollzogen werden.

Weil die Automobilhersteller nur wenig über ihre Antriebstechnologie preisgeben, sind die Berechnungen und Simulationen eine Annäherung an die Realität. Deshalb lohnt es sich nicht, dies extrem detailliert und genau zu machen. Das Ziel ist, dass am Ende dieser Berechnungen die Verbrauchswerte auf 100km übereinstimmen mit den Werten aus dem „DAT-Leitfaden“ [16].

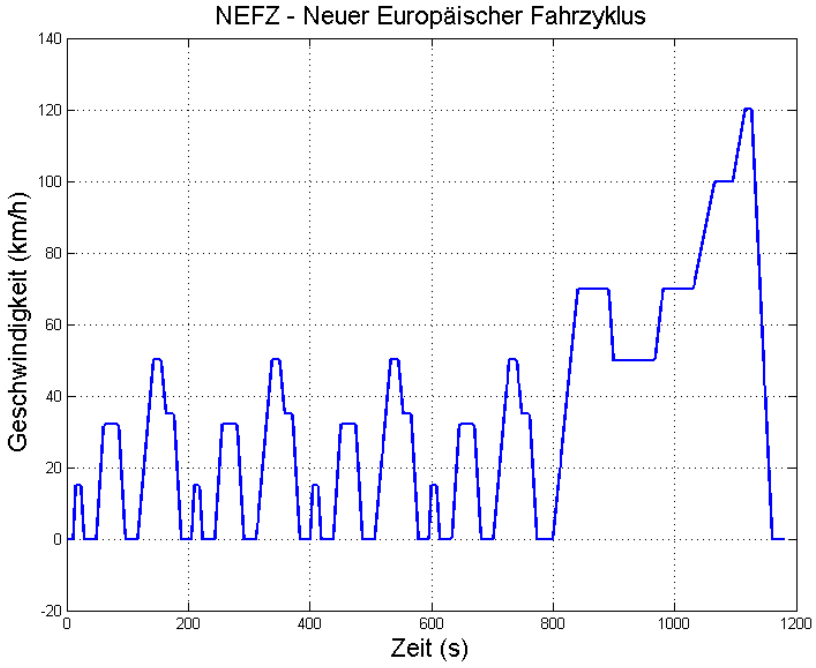


Abbildung 4.1: der NEFZ aus den Europäischen Richtlinien

Zunächst wird der NEFZ aus der Richtlinie 70/220/EWG in MATLAB übernommen. Dieser ist in Abbildung 4.1 dargestellt.

4.1.1 Berechnung der benötigten Leistung während des NEFZ

Zur Berechnung der benötigten Leistung während des NEFZ wird der Rollwiderstand, der Luftwiderstand und die benötigte kinetische Energie zur Beschleunigung berücksichtigt. Um auch den Wirkungsgrad des Akkus und Motors mit einzubeziehen, wird der Verlustfaktor V_F eingeführt.

Rollwiderstand:
F_{Roll}: Rollwiderstand in N, C_r: Rollwiderstandskoeffizient, m: Masse in kg,
g: Erdbeschleunigung (9,81$\frac{m}{sec^2}$)

$$F_{Roll} = C_r \cdot m \cdot g \tag{4.1a}$$

Luftwiderstand:
F_{Luft}: Luftwiderstand in N, A: Stirnfläche des Fahrzeugs in m^2, C_w: Luftwiderstandsbeiwert
D: Dichte der Luft (1,29$\frac{kg}{m^3}$), v: gefahrene Geschwindigkeit in $\frac{m}{sec}$

$$F_{Luft} = \frac{A}{2} \cdot C_w \cdot D \cdot v^2 \tag{4.1b}$$

kinetische Energie:
E_{kin}: kinetische Energie, m: Masse in kg, v: gefahrene Geschwindigkeit in $\frac{m}{sec}$

$$E_{kin} = \frac{1}{2} \cdot m \cdot v^2 \tag{4.1c}$$

Daraus lässt sich ableiten, was über das Auto bekannt sein muss. Es wird der Rollwiderstandskoeffizient C_r, der Luftwiderstandsbeiwert C_w, die Masse m und die Stirnfläche A des Fahrzeugs benötigt. Die Masse und die Abmaße zur Berechnung der Stirnfläche sind für jedes Fahrzeug angegeben. VW ist einer der wenigen Autohersteller die den Luftwiderstandsbeiwert offen darlegen. Dieser beträgt beim „VW e-up!“ 0,308=C_w. Der Rollwiderstandskoeffizient hängt eher von der Fahrbahn ab. Dieser beträgt bei einem Pkw auf Asphalt C_r=0,013.

Um nun die Leistung während des NEFZ zu erhalten, muss noch eine Umrechnung stattfinden und der Verlustfaktor V_F einbezogen werden. Da Leistung gleich Kraft mal Geschwindigkeit ist, ergibt sich für den Rollwiderstand und Luftwiderstand Folgendes.

Leistung des Rollwiderstands nach Gleichung 4.1a:
P_{Roll}: Leistung des Rollwiderstands in W, v: gefahrene Geschwindigkeit in $\frac{m}{sec}$
V_F: Verlustfaktor

$$P_{Roll} = C_r \cdot m \cdot g \cdot v \cdot V_F \tag{4.2a}$$

Leistung des Luftwiderstandsnach Gleichung 4.1b:
P_{Luft}: Leistung des Luftwiderstands in W, V_F: Verlustfaktor

$$P_{Luft} = \frac{A}{2} \cdot C_w \cdot D \cdot v^3 \cdot V_F \tag{4.2b}$$

Um aus der kinetischen Energie die Leistung zu gewinnen, muss die kinetisch Energie abgeleitet werden.

P_{kin}: kinetische Leistung in W

$$P_{kin}(t) = \frac{dE_{kin}(t)}{dt} \tag{4.3a}$$

Da die vorliegenden Werte in MATLAB zeitdiskret sind, ergibt sich mit der Simulationsschrittweite ts:

$$P_{kin} = \frac{\triangle E_{kin}}{\triangle t} = \frac{\triangle E_{kin}}{ts} \tag{4.3b}$$

Wird die Gleichung 4.1c eingesetzt und der Verlustfaktor V_F berücksichtigt, ergibt sich für P_{kin}:

$$P_{kin} = \frac{0.5 \cdot m \cdot v_2^2 - 0.5 \cdot m \cdot v_1^2}{ts} \cdot V_F = \frac{m \cdot (v_2^2 - v_1^2)}{2 \cdot ts} \cdot V_F \tag{4.3c}$$

Da die Berechnung für die Leistung bekannt ist, wird mit einer typischen Fallunterscheidung der NEFZ durchlaufen. Dabei unterteilt die Fallunterscheidung den NEFZ in vier Bereiche:

1. **stehen** - Wenn das Auto steht, wird keine Leistung benötigt.
2. **beschleunigen** - Beim Beschleunigen setzt sich die Leistung aus P_{Roll},P_{Luft} und P_{kin} zusammen. Sofern das Fahrzeug in Bewegung gerät wirkt der Roll- und Luftwiderstand. Die kinetische Leistung wird aufgrund der Geschwindigkeitsänderung addiert, da $(v_2^2 - v_1^2) > 0$ ist.
3. **bremsen** - Der Bremsvorgang besteht nur aus P_{Roll} und P_{Luft}. Da die kinetische Leistung $(v_2^2 - v_1^2) < 0$ ist, würde das Fahrzeug beim Bremsen Leistung erhalten, sprich Rekuperation. Dieser Fall wird in dieser Simulation nicht berücksichtigt.
4. **konstant** - Bei konstanter Geschwindigkeit wird nur gegen den Roll- und Luftwiderstand gearbeitet. Deshalb besteht er ebenfalls aus P_{Roll} und P_{Luft}.

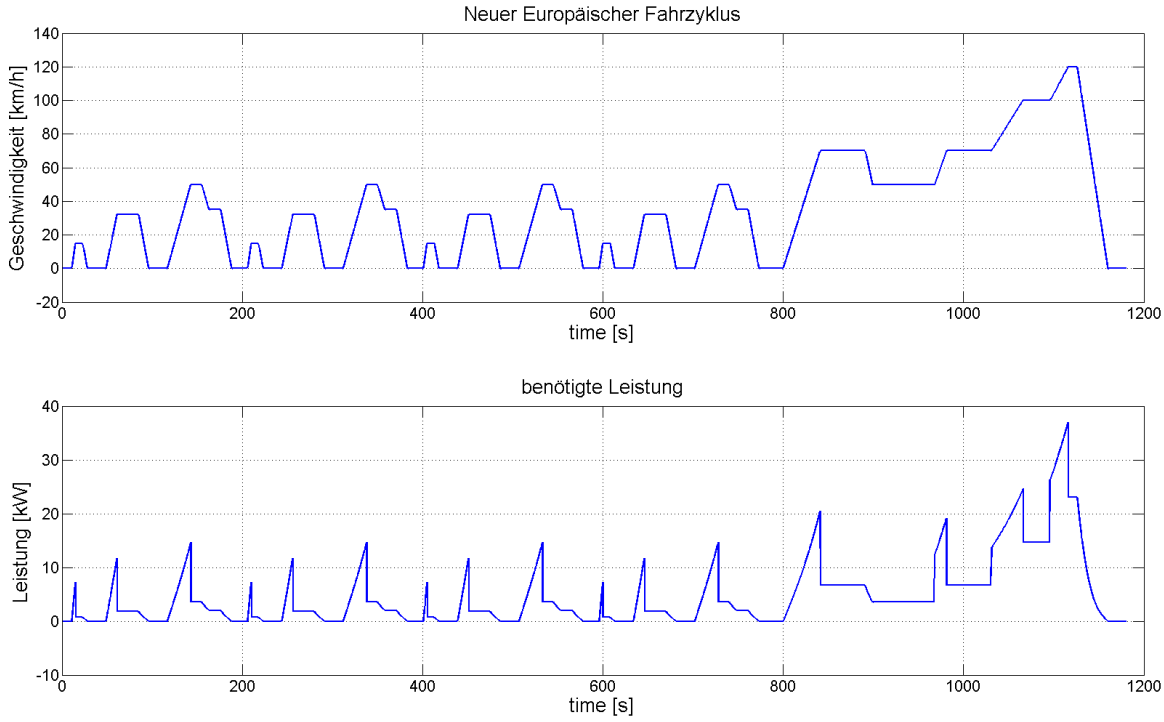


Abbildung 4.2: der geschätzte Leistungsverbrauch eines „VW e-up!“ während des NEFZ

Das Ergebnis ist in Abbildung 4.2 dargestellt. Es ist gut zu erkennen, dass der Beschleunigungsvorgang durch die benötigte kinetische Energie die höchste Leistung benötigt. Des

Weiteren ist bei hohen Geschwindigkeiten, wie beim außerorts Zyklus, der kubische Einfluss des Luftwiderstandes zu sehen. Bei niedrigen Geschwindigkeiten ist eher der lineare Rollwiderstand dominant.

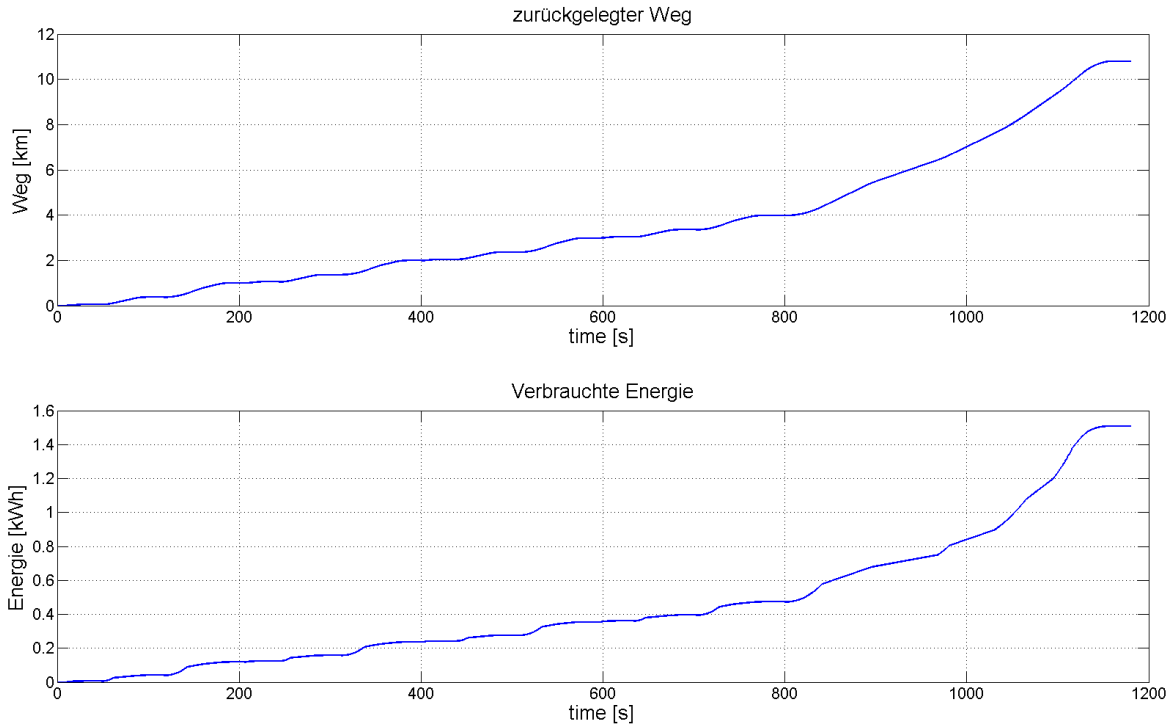


Abbildung 4.3: die Integration der Leistung und Geschwindigkeit ergibt Energie und Weg

Wird die Geschwindigkeit und die Leistung über die Zeit auf integriert, ergibt sich der zurückgelegten Weg und die verbrauchte Energie. In Abbildung 4.3 ist die Integration zu sehen. Aus den Endwerten der Integration lässt sich die benötigte Energie auf 100km berechnen.

Fahrzeugbezeichnung	Verbrauch in $\frac{kWh}{100km}$ laut Berechnung	Verbrauch in $\frac{kWh}{100km}$ laut „DAT-Leitfaden“
Renault Twizy 45	6,54	5,8
VW e-up!	13,97	11,7
Citroen C-Zero	12,57	12,6

Tabelle 4.1: Vergleich des Verbrauchs auf 100km zwischen der MATLAB Berechnung und dem „DAT-Leitfaden“

Der Vergleich der Verbrauchswerte auf 100km zwischen der Berechnung und den öffentlich bekannten Werten zeigt beim „VW e-up!“ starke Unterschiede. Das liegt daran, dass der „VW

e-up!“ rekuperieren kann, d.h. er kann Energie im Bremsvorgang zurückgewinnen. Dies wurde in den Berechnungen nicht berücksichtigt. Kleinere Abweichungen sind durch die simple Berechnung, sowie unbekannten C_r und C_w Koeffizienten zu erklären. Die MATLAB Skripte befinden sich im Anhang C.1 und C.2 sowie auf DVD.

4.1.2 Simulation der Batterie

Die Simulation der Autobatterie findet mit der Toolbox „Simulink“ in MATLAB statt. Dabei werden hauptsächlich die fertigen Modelle aus „Simscape“ benutzt. „Simscape“ ist eine umfangreiche Modellbibliothek zur Simulation von physikalischen Systemen. Als Grundlage der Simulation dienen die Angaben der Automobilhersteller zur Autobatterie, sowie das Batteriemodell der Bibliothek „Simscape“.

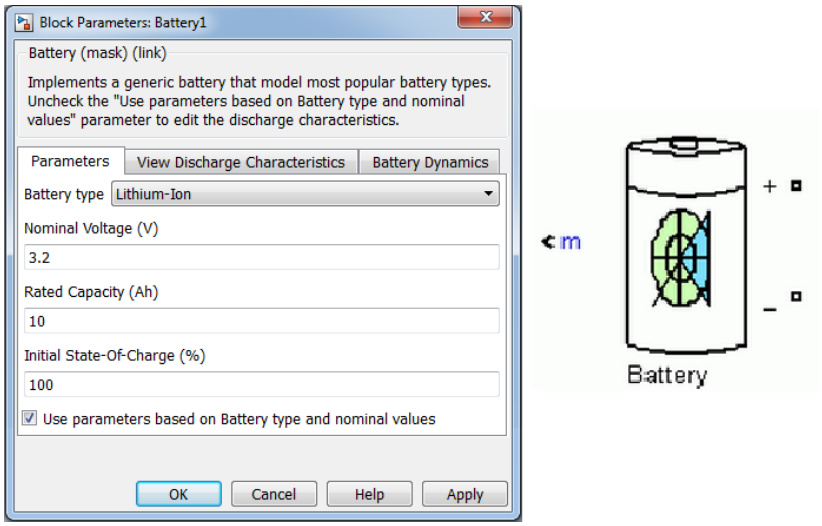


Abbildung 4.4: Das Batteriemodel aus Simscape

Abbildung 4.4 stellt das dynamisches Batteriemodell dar. Es wurde nach dem Datenblatt der Lithium-Zellen konfiguriert, die während der Erprobung zum Einsatz kommen. Es wurde speziell für die E-Mobilität entwickelt. Die Grundlage bildet das „Shepherd“ Batteriemodell. Olivier Tremblay und Louis-A. Dessaint veröffentlichten 2009 ein Bericht [25] zu diesem Modell.

Da vom „VW e-up!“ die meisten Details über den Aufbau der Batterie bekannt sind, würde die Simulation auf deren Fakten basieren. Aus dem Informationsmaterial von VW [26] wurden die Eckdaten der Batterie extrahiert.

Eckdaten der Batterie:

- Nennspannung: 374V
- Module: 17
- Zellen: 204
- Zellpaare: 102
- Kapazität: 18,7 kWh

Im „BATSEN“ Projekt stehen uns Lithium-Eisenphosphat-Zelle, kurz LiFePO4, zur Verfügung. Sie sind von der Firma Headway. Die genaue Bezeichnung lautet „40152SE“. Sie besitzen eine Kapazität von 15 Ah bei einer Nennspannung von 3,3V.

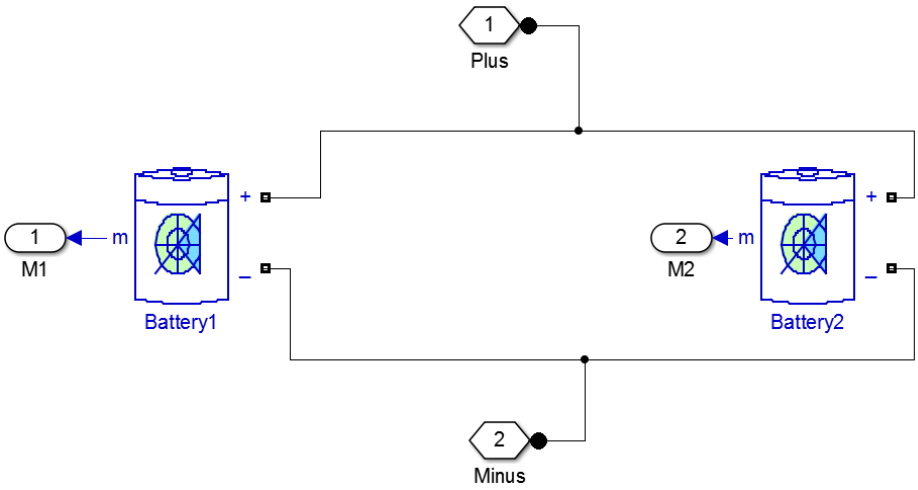


Abbildung 4.5: Die Lithium-Ionen Autobatterie auf der 1. Ebene der Simulationen

Um mit den Lithium-Zellen aus dem „BATSEN“ Projekt die verwendeten Zellen von VW nachzubilden, müssen zwei Zellen parallel geschaltet werden, da ansonsten die Kapazität nur halb so groß wie in der Realität wäre. Abbildung 4.5 zeigt die unterste Ebene der Simulation. Sie dient ausschließlich dem Nachbilden der VW Zellen.

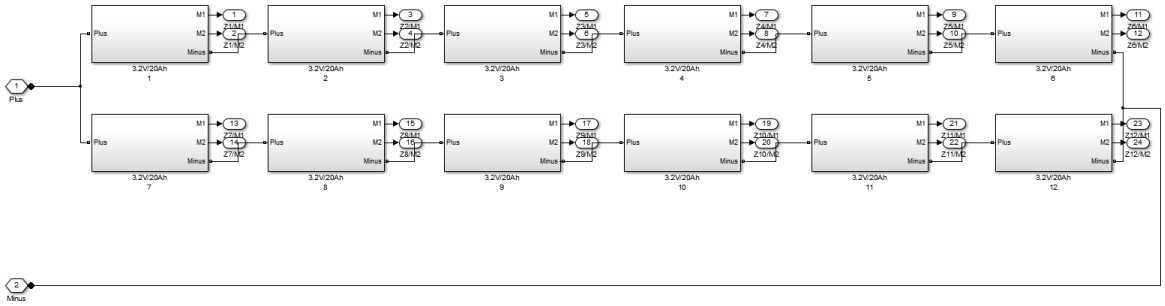


Abbildung 4.6: Die Lithium-Ionen Autobatterie auf der 2. Ebene der Simulationen

Die zweite Ebene der Simulation besteht ausschließlich aus Modellen der ersten Ebene. Sie bildet ein Modul nach. Ausgehend davon, dass alle 17 Module in Reihe geschaltet sind und zusammen eine Spannung von 374V haben, werden 6 Zellpaare in Reihe geschaltet. In Abbildung 4.6 ist der Aufbau eines Moduls zu sehen.

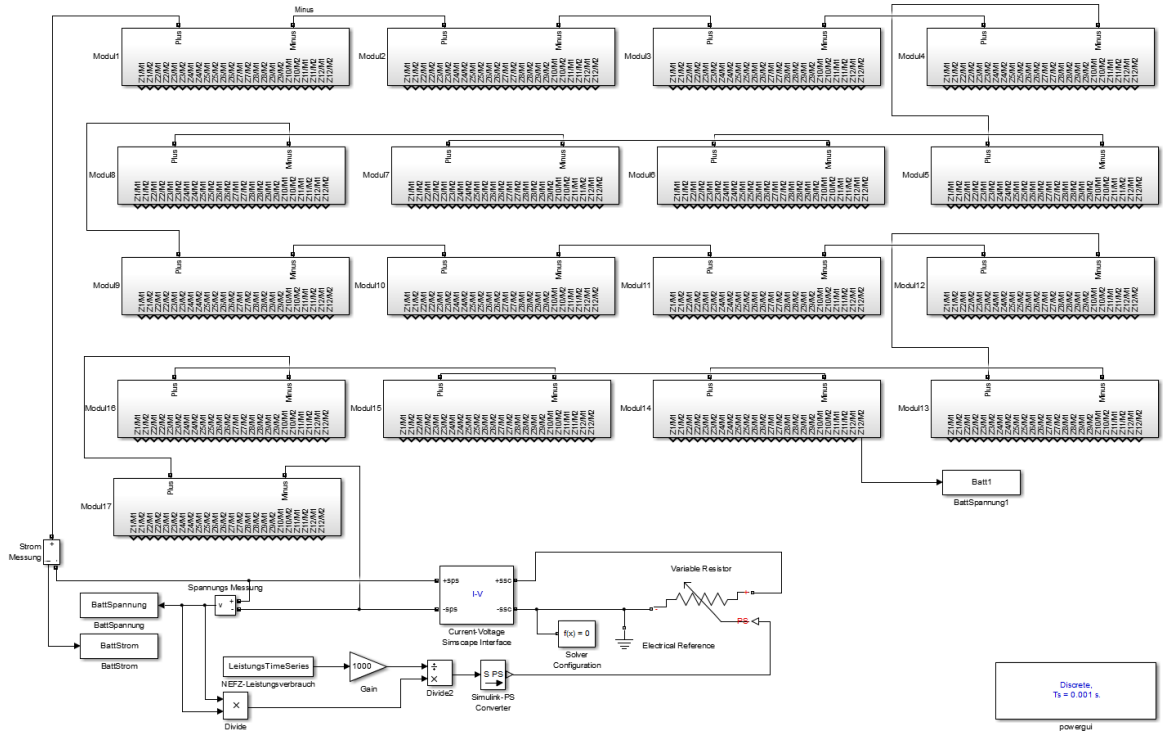


Abbildung 4.7: Die Lithium-Ionen Autobatterie auf der 3. Ebene der Simulationen

In der letzten Ebene, Abbildung 4.7, werden 17 Module zu einer Gesamtbatterie geschaltet. Ein variabler Widerstand sorgt für nötige Last während der Simulation. Dabei werden von der einzelnen Zelle und der Gesamtbatterie Strom und Spannung aufgezeichnet.

4.1.3 Auswertung der Simulation

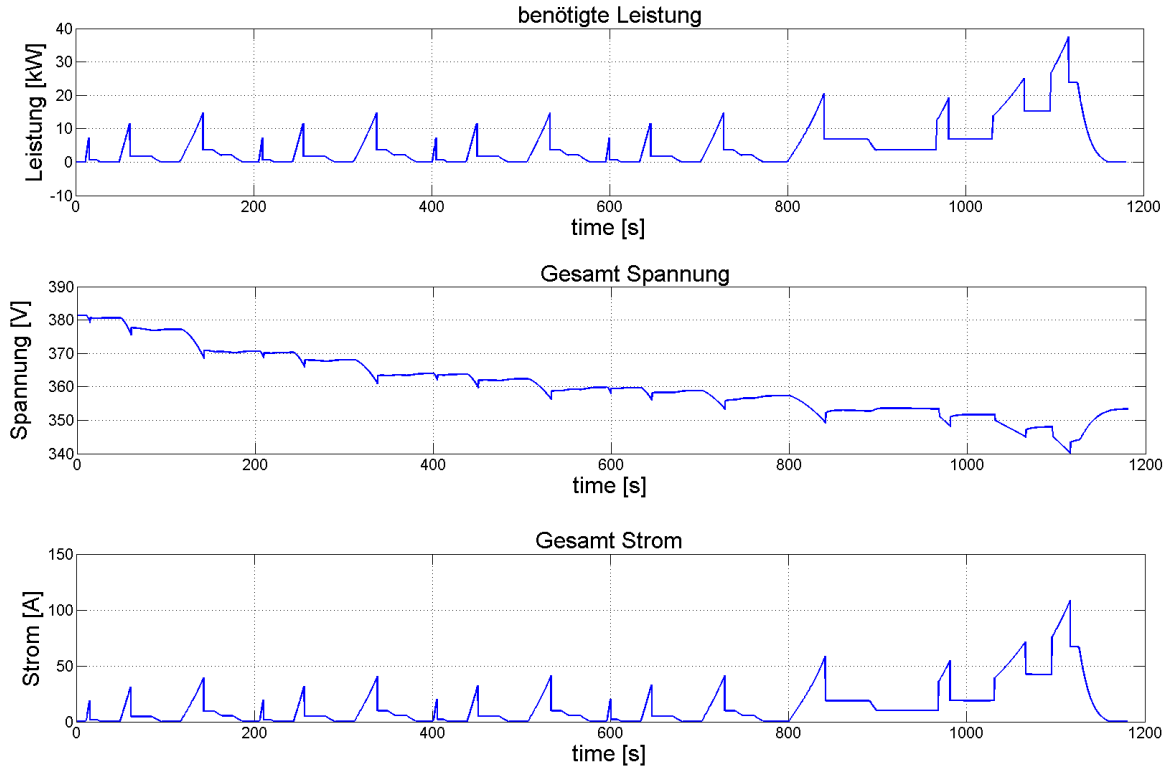


Abbildung 4.8: Simulationsergebnisse der Lithium-Ionen Autobatterie

Abbildung 4.8 zeigt die gewonnen Simulationsergebnisse der Gesamtbatterie. Es ist deutlich erkennbar, dass die Spannung während einer großen Belastung einbricht. Am Ende der Simulation ist die Spannung der unbelasteten Batterie um 25V gesunken. Der Spitzenwert des Stroms liegt während der Geschwindigkeit von 120 $\frac{km}{h}$ bei 100A.

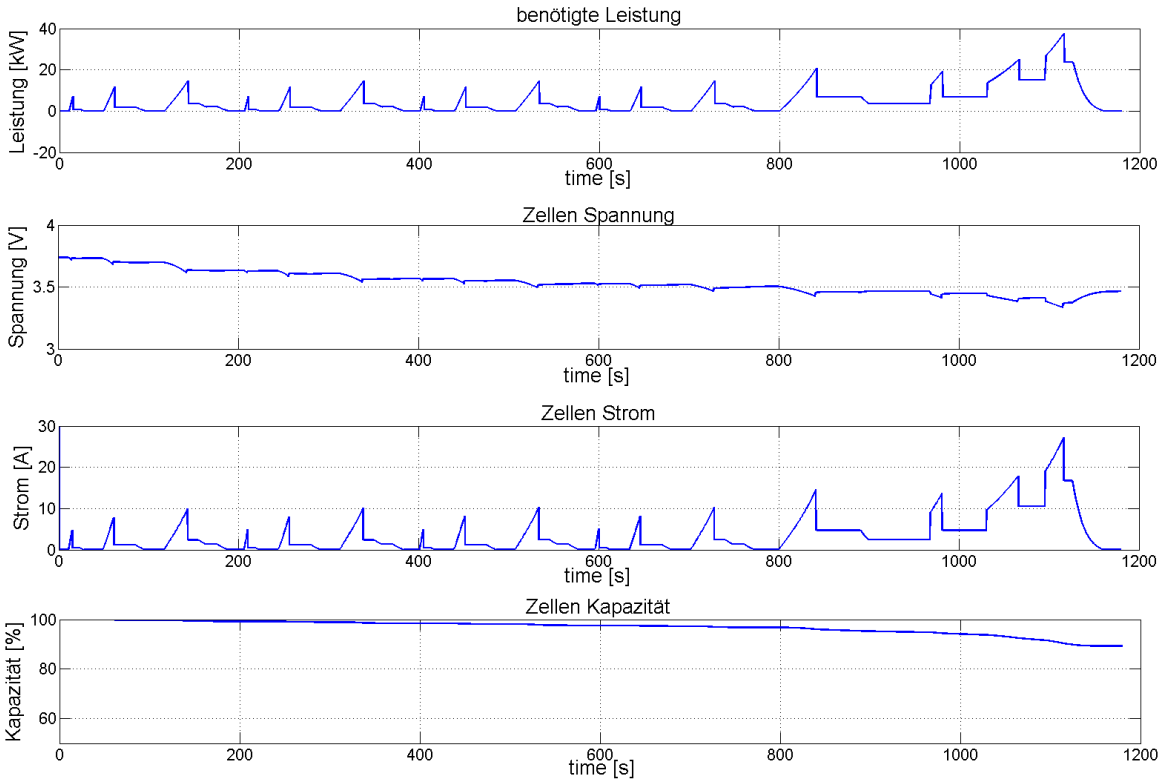


Abbildung 4.9: Simulationsergebnisse der einzelnen Zelle

In Abbildung 4.9 sind die Verläufe für Spannung, Strom und Kapazität einer einzelnen Zelle zu sehen. Der Verlauf ist ähnlich wie bei der Gesamtbatterie, jedoch bei der Spannung um den Faktor 102 und beim Strom um den Faktor 4 kleiner. Da jeweils in jedem Modul 6 Zellenpaare in Reihe geschaltet sind, ergibt das bei 17 Modulen 102 Zellenpaare in Reihe. Somit ist der Faktor 102 bei der Spannung plausibel. Beim Strom ist der Faktor 4 dadurch zu erklären, dass ein Zellenpaar aus insgesamt vier parallel geschalteten Zellen des „BATSEN“ Projekts besteht. Die höchste Belastung findet während des Zyklus außerorts statt, bei $120\frac{km}{h}$. Dort wird die Zelle mit 1,7C entladen. Selbst bei einer dauerhaften Belastung von 1,7C ist das kein Problem für eine Lithium-Zelle.

4.1.4 Messkonzept basierend auf NEFZ

Basierend auf den Simulationsergebnissen, wird nun das Messkonzept ausgearbeitet. Aus den ermittelten Strom- und Spannungswerten während des NEFZ wird mittels des ohmschen Gesetzes der dynamische Widerstand berechnet. Diesen Widerstand gilt es, bestmöglich nachzubilden. Dazu stehen dem „BATSEN“ Projekt drei Hochleistungspotentiometer zu Verfügung, die einen maximalen Strom von 20 Ampere zulassen. Da das Zykliersystem vier schaltbare Relais besitzt, werden drei Relais mit den Potentiometer beschaltet, während das vierte Relais zum Laden der Zelle dient.

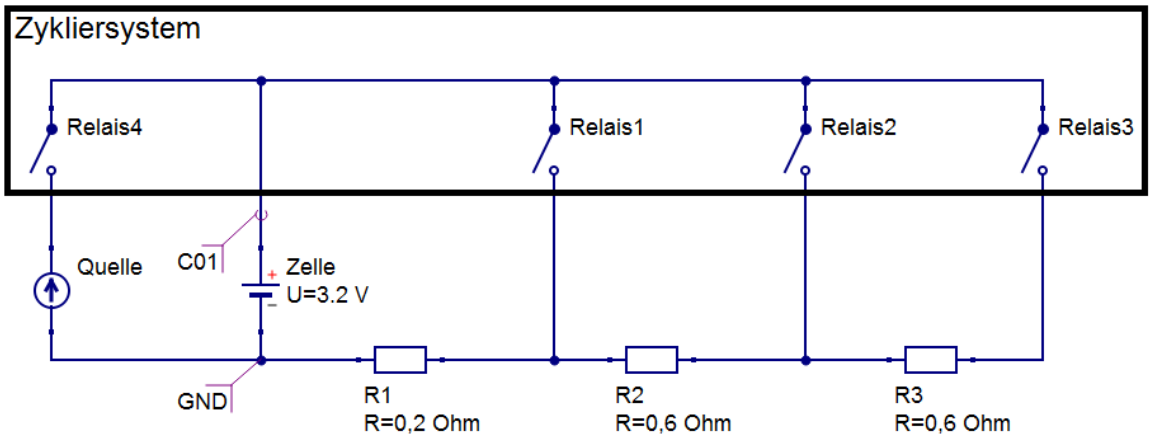


Abbildung 4.10: Messaufbau des NEFZ Messkonzepts

Abbildung 4.10 zeigt den Aufbau zum NEFZ Messkonzept. Da die Widerstände einen maximalen Widerstand von 0,66Ω haben, werden sie in Reihe geschaltet. Dadurch sind auch größere Widerstände möglich. Durch das Schalten der einzelnen Relais ergibt sich folgender Gesamtwiderstand:

- **Relais1 =** $R_1 = 0,2\Omega$
- **Relais2 =** $R_1 + R_2 = 0,8\Omega$
- **Relais3 =** $R_1 + R_2 + R_3 = 1,4\Omega$

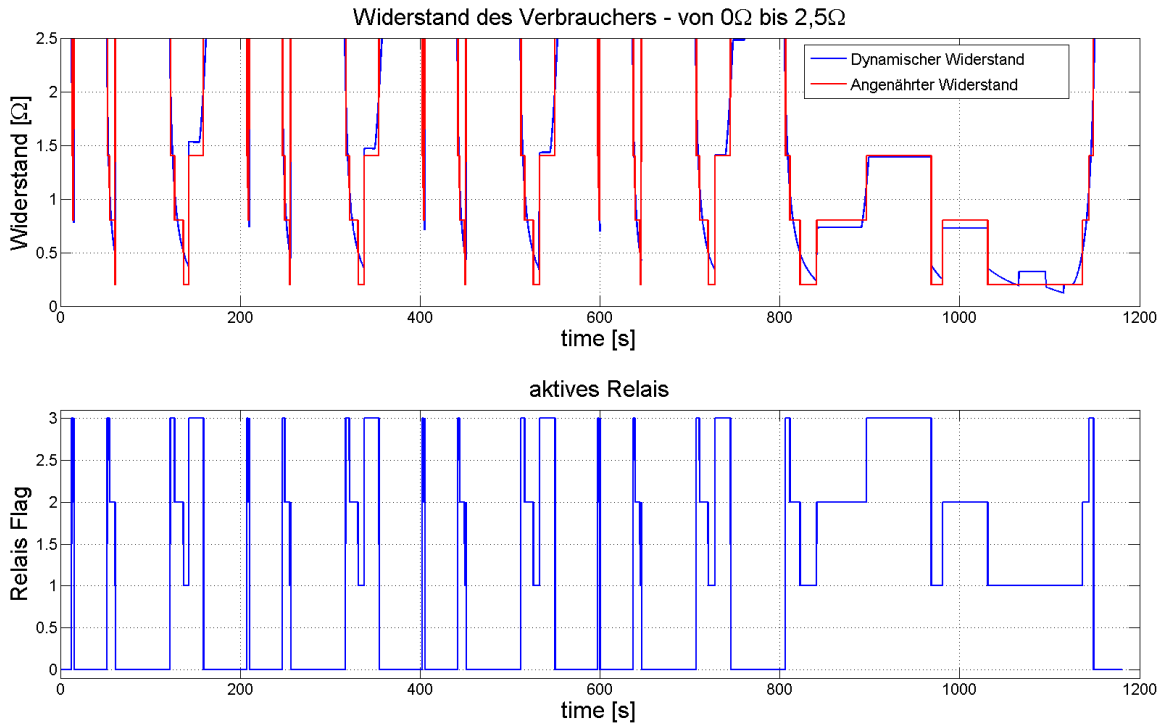


Abbildung 4.11: Nachbildung der dynamischen Last mit drei Leistungswiderständen

Abbildung 4.11 besteht aus zwei Plots. Der Obere zeigt die Annäherung des dynamischen Widerstands (blau) mit den drei Hochleistungspotentiometer (rot). Treppenförmig passen sich die Widerstände an die tatsächliche Last an. Der Untere stellt die zu schaltenden Relais während des NEFZ dar.

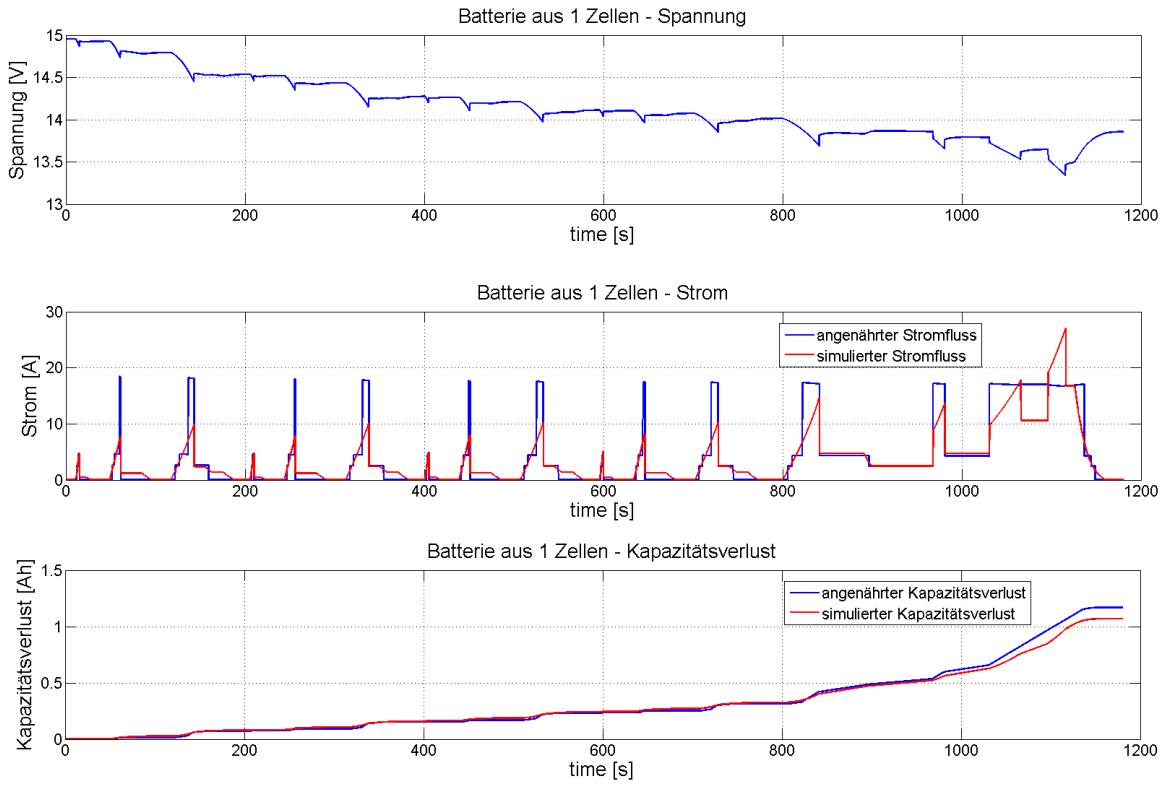


Abbildung 4.12: Simulationsergebnisse der Nachbildung

Durch das treppenförmige Nachbilden der Last verändert sich hauptsächlich der Stromfluss. Dieser ist in Abbildung 4.12, mittlerer Plot, dargestellt. Der Strom erhält ebenso einen stufenförmige Charakteristik. Ansonsten verhält sich die Annäherung mit den drei Widerständen genauso wie die Simulation. Selbst die verbrauchte Energie ist am Ende des NEFZ nahezu identisch.

4.2 Messkonzept zur Erstellung von Zellenprofilen

Der Schwerpunkt dieses Konzeptes liegt in der Untersuchung der Kapazität unter Berücksichtigung der Temperatur während des Alterungsvorgangs. Die gewonnen Daten liefern Informationen über den Zelltyp und ermöglichen eine genaue Schätzung über den SOC „State of Charge“ und den SOH „State of Health“.

Eine Messdatenreihe stellt das Profil einer Zelle dar. Diese Messdatenreihe besteht aus neun einzelnen Messungen. Jede Messung beginnt mit dem Aufladen der Batterie bei $20°C$. Nach einer dreistündigen Ruhephase nach dem Laden, wird bei drei unterschiedlichen Temperaturen die Batterie entladen. Bei jeder Temperatur wird wiederum mit drei unterschiedlichen Entladeströmen gearbeitet.

Messung	Temperatur	Entladestrom
1	10˚C	$^1/_3$ C
2	10˚C	$^2/_3$ C
3	10˚C	$^3/_3$ C
4	20˚C	$^1/_3$ C
5	20˚C	$^2/_3$ C
6	20˚C	$^3/_3$ C
7	30˚C	$^1/_3$ C
8	30˚C	$^2/_3$ C
9	30˚C	$^3/_3$ C

Tabelle 4.2: alle neun Messungen einer Messdatenreihe

In Tabelle 4.2 sind die Werte zu den Temperaturen und den Entladeströmen explizit angegeben. Zur Erläuterung, der C-Faktor, der beim Laden und Entladen angegeben wird, ist eine Normierung auf der Kapazität der Zelle. Als Beispiel gehen wir davon aus, dass eine Zelle mit 20Ah mit 0,5C entladen werden soll. Das bedeutet, die Zelle wird mit 10A entladen.

No.	Dauer [min]	Relais	Beschreibung	Config-Befehl	Temperatur [°C]
01	1	0	warten	relsel [No.] wait 1 0	20
02	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
03	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	10
04	x	2	entladen mit 1/3C	relsel [No.] discharge_cellV [1/3C] 2	
05	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20
06	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
07	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	
08	x	2	entladen mit 1/3C	relsel [No.] discharge_cellV [1/3C] 2	
09	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	
10	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
11	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	30
12	x	2	entladen mit 1/3C	relsel [No.] discharge_cellV [1/3C] 2	
13	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20
14	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
15	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	10
16	x	2	entladen mit 2/3C	relsel [No.] discharge_cellV [2/3C] 2	
17	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20
18	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
19	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	
20	x	2	entladen mit 2/3C	relsel [No.] discharge_cellV [2/3C] 2	
21	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	
22	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
23	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	30
24	x	2	entladen mit 2/3C	relsel [No.] discharge_cellV [2/3C] 2	
25	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20
26	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
27	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	10
28	x	2	entladen mit 3/3C	relsel [No.] discharge_cellV [3/3C] 2	
29	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20
30	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
31	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	
32	x	2	entladen mit 3/3C	relsel [No.] discharge_cellV [3/3C] 2	
33	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	
34	x	1	laden bis zur „LSS-Ladeschlussspannung“	relsel [No.] charge_cellV [LSS] 1	
35	180	0	warten – Ruhephase nach dem Laden	relsel [No.] wait 180 0	30
36	x	2	entladen mit 3/3C	relsel [No.] discharge_cellV [3/3C] 2	
37	180	0	warten – Ruhephase nach dem Entladen	relsel [No.] wait 180 0	20

Tabelle 4.3: allgemeines Messkonzept zur Erstellung von Zellenprofilen

Tabelle 4.3 zeigt das allgemeine Messkonzept zur Zellenprofilerstellung. Um die Messung sicherer zu machen, werden beim Laden und Entladen spannungsabhängige Schaltmomente benutzt. Da dieses Messkonzept allgemein gültig für alle Batterietypen anwendbar bleiben soll, wurden die gelben Zeilen eingefügt. Die gelben Zeilen (No. 02,14 und 26) sind Aufladevorgänge, die je nach Zelltyp durch separate Zellaufladungen ausgetauscht werden müssen. Das bedeutet, wenn die Zellen untereinander durch schlechte Verarbeitung stark voneinander abweichen, gerät die Batterie schnell aus der Balance. Wenn das der Fall ist, muss bei den gelben Zeilen jede Zelle der Batterie separat aufgeladen werden, damit die Batterie in Balance kommt. Darauf wird im folgenden Kapitel 4.3 näher eingegangen. Da durch die spannungsabhängige Schaltmomente die Dauer der Lade- und Entladevorgänge schlecht

ein schätzbar ist, müssen ,solange der Temperaturschrank nicht vom Zykliersystem gesteuert werden kann, die 37 Schritte aufteilt werden.

Beim dargestellten Messkonzept wird davon ausgegangen, dass auf Relais 1 die Spannungsquelle liegt, die auf die Ladeschlussspannung eingestellt ist. Dabei ist die Strombegrenzung auf 1,5C. Auf Relais 2 liegt die elektronische Last, die je nach gewünschten Entladung eingestellt wird. Die Strombegrenzung ist ebenfalls auf 1,5C gestellt.

4.3 Ladevorgang

Das Laden sollte bei einheitlichen $20°C$ mit kleinen Strömen die 0,5C nicht überschreiten, statt finden, da ansonsten Werte wie der Restladegrad verfälscht werden. Je nach Zellentyp müssen die einzelnen Zellen erst auf ein gemeinsames Kapazitätslevel gebracht werden bevor sie gemeinsam zykliert werden. Ansonsten besteht die Gefahr einzelne Zellen in der Batterie zu überladen, da das Zykliersystem über keinen Laderegler verfügt. Bei Blei-Säure Akkus ist das kein Problem, solange der Batterie regelmäßig Wasser zugeführt wird. Nickel-Cadmium Akkus hingegen laufen aus und werden unbrauchbar. Der gefährlichste Effekt ist bei Lithium-Ionen Akkus zu beobachten. Sie überhitzen und brennen ab. Dieser Effekt wird auch als „thermal runaway“ bezeichnet. Zur Veranschaulichung dieser Problematik ist im folgenden Verlauf eine Simulation zum Laden von ungleichmäßig geladen Lithium-Ionen Zellen und eine Auswertung einer bereits existierenden Zyklierung einiger Lithium-Ionen Zellen dargestellt.

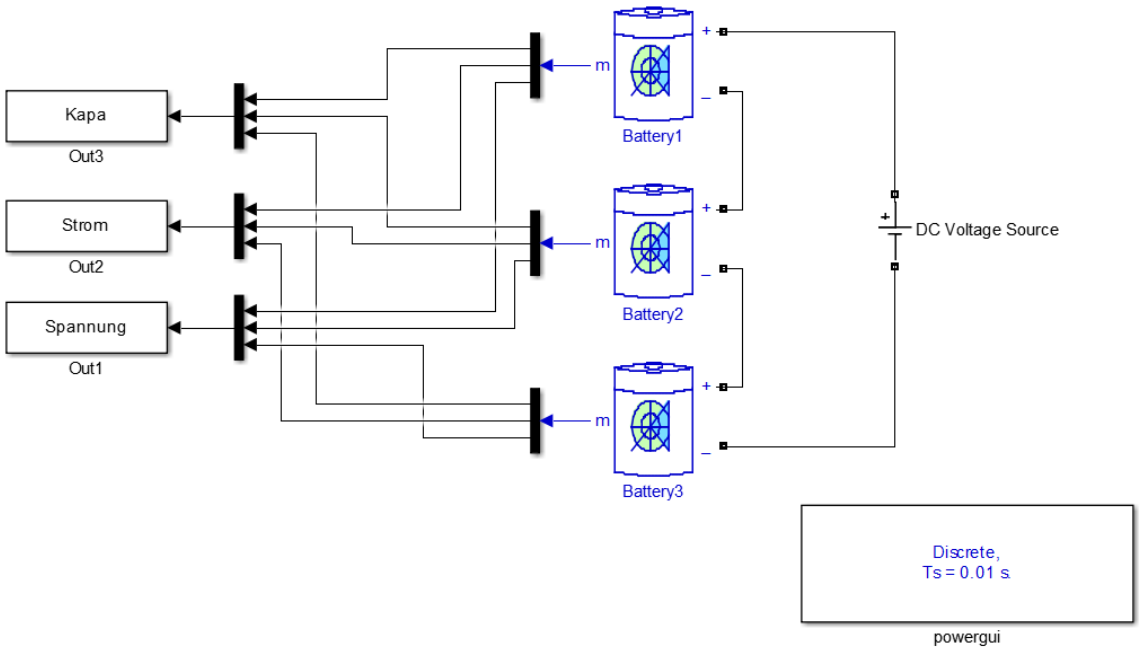


Abbildung 4.13: Simulation von drei in Reihe geschalteten Lithium-Ionen Zellen mit unterschiedlichem Ladezustand

Die Abbildung 4.13 zeigt eine Simulation bei der drei Lithium-Ionen Zellen mit unterschiedlichen Ladezustand in Reihe geladen werden. Dabei wurde auf das schon bekannte Batteriemodell aus der „MATLAB Simulink Simscape“ Bibliothek zurückgegriffen. Von allen drei Zellen werden Spannung, Strom und Kapazität aufgezeichnet.

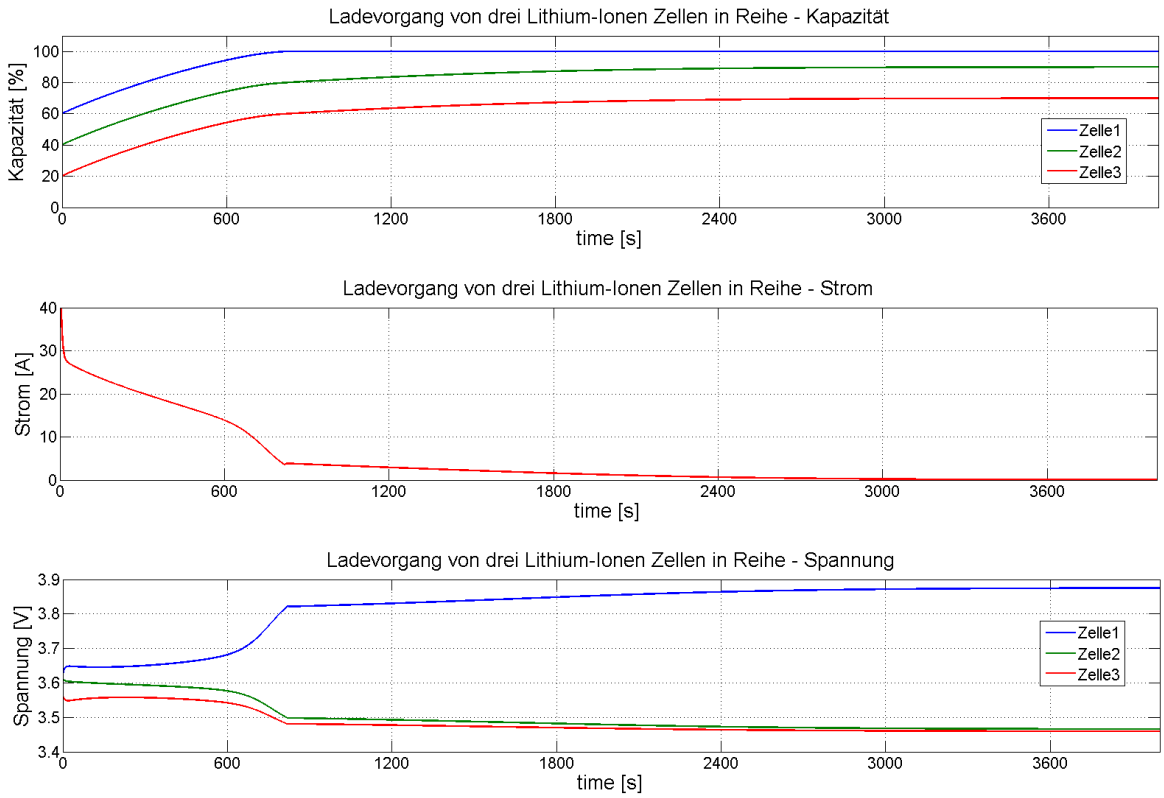


Abbildung 4.14: Simulationsergebnisse für den Ladevorgang von drei in Reihe geschalteten Lithium-Ionen Zellen

Lithium-Ionen Zellen haben über einen weiten Bereich eine gleichbleibende Spannung, deshalb ist es auch schwer, von der Spannung auf die Kapazität zu schließen. Nur am Anfang und Ende der Kapazität verändert sich die Spannung stark. Dies ist auch gut bei Zelle 1 (blau) zu sehen. Ab einer Kapazität von 85% steigt die Spannung an der Zelle stark an. Zum gleichen Zeitpunkt sinkt die Spannung an Zelle 2 und Zelle 3. Da die Gesamtspannung der Batterie annähernd gleich der Ladespannung ist, verringert sich der Strom. Als Ergebnis ist festzustellen, dass eine Zelle längst voll ist und immer mit Strom durchflossen wird und zwei Zellen die nie voll geladen sein werden. Es findet ganz klar eine Überbeanspruchung der Zelle 1 statt.

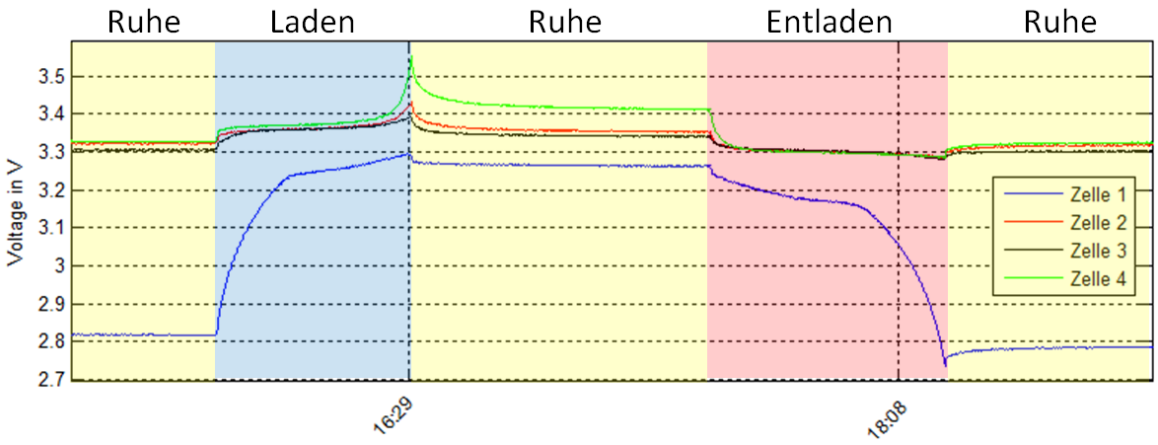


Abbildung 4.15: Messung eines 4 Zellen Lithium-Ionen Akkuspack

Johannes Röhn hat in seiner Thesis [7, S.95] einige Zyklierungen mit einem 4-Zellen-Lithium-Ionen Akkupack durchgeführt. Abbildung 4.15 zeigt einen aufbereiteten Ausschnitt aus dieser Zyklierung. Zelle 1 (blau) hat den geringsten Ladestand, Zelle 2 (gelb) den höchsten. Beim Laden des Akkus beendet die Zelle mit der höchsten Kapazität den Ladevorgang. Die Zelle mit der geringsten Kapazität beendet den Entladevorgang. Zelle 2 (rot) und Zelle 3 (schwarz) werden weder voll geladen, noch leer entladen. Es findet bei der Zyklierung eine einseitige Belastung auf Zelle 1 und Zelle 2 statt. Ohne die festgelegten spannungsabhängigen Schaltmomente des Zykliersystems würde der Lithium-Ionen Akkupack kaputt gehen.

Aus diesem Grund ist beim Zyklieren von Lithium-Ionen Batterien nur mit spannungsabhängigen Schaltmomenten zu arbeiten. Dass die Zellen die gleiche Kapazität haben, ist nur von kurzer Dauer, da durch mehrmaliges Zyklieren die Zellen langsam aus dem Gleichgewicht kommen.

4.4 Messplan

Um die Zellen systematisch über einen langen Zeitraum zu messen, wird ein Messplan erstellt. Dieser Messplan beinhaltet zunächst die zwei Messungen der eben vorgestellten Messkonzepte. Die bereits gesammelten Erfahrungen mit den Messkonzepten zeigt, dass für eine Zellprofilmessung eine ganze Woche benötigt wird. Bei der NEFZ-Messung sind drei pro Woche möglich. Zum Messplan gehören zwei Teile, zum einen ein Zellkatalog, in dem alle Zellen, die in den Messplan verwendetet werden, aufgeführt sind. Der zweite Teil ist der Messplan mit einer wöchentlichen Auflösung. Beide Teile sind in einer Excel-Tabelle enthalten und miteinander verknüpft. Alle Zellen, die im Zellenkatalog eingetragen sind, erscheinen auch im Messplan mit ihrer Identifikationsbezeichnung.

Identität	Hersteller	Bezeichnung	Typ	Kapazität	Nominale Spannung	Ladeschluss Spannung	Dauer Entladestrom	max. Ladestrom
Lin1	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin2	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin3	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin4	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C

Abbildung 4.16: Zellkatalog

Abbildung 4.16 zeigt diesen Zellkatalog. Er enthält alle wichtigen Eckdaten zu jeder aufgeführten Zelle. Das erleichtert das Einstellen der Messumgebung und es wird immer mit den gleichen Spezifikationen gearbeitet. Der Zellkatalog ist ebenfalls im Anhang D.2 abgebildet.

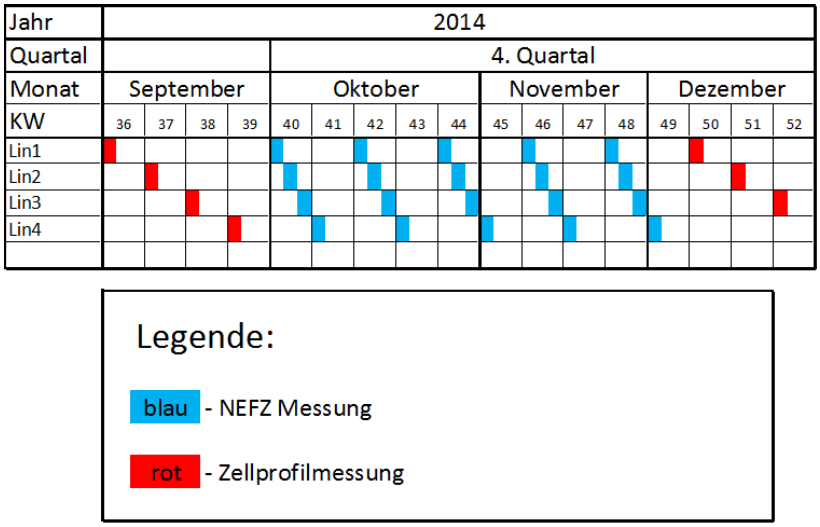


Abbildung 4.17: Ausschnitt des Messplans

In Abbildung 4.17 ist ein Ausschnitt aus dem diesjährigen Messplan zu sehen. Von den roten Feldern, die die Zellprofilmessung repräsentieren, befinden sich pro Kalenderwoche nur ein Eintrag. Von den blauen Feldern sind in einer Kalenderwoche 3 Eingetragen. Sie stehen für die Messung nach dem NEFZ-Messkonzept. Die Anzahl der Messungen an einer Zelle pro Jahr richtet sich nach der Anzahl der Zellen, die im Messplan berücksichtigt werden. Das liegt daran, dass mit dem vorhandenen Equipment sind nicht mehr Messungen zu realisieren.

Die Hoffnung ist, dass nach langer Nutzung der Zellen durch das NEFZ-Messkonzept eine Veränderung der Zellen festgestellt werden kann. Dieser Alterungsprozess soll durch die Zellprofilmessung erfasst werden. Aus diesen Messungen sind Rückschlüsse auf die Lebensdauer der Batterien in der E-Mobility Branche möglich. Im Anhang D.1 ist der Gesamtmessplan abgebildet.

5 Schnittstellen-Optimierung und Dokumentation

Basierend auf den Voruntersuchungen aus Kapitel 2.3 werden jetzt die Schnittstellen und die Dokumentation optimiert. Beide Optimierungen sollen die Handhabung des Zykliersystems vereinfachen. Hinzu kommen die Überwachungsmöglichkeiten, die ein Fernzugriff über die LAN-Schnittstelle gestatten.

5.1 Inbetriebnahme der LAN-Schnittstelle an der HAW

Wie bereits festgestellt wurde, funktioniert die Einbindung des Zykliersystems im HAW-Netzwerk nicht. Nach einem längeren Schriftverkehr mit der Administration der HAW liegt der Schluss nahe, dass das Gerät vom DHCP-Server abgewiesen wird, da seine MAC-Adresse dem Server nicht bekannt ist. Da es sich bei bei dem Zykliersystem um zwei existierende Prototypen handelt, ist es überstürzt, sich beim IEEE „Institute of Electrical and Electronics Engineers“ einen neuen MAC-Adressraum zu kaufen. Der preiswerteste kostet derzeit 625$. Deshalb wird sich für die Alternative entschieden, die MAC-Adresse aus einer alten Netzwerkkarte auszulesen und sie anschließend zu vernichten.

Beide Zykliersysteme besitzen derzeit die gleiche MAC-Adresse. Deshalb ist es nicht ratsam sie gleichzeitig im HAW-Netzwerk einzubinden. Herr Peter Berner ist zuständig für die Vergabe von IP-Adressen im HAW-Netzwerk. Durch die Mitteilung der MAC-Adresse an die Administration bekommt das Zykliersystem eine feste IP-Adresse vom DHCP-Server. Mit dieser IP-Adresse und dem richtigen Port ist das Zykliersystem im HAW-Netzwerk ansprechbar.

HAW-Netzwerkinformation im Überblick:

MAC-Adresse: 00:10:A7:18:0E:F5
IP-Adresse: 141.22.14.199
TCP-Port: 56936

5.2 Vorbereitung der RS232-Schnittstelle

Um nicht die bestehende galvanische Trennung zwischen dem Zykliersystem und dem Hausnetz aufzuheben, muss die RS232 über einen Optokoppler nach außen geführt werden, da bei einer 9-poligen RS232 Schnittstelle GND über Pin5 verbunden wird. Der verwendete Optokoppler ist in Abbildung 5.1, unter der Bezeichnung „ROLINE RS232 Optokoppler“, zu sehen.

Abbildung 5.1: ROLINE RS232 Optokoppler

Der Optokoppler hat zwei RS232 Anschlüsse. Die eine Seite ist die Device Seite und hat eine DCE Pinbelegung, die andere Seite ist die Host Seite mit einer DTE Pinbelegung. Die Stromversorgung erhält der Optokoppler durch den Host. Weil das Bauteil in das Gehäuse des Zykliersystems montiert werden soll und auch von dort seine Stromversorgung erhält, wird das Zykliersystem automatisch zum Host. Die Geräte, die an das Zykliergerät angeschlossen werden, sind somit das Device.

Auf der Device Seite ist die Lösung einfach. Da der Device Ausgang eine DCE Pinbelegung ist und viele Geräte, wie beispielsweise die Kühlkammer DCE Anschlüsse besitzen, ist hier ein einfaches Null Modem Kabel, das DCE mit DCE Anschlüsse verbindet, anzuwenden.

Auf der Host Seite ist die Lösung komplizierter, da die Pinbelegung auf dem Mainboard weder DCE noch DTE entspricht. Um eine Verbindung vom Mainboard zur Host Seite des Optokopplers zu bekommen, muss ein Kabel angefertigt werden.

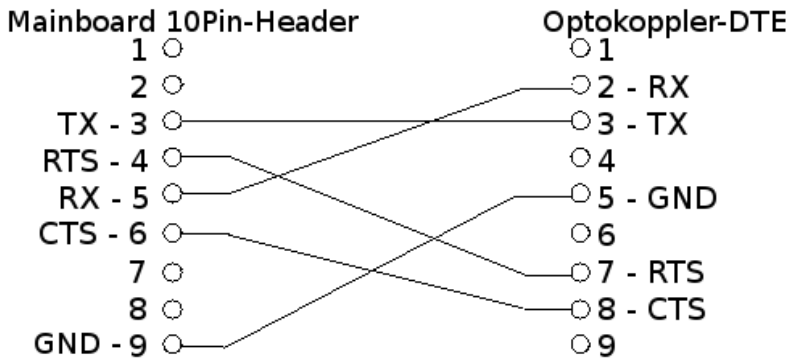


Abbildung 5.2: Pinbelegung des Kabels zwischen Mainboard und RS232 Optokoppler

Mit dem Kabel, das in Abbildung 5.2 dargestellt ist, lässt sich eine Verbindung zum Optokoppler herstellen. Die Stromversorgung wird direkt vom Schaltnetzteil hinter dem Ein-/ Ausschalter abgegriffen. Des Weiteren ist zu beachten, dass auf dem Mainboard JP7 überbrückt ist. JP7 schließt die Stromversorgung für den Leitungstreiber IC des RS232 Ports.

Jetzt ist der RS232 Port galvanisch getrennt und kann somit programmiert werden. In der bestehenden Programmierung wurde er noch nicht benutzt.

5.3 Dokumentation

Die aktuellste Dokumentation ist die „Kurzanleitung für den Batterie Zyklierprüfstand“ [8]. Sie wurde von Herr Wisniewski erstellt und von Herrn Röhn zuletzt bearbeitet. Nun muss diese Kurzanleitung durch die Modifikationen am Zykliersystem angepasst und erweitert werden. Die bereits existierende schlanke Kurzanleitung soll weiterhin übersichtlich und somit schnell von jedermann zu benutzen sein. Da das Auswechseln und Kalibrieren viel Zeit in Anspruch nimmt, werden diese Themen im folgenden Kapitel behandelt.

5.3.1 Das Auswechseln des ADCs

Beim Auswechseln des ADCs auf der Schutzschaltung ist auf die korrekte Einbaulage zu achten, da es ansonsten zur sofortigen Zerstörung des ADCs kommt.

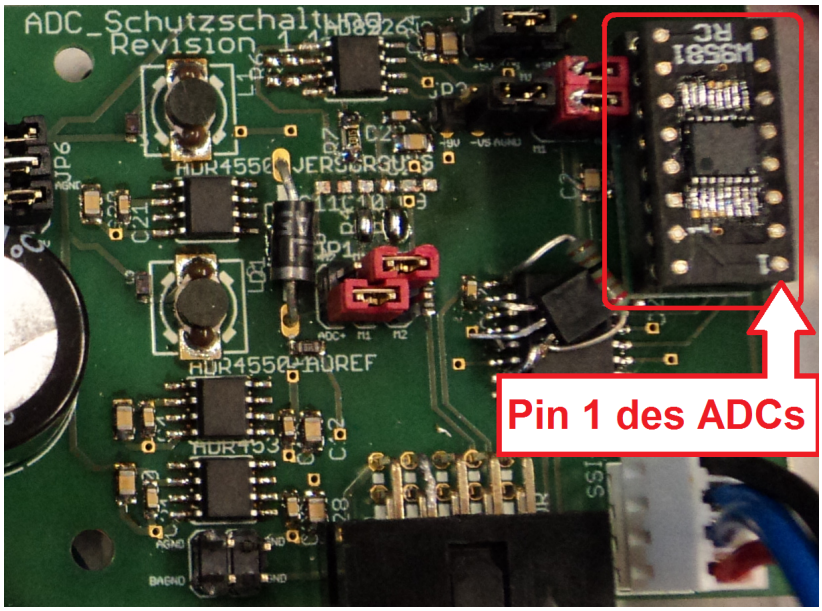


Abbildung 5.3: Einbaurichtung des ADCs in der Schutzschaltung

Abbildung 5.3 zeigt die korrekte Einbaulage. Wenn die ADC Schutzschaltungsplatine lesbar ausgerichtet ist, muss Pin 1 des ADCs sich unten rechts befinden. Durch den Sockel, auf dem sich der ADC befindet, ist ein Austausch einfach und unproblematisch.

5.3.2 Kalibrierung

Nach dem Auswechseln gewisser Bauteile wie z.B. ADC, Sicherung oder Relais, muss eine Kalibrierung durchgeführt werden. Zur Zeit existieren zwei Arten der Kalibrierung, die Lineare und die Quadratische. Die lineare Kalibrierung braucht zwei Messpunkte und die quadratische Kalibrierung braucht drei Messpunkte. Gewählt werden Messpunkte in den Spannungsbereichen, in denen Später häufig gemessen werden soll, dann ist die Genauigkeit nach der Kalibrierung am höchsten. Nickel-Cadmium Zellen und Nickel-Metallhydrid Zellen haben eine Nennspannung von 1,2V, während Blei-Säure Zellen eine Nennspannung von 2V und Lithium-Ionen Zellen eine Nennspannung von 3,2V haben.

Die Kalibrierung ist in drei Schritte unterteilt:

1. Aufnahme der Messpunkte
2. Berechnung der Koeffizienten
3. Einbetten der neuen Koeffizienten

1. Die Aufnahme der Messpunkte
Zur Kalibrierung wird das „FLUKE 45“ Multimeter eingesetzt, da es eine Genauigkeit von 1µV besitzt. Als Spannungsquelle dient das „ROHDE & SCHWARZ NGT35“. Mit der Funktion *adc_calibrate()* werden nun 20 Werte pro Messpunkt aufgenommen, d.h. bei der linearen Kalibrierung werden 20 Messwerte bei 1,2V und 20 Messwerte bei 3,2V aufgenommen. Bei der quadratischen Kalibrierung sind das bei 1,2V, 2V und bei 3,2V jeweils 20 Messwerte. Dies wird für jeden der 12 Kanäle durchgeführt. Aus allen 20 Messwerten pro Messpunkt wird der arithmetische Mittelwert gebildet. Am Ende existiert zu jedem Messpunkt ein Messwert. Die aufgenommenen Messwerte sind die direkte codierte Ausgabe des ADCs, diese Werte besitzen keine Einheit.

2.A Die Berechnung der Koeffizienten für die lineare Kalibrierung
Bei einem Angleich zwischen den idealen Messpunkten und den codierten Messwerten des ADCs ergibt sich folgende lineare Funktion:

$Code$: codierten Messwerten des ADCs, U_{MP}: idealer Messpunkt,
$Gain$: Verstärkungsfaktor, $Offset$: Verschiebungsfaktor

$$U_{MP} = Gain \cdot Code + Offset \tag{5.1a}$$

Da der µC intern den Messpunkt in µV berechnet, muss dies berücksichtigt werden.

$$U_{MP} \cdot 10^6 = Gain \cdot Code + Offset \tag{5.1b}$$

Da der $Gain$ und $Offset$ unbekannt sind, müssen zwei Gleichungen aufgestellt werden, dazu werden die zwei Messpunkte benutzt.

$$U_{MP1} \cdot 10^6 = Gain \cdot Code_1 + Offset \tag{5.1c}$$
$$U_{MP2} \cdot 10^6 = Gain \cdot Code_2 + Offset \tag{5.1d}$$

Wenn nun das Gleichungssystem in eine Matritzen- und Vektorenschreibweise überführt wird, lässt es sich einfach mit MATLAB lösen. Des Weiteren ist diese Methode leicht übertragbar auf Kalibrierungen mit mehr als zwei Messpunkten.

$$\begin{pmatrix} U_{MP1} \cdot 10^6 \\ U_{MP2} \cdot 10^6 \end{pmatrix} = \begin{pmatrix} Code_1 & 1 \\ Code_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} Gain \\ Offset \end{pmatrix} \tag{5.1e}$$

Die Lösung des Gleichungssystems lautet:

$$\begin{pmatrix} Gain \\ Offset \end{pmatrix} = \begin{pmatrix} Code_1 & 1 \\ Code_2 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} U_{MP1} \cdot 10^6 \\ U_{MP2} \cdot 10^6 \end{pmatrix} \tag{5.1f}$$

Diese Berechnung wird für alle 12 Kanäle durchgeführt.

2.B Die Berechnung der Koeffizienten für quadratische Kalibrierung
Bei der quadratischen Kalibrierung gibt es insgesamt drei Koeffizienten. Deshalb werden drei Messpunkte gebraucht, mit deren Hilfe können drei Gleichungen aufgestellt werden. Ansonsten ist die Berechnungsmethode der Koeffizienten ähnlich der linearen Kalibrierung.

$Code$: codierte Messwerte des ADCs, U_{MP}: idealer Messpunkt,
A: erster Koeffizient, B: zweiter Koeffizient, C: dritter Koeffizient

$$U_{MP} = A \cdot Code^2 + B \cdot Code + C \tag{5.2a}$$

Da der µC intern den Messpunkt in µV berechnet, muss dies berücksichtigt werden.

$$U_{MP} \cdot 10^6 = A \cdot Code^2 + B \cdot Code + C \tag{5.2b}$$

Mit den drei Messpunkten werden jetzt drei Gleichungen gebildet.

$$U_{MP1} \cdot 10^6 = A \cdot Code_1^2 + B \cdot Code_1 + C \tag{5.2c}$$
$$U_{MP2} \cdot 10^6 = A \cdot Code_2^2 + B \cdot Code_2 + C \tag{5.2d}$$
$$U_{MP3} \cdot 10^6 = A \cdot Code_3^2 + B \cdot Code_3 + C \tag{5.2e}$$

Überführung in eine Matritzen- und Vektorenschreibweise.

$$\begin{pmatrix} U_{MP1} \cdot 10^6 \\ U_{MP2} \cdot 10^6 \\ U_{MP3} \cdot 10^6 \end{pmatrix} = \begin{pmatrix} Code_1^2 & Code_1 & 1 \\ Code_2^2 & Code_2 & 1 \\ Code_3^2 & Code_3 & 1 \end{pmatrix} \cdot \begin{pmatrix} A \\ B \\ C \end{pmatrix} \tag{5.2f}$$

Die Lösung des Gleichungssystems lautet:

$$\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} Code_1^2 & Code_1 & 1 \\ Code_2^2 & Code_2 & 1 \\ Code_3^2 & Code_3 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} U_{MP1} \cdot 10^6 \\ U_{MP2} \cdot 10^6 \\ U_{MP3} \cdot 10^6 \end{pmatrix} \tag{5.2g}$$

Diese Berechnung wird für alle 12 Kanäle durchgeführt.

3. Das Einbetten der neuen Koeffizienten
Die neu berechneten Koeffizienten müssen in der Struktur config in der „config.c“ zugewiesen werden. Dabei ist darauf zu achten, dass es sich hierbei immer um positive Integer handelt. Sollte ein Gleitkommakoeffizient existieren, muss dieser mit einem Faktor multipliziert werden, der diesen Koeffizient in einen Integer umwandelt. Der Kehrwert des Faktors ist anschließend in der Kalibrierung zu berücksichtigen. Diese befindet sich in der Funktion „*adc_get_voltage(unsigned int choose)*“ in der Datei „myadc.c“. Des Weiteren sind die Koeffizienten auch in der Konfigurationsdatei „config.txt“ einzubetten.

6 Erprobung

Am Ende der Thesis wird die Funktionalität des Zykliersystems erprobt. Dazu werden ausgewählte Zellen zykliert und die ADC Schutzschaltung auf eine ausreichende Genauigkeit hin geprüft. Die RS232 und LAN Schnittstelle werden mit der Terminal Software „RealTerm“ und dem Terminal von „Code Composer Studio 5.5“ getestet. Auftretende Probleme während der Erprobung werden ebenfalls dargestellt.

6.1 Schnittstellen

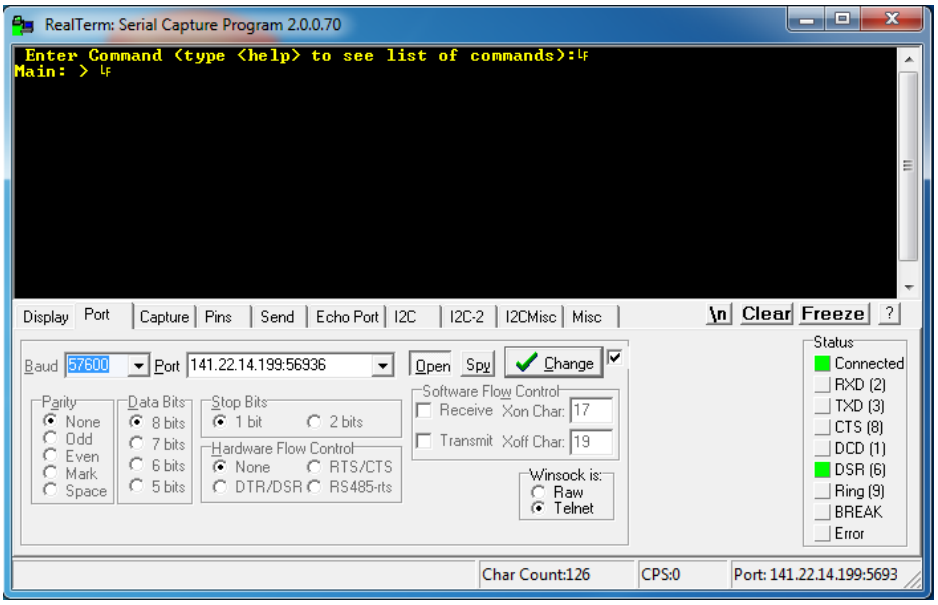


Abbildung 6.1: Verbindung zum Zykliersystem über die Ethernet-Schnittstelle im HAW-Netzwerk

Abbildung 6.1 zeigt die aufgebaute Verbindung zum Zykliersystem über das Ethernet im HAW-Netzwerk. Auch die Nutzung über eine VPN Verbindung ins HAW-Netzwerk verlief problemlos. Die Anleitung zum Verbindungsaufbau befindet sich in der aktuellen „Kurzanleitung für den Batterie-Zyklierprüfstand“, Anhang F.1.

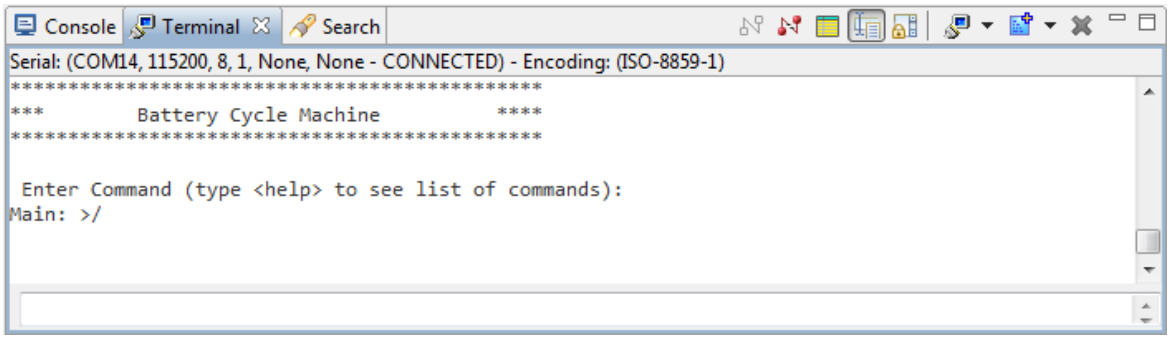


Abbildung 6.2: Verbindung zum Zykliersystem über die RS232-Schnittstelle

Um die RS232 Schnittstelle zu verifizieren, wird die UART-Ausgabe von UART0 auf UART2 umgeleitet. Üblicherweise kommuniziert UART0 des µC über den FTDI-Chip „FT2232D“ und der USB Schnittstelle mit dem PC, er liegt auf dem COM-Port 11. UART2 des µC läuft über einen Leitungstreiber, den neu implementierten Optokoppler und der RS232 Schnittstelle zum PC, dieser liegt am COM-Port 14. Abbildung 6.1 zeigt das Terminal von „Code Composer Studio 5.5“, dass gerade eine Verbindung über den COM-Port 14 (UART2) aufgebaut hat.

Die LAN und die RS232 Schnittstelle wurden erfolgreich getestet, sie stehen nun zur weiteren Verwendung zur Verfügung.

6.2 Genauigkeit der ADC Schutzschaltung

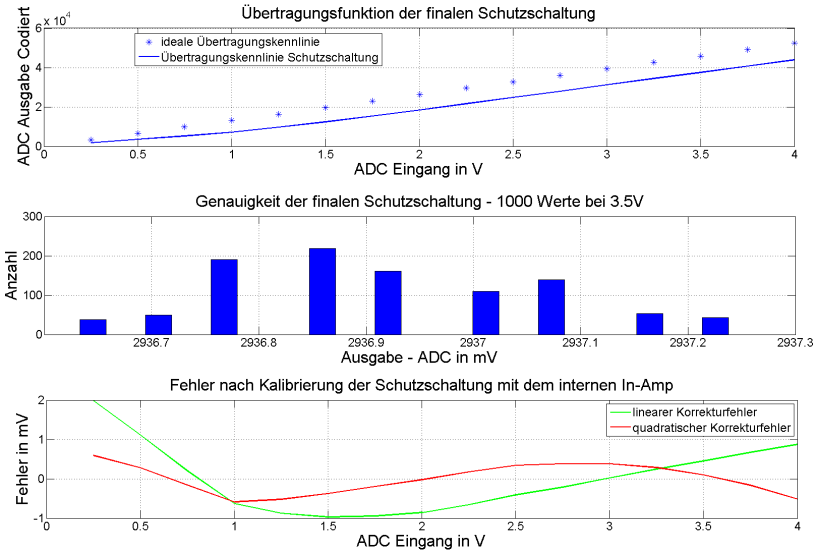


Abbildung 6.3: Genauigkeitsüberprüfung der finalen ADC Schutzschaltung

Wie schon beim Verifizieren der verschiedenen Schutzschaltungen in Kapitel 3.2 wird auch bei der finalen Schutzschaltung die Genauigkeit überprüft. Bei der überprüften Schutzschaltung handelt es sich um die, mit dem externen Instrumentenverstärker „AD8226“. Die Ergebnisse dieser Messung sind in Abbildung 6.3 dargestellt.

Die Genauigkeit liegt bei 1mV und hat sich damit um den Faktor 4 seit der Entwickelung der Schutzschaltung verbessert. Dies liegt an der eigenen Stromversorgung und der galvanischen Trennung der Schutzschaltung. Trotzdem kommt der ADC ohne Instrumentenverstärker auf eine höhere Genauigkeit. Ohne diesen, nur mit der Suppressordiode, besitzt der ADC mit der vereinfachten Schutzschaltung eine Genauigkeit von 0,2mV.

6.3 Messumgebung

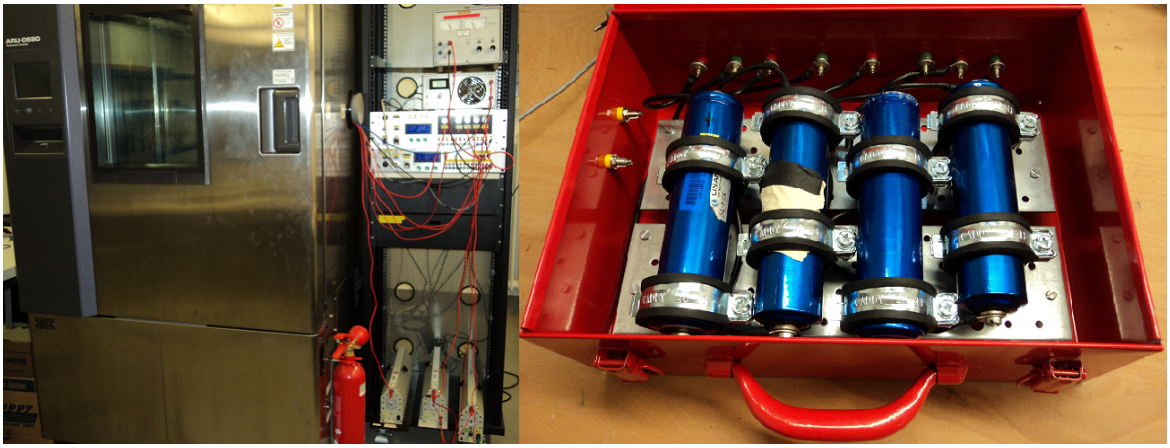


Abbildung 6.4: links:Laboraufbau während den Zyklierungen
rechts: Messkoffer mit vier Lithium-Ionen Zellen

Da die Zyklierung mit Lithium-Ionen Zellen stattfindet, bei denen stets die Gefahr eines „thermal runaway“ existiert, muss für eine sichere Messumgebung gesorgt werden. Für diesen Fall wird ein Messkoffer gebaut, der vier Zellen aufnehmen kann. Bei allen vier Zellen wird der Plus und Minus Pol der Zellen nach außen geführt. Mit kurzen Laborkabeln können sie in Reihe geschaltet werden. Der Messkoffer ist in Abbildung 6.4 rechts zu sehen.
Links in Abbildung 6.4 ist die komplette Messumgebung zu sehen. Zu weiteren Sicherheit befindet sich in unmittelbarer Nähe ein Feuerlöscher. Für die Messung steht ein Temperaturschrank, eine elektronische Last, eine Stromquelle und drei einstellbare Hochleistungswiderstände zur Verfügung.

6.4 Zyklierung ausgewählter Zellen

Nun wird das NEFZ-Messkonzept mit dem Zykliersystem erprobt. Dafür werden Lithium-Ionen Zellen der Firma „Headway“ ausgewählt. Da zurzeit das Zykliersystem nur minütlich die Relais schalten kann, ist eine Vereinfachung des NEFZ-Messkonzepts nötig. Dazu wurden zwei unterschiedliche Vereinfachungen untersucht.

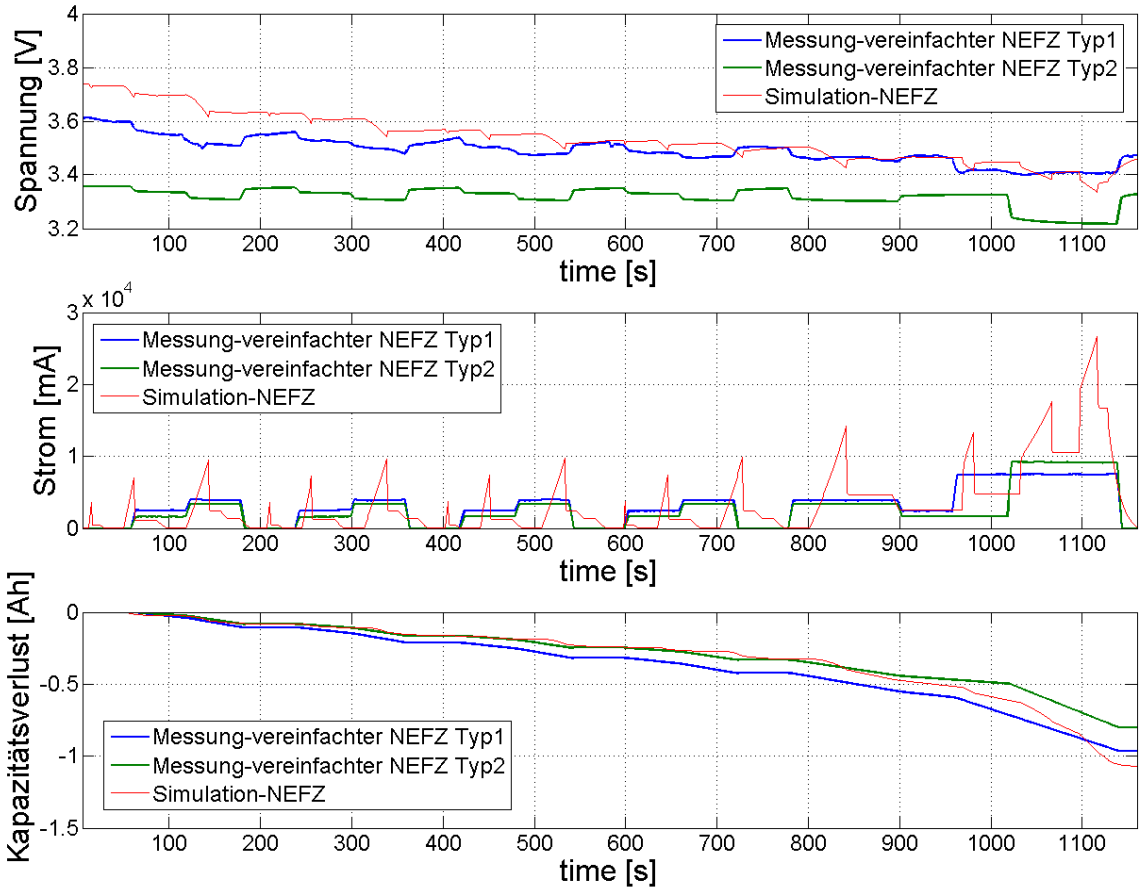


Abbildung 6.5: Messergebnisse zweier NEFZ-Annäherungen

In Abbildung 6.5 sind beide vereinfachten NEFZ-Messungen dargestellt. Sie wurden mit dem Zykliersystem aufgezeichnet und mit dem eigentlichen NEFZ-Messkonzept zum Vergleich dargestellt.

Werden beide vereinfachte NEFZ-Messkonzepte verglichen, fällt auf, dass beim Spannungsverlauf kaum ein Unterschied zwischen beiden zu sehen ist. Die Differenz der Spannungspegel lässt sich durch den unterschiedlichen Ladezustand erklären. Beim Strom hingegen ist eine Veränderung im Zyklus außerorts zu erkennen. Beide Varianten nähern sich vom Energieverbrauch der Simulation an.

Bei den weiteren Zyklierungen wird Typ2 benutzt, da er mehr Strom im Zyklus außerorts benötigt und dies den NEFZ am besten annähert. Mit dieser Variante wird nun bei drei verschiedenen Temperaturen die Zyklierung vollzogen.

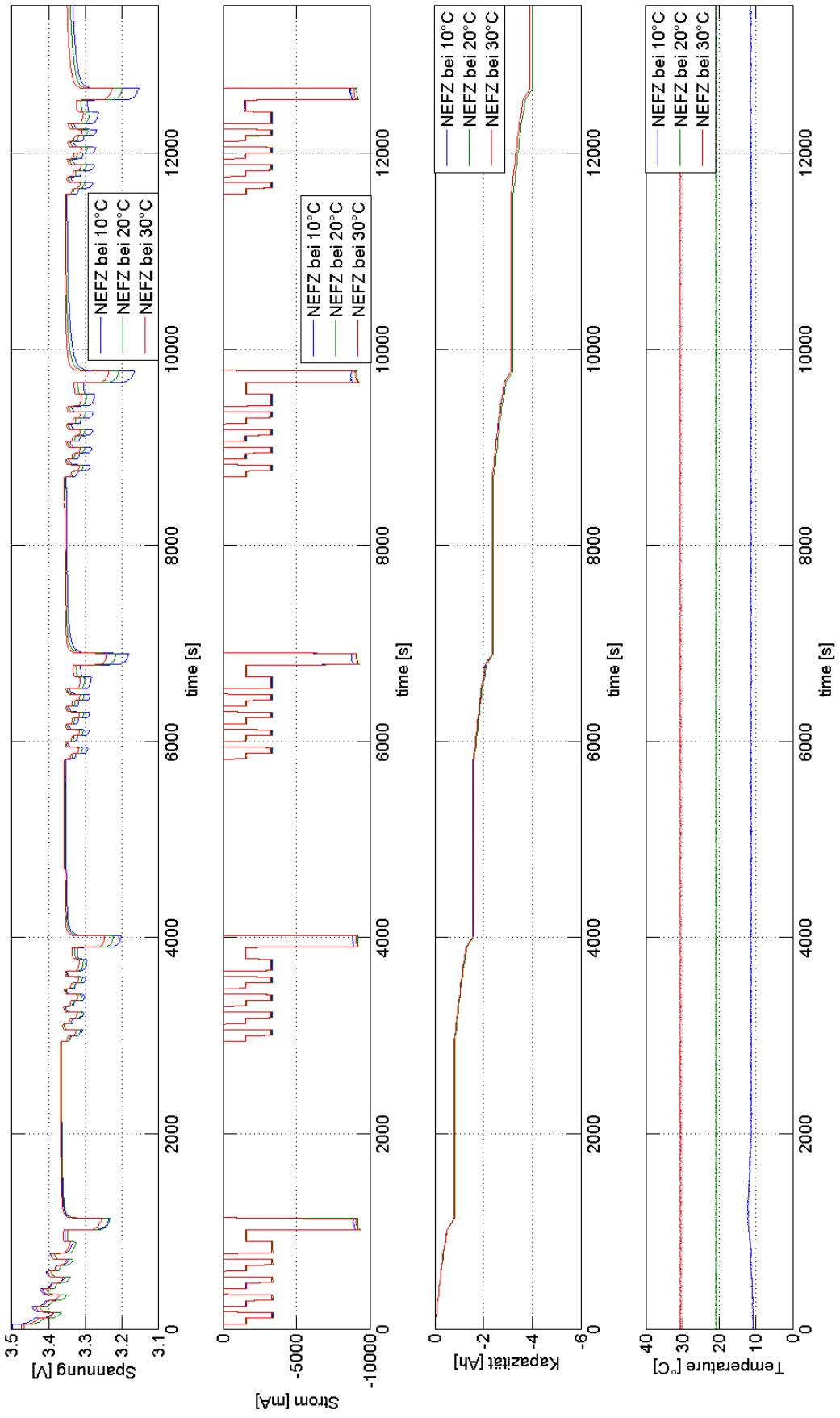


Abbildung 6.6: NEFZ bei verschiedenen Temperaturen

In Abbildung 6.6 sind die Messergebnisse des Zykliersystem dargestellt. Bei drei verschiedenen Temperaturen wurde mit der gleichen Zelle nach einer Aufladung bei $20°C$ fünfmal hintereinander der NEFZ durchlaufen. Zwischen jedem NEFZ gibt es eine halbe Stunde Pause. Bei jedem NEFZ sind deutlich die vier Zyklen innerorts und der Zyklus außerorts zu erkennen.

Wird der Spannungsverlauf betrachtet, so ist festzustellen, dass bei niedrigen Temperaturen die Spannung bei Belastung deutlich tiefer einbricht. Dieser Effekt ist besonders gut bei den Zyklen außerorts zu erkennen. Des Weiteren ist auch der rapide Abfall der Spannung nach dem ersten NEFZ erkennen, das ist typisch für eine Lithium-Ionen Zelle. Am Anfang und am Ende seiner Kapazität verändert sich die Spannung am meisten. Deshalb kann bei einer Lithium-Ionen Zelle aus der Spannung nur schwer auf ihre Kapazität geschlossen werden.

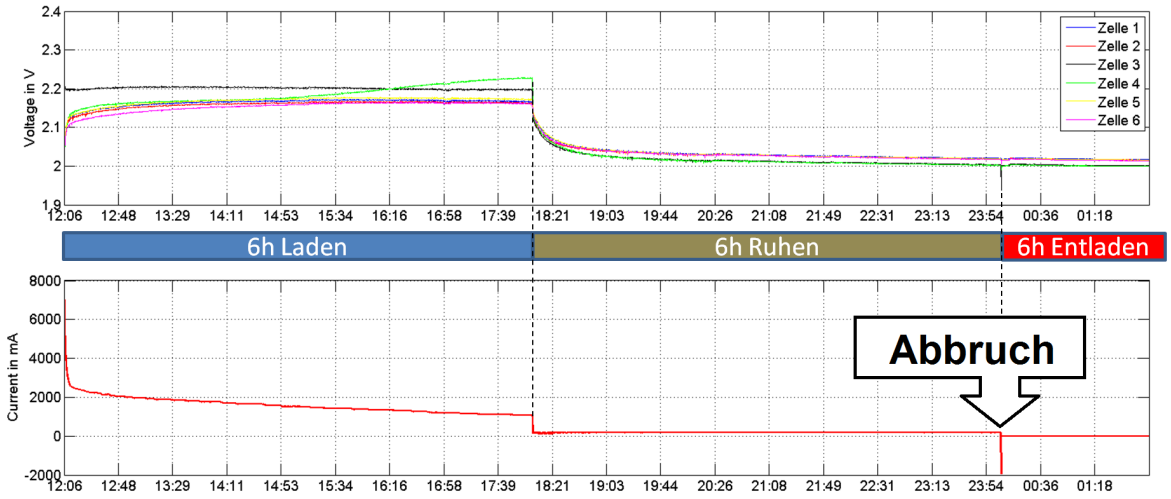


Abbildung 6.7: Abbruch der Zyklierung durch das Messen im Umschaltmoment

Bei der Zyklierung einer Blei-Säure Batterie, bestehend aus sechs Zellen, kam es bei der Erprobung zum vorzeitigen Abbruch. Dieser Fall ist in Abbildung 6.7 dargestellt. Nach einer sechsstündigen Ruhephase soll die Batterie entladen werden. Direkt beim Schalten des Relais wird nun gemessen. Da die Spannung kurz einschwingt, befindet sie sich außerhalb der eingestellten zulässigen Spannungsgrenzen und führt zum Abbruch der Messung. Dieses Problem tritt nur sporadisch auf, da der Schaltmoment des Relais ständig variiert.

7 Fazit

Abschließend wird nun die Aufgabenstellung aus Anhang A.1 mit der Bachelorarbeit verglichen. Dazu wird zusammenfassend über jede erledigte Aufgabe der Aufgabenstellung berichtet. Anschließend findet eine Auswertung im Hinblick der E-Mobilität statt. Das Ende dieser Arbeit bildet eine Reihe von Vorschlägen zur Weiterentwicklung des Zykliersystems.

7.1 Zusammenfassung

Laut Aufgabenstellung gibt es vier Hauptaufgaben, die zu erledigen waren. Diese werden nun auf die Umsetzung hin überprüft.

1. **Entwicklung einer Schutzschaltung**
   Nach der Analyse möglicher Fehlermöglichkeiten in der ADC-Schaltung wurden fünf verschiedene Schutzschaltungen aufgebaut und diese anschließend bewertet. Basierend auf dieser Bewertung wurde eine Schutzschaltung entwickelt. Sie ist vom Zykliersystem galvanisch getrennt und bietet diverse Einstellmöglichkeiten. Es kann ein externer Instrumentenverstärker in die Messung einbezogen werden, bei dem die Versorgungsspannung frei wählbar ist. Des Weiteren lässt sich die galvanische Trennung optional aufheben. Die Benutzung des externen In-Amps verschafft zusätzliche Sicherheit, wobei allerdings Genauigkeit eingebüßt wird. Mit dem In-Amp wird eine Genauigkeit von 1mV erreicht, ohne ihn beträgt sie 0,2mV. Die Erprobung der Schutzschaltung mit dem externen In-Amp verlief problemlos.

2. **Verbesserung der Kommunikationsschnittstellen**
   Bei der Prüfung der RS232 Schnittstelle fiel auf, dass die Pinbelegung nicht der RS232 Norm entspricht. Hinzu kommt, dass diese Schnittstelle galvanisch getrennt werden muss, bevor sie verwendet werden kann. Um sie für die Kopplung mit der Kühlkammer vorzubereiten, wurde ein Kabel angefertigt, dass das Problem mit der Pinbelegung löst. Für die galvanische Trennung wurde ein Optokoppler benutzt. Um die LAN-Schnittstelle im HAW-Netzwerk zu benutzen, musste die MAC-Adresse vom Zykliersystem angemeldet werden. Dabei erhielt das Zykliersystem die MAC-Adresse von einer ausrangierten Netzwerkkarte.

3. **Planen und Durchführen von Batterieuntersuchungen**
   Es wurden zwei verschiedene Messkonzepte zur Batterieuntersuchung entwickelt, sowie ein Messplan erstellt. Das Messkonzept nach dem NEFZ ist stark an die E-Mobilität angelehnt. Um die Belastung einer Zelle während des NEFZ zu ermitteln, wurden umfangreiche Simulationen durchgeführt. Darauf basierend existiert nun ein vereinfachtes Messverfahren, dass wurde auch bei verschiedenen Temperaturen erfolgreich getestet. Das zweite Messkonzept ist zur Erstellung eines Zellenprofiles entwickelt worden. Es basiert auf Erfahrungen von bereits durchgeführten Zyklierungen. Des Weiteren wurde ein Messplan erstellt, der die beiden Messkonzepte beinhaltet.

4. **Erstellen einer ausführlichen Dokumentation**
   Die erstellte Dokumentation befindet sich an zwei stellen dieser Arbeit. Zum einen gibt es eine ausführliche Beschreibung zum Auswechseln des ADCs, sowie das Kalibrieren in Kapitel 5.3. Die Anleitung zum Verbindungsaufbau über VPN und Ethernet, sowie Beispiele zum korrekten Messen wurden der bereits vorhandenen „Kurzanleitung für den Batterie-Zyklierprüfstand“ angehängt. Diese Kurzanleitung befindet sich im Anhang F.1.

7.2 Vorschläge zur Weiterentwicklung des Zykliersystems

Die hier erwähnten Vorschläge sind während der Durchführung der Arbeit entstanden. Sie dienen als Anhaltspunkte für die Weiterentwicklung des Zykliersystems. Die Vorschläge sind mit hoher und niedriger Priorität unterteilt.

Vorschläge - mit hoher Priorität:

- **Anbindung des Temperaturschrankes über RS232**
  Nach der galvanischen Trennung der RS232-Schnittstelle ist eine Routine zu erstellen, die den Temperaturschrank steuert. Derzeit ist es sehr umständlich eine Zellenprofilmessung durchzuführen, da die Temperatur manuell am Temperaturschrank eingestellt werden muss.
- **Sekundengenaues Schalten der Relais**
  Um beim NEFZ-Messkonzept eine bessere Annäherung an die Simulation zu erhalten, müssen die Relais in der Lage sein, die Widerstände sekündlich umzuschalten.
- **Widerstandsmatrix für das NEFZ-Messkonzept**
  Zurzeit sind drei verschiedene Widerstände schaltbar. Das liegt daran, dass ein Relais zum Laden der Batterie benutzt wird. Die übrigen drei Relais bleiben zum Schalten der Widerstände. Derzeit sind die Relais nur einzeln schaltbar, d.h. das gleichzeitige Schalten von zwei Relais wird nicht unterstützt. Mit einer gut überlegten Widerstandsmatrix und der Unterstützung gleichzeitig schaltender Relais, könnte wiederum eine bessere Annäherung an der Simulation vom NEFZ-Messkonzepts erreicht werden.
- **Auskunft des aktuellen „cycle step“ über Ethernet**
  Ob die Zyklierung noch läuft ist derzeit über einen Fernzugriff mit dem Befehl „alive“ erkennbar, aber leider nicht wo sie sich gerade befindet. Ein zusätzlicher Befehl zur Abfrage des aktuellen „cycle step“ wäre hilfreich.
- **Optional einschaltbare Laderegler**
  Bisher war ein Laderegler nicht erforderlich, da fast ausschließlich Blei-Säure Batterien zykliert wurden. Wenn jedoch Lithium-Ionen Zellen zykliert werden, ist der Einsatz eines Ladereglers sinnvoll. Diese Problematik ist in Kapitel 4.3 genauer erläutert.
- **Komplettes Redesign der Hauptplatine**
  Das letzte erstellte Platinenlayout liegt nun ein Jahr zurück. Seitdem haben die Thesis von Herrn Röhn und auch diese Thesis Veränderungen an der Hauptplatine bewirkt, die durch ein Redesign berücksichtigt werden müssen. Zudem können die externen Optokoppler für USB und RS232 auf der Hauptplatine integriert werden.

Vorschläge - mit niedriger Priorität:

- **Wechsel des ADCs**
  Da sich der ADC „AD7798“ als sehr empfindlich herausgestellt hat und der interne Instrumentenverstärker, sowie der „buffered mode“ unbrauchbar sind, liegt die Entscheidung nahe, einen anderen ADC mit integrierten Überspannungsschutz zu benutzten.
- **Entwickelung eines Rütteltisch**
  Oft wird in den Batterieprüflabore ein Rütteltisch während der Zyklierungen benutzt. Das soll die Autobewegung während einer Fahrt nachahmen. Dies ist auch für unsere Zyklierungen denkbar.
- **Automatische Erkennung der angeschlossenen Zellen**
  Durch die zwei 100nA Stromquellen im ADC besteht die Möglichkeit, am Eingang eine angeschlossene Batteriezelle zu erkennen. Damit lässt sich eine Aussage treffen, wie viele Kanäle von den 12 möglichen belegt sind.

7.3 Auswertung im Hinblick der E-Mobilität

Wie schon in der Einführung 1.3 erwähnt schreitet die E-Mobilität stetig voran. Viele neue Batterieprüflabore wurden in den letzten zwei Jahren von Firmen, wie TÜV SÜD und SGS Germany GmbH aufgebaut. Somit ist das „BATSEN“ Projekt für die Zukunft ausgezeichnet aufgestellt. Werden derzeitige Prüfmethoden der Batterieprüflabore in Deutschland mit den uns zur Verfügung stehenden Mitteln verglichen fällt auf, dass die Kapazitäten fehlen, um die selben Prüfungen vorzunehmen. Im Hinblick auf die E-Mobilität hat das „BATSEN“ Projekt allerdings eine forschende und beratende Rolle. Deshalb ist es für das „BATSEN“ Projekt wichtig, sich von den bestehenden Batterieprüflaboren abzugrenzen.
Mit dem entwickelten NEFZ-Messkonzept ist dies auch gelungen, da solch ein Konzept noch nicht existiert. Auch der Ausbau des Zykliersystems ist unter diesen Gesichtspunkten ein Fortschritt, da es eine Eigenentwicklung innerhalb des „BATSEN“ Projekts darstellt.

Tabellenverzeichnis

Abbildungsverzeichnis

Glossar

µC Mikrocontroller

ADC Analog to Digital Converter

BATSEN Drahtlose Zellensensoren für Fahrzeugbatterien

CMRR Common-Mode Rejection Ratio

CS Chip select

CTS Clear to Send

DAT Deutsche Automobil Treuhand GmbH

DCE Data Communication Equipment

DHCP Dynamic Host Configuration Protocol

DTE Data Terminal Equipment

ESD electrostatic discharge

EWG Europäischen Wirtschaftsgemeinschaft

GLS Gleichungssystem

HAW Hamburg Hochschule für Angewandte Wissenschaften Hamburg

IEC International Electrotechnical Commission

IEEE Institute of Electrical and Electronics Engineers

In-Amp Instrumentation Amplifier

LAN Local Area Network

LI-ION Lithium-Ionen

LiFePO4 Lithium-Eisenphosphat

lwIP Lightweight TCP/IP stack

NEFZ Neuer Europäischer Fahrzyklus

Qucs Quite Universal Circuit Simulator

RS232 Recommended Standard 232

RTS Ready to Send

SMD Surface-Mounted Device

SOC State of Charge

SOH State of Health

SPI Serial Peripheral Interface

TCP Transmission Control Protocol

TI Texas Instruments

UART Universal Asynchronus Receiver Transmitter

UL Underwriters Laboratories

USB Universal Serial Bus

Literaturverzeichnis

[1] WISNIEWSKI, Thomas: *Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2013

[2] HILLERMANN, Lars: *Starterbatterie in Lithium-Eisen-Phosphat-Technologie parallele Zellenmodule mit Überwachungs- und Leistungselektronik*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit, 2012

[3] LOSCHWITZ, Rico: *Überwachungs-, Zellenbalancierungs- und Leistungselektronik für eine Starterbatterie in Lithium-Eisen-Phosphat-Technologie*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2013

[4] TEXAS INSTRUMENTS (Hrsg.): *Stellaris® LM3S9D92 Microcontroller DATA SHEET*. TI: TEXAS INSTRUMENTS, Version: 2012. http://www.ti.com/lit/ds/symlink/lm3s9d92.pdf. – Abruf: 02.06.2013

[5] SCHLÜTER, Jan: *Entwurf und Realisierung eines modular aufgebauten Experimentierboards für Mikrocontroller*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2013

[6] STEINMANN, Tobias: *Hard- und Softwareentwicklung für einen Controller-gesteuerten, vernetzten Zellspannungsgenerator*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2012

[7] RÖHN, Johannes: *Batteriezellen-Zyklierprüfstand mit ARM-Mikrocontroller Software sowie Mess- und Leistungsschaltung*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorarbeit, 2014

[8] BATSEN (Hrsg.): *Kurzanleitung Batteriezellen-Zyklierprüfstand*. : BATSEN, Version: 2014

[9] KIENBAUM CONSULTANTS INTERNATIONAL GMBH (Hrsg.): *Kienbaum-Studie widerlegt Abgesang auf die Elektromobilität*. : Kienbaum Consultants International GmbH, Version: 2014. http://www.kienbaum.de/desktopdefault.aspx/tabid-83/154_read-607/153_read-237/. – Abruf: 05.07.2014

[10] BP (Hrsg.): *Statistical Review of World Energy 2013*. : BP, Version: 2013. http://www.bp.com/content/dam/bp/pdf/statistical-review/statistical_review_of_world_energy_2013.pdf. – Abruf: 05.07.2014

[11] AUTO MOTOR UND SPORT (Hrsg.): *VW und Byd planen Elektro-Kooperation*. : AUTO MOTOR UND SPORT, Version: 2009. http://www.auto-motor-und-sport.de/eco/lithium-ionen-batterien-von-byd-vw-und-byd-planen-elektro-kooperation-1268301.html. – Abruf: 17.07.14

[12] TÜV SÜD (Hrsg.): *Batterieprüfungen*. : TÜV SÜD, Version: 2014. http://www.tuev-sued.de/home-de/fokus-themen/das-potential-der-elektromobiltaet/batteriepruefungen. – Abruf: 17.07.14

[13] SGS (Hrsg.): *PRÜF- UND ZULASSUNGSSERVICES FÜR LITHIUM-IONEN-BATTERIEN*. : SGS, Version: 2012. http://www.sgsgroup.de/~/media/Local/Germany/Documents/Brochures/Auto/sgs-cqe-batterytesthouse-db-a4-de-0312.pdf. – Abruf: 17.07.14

[14] EWG (Hrsg.): *91/441/EWG*. : EWG, Version: 1991. http://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:31991L0441&from=DE. – Abruf: 22.07.14

[15] T&E (Hrsg.): *Mind the Gap! Why official car fuel economy figures don?t match up to reality*. : T&E, Version: 2013. http://www.transportenvironment.org/sites/te/files/publications/Real%20World%20Fuel%20Consumption%20v15_final.pdf. – Abruf: 22.07.14

[16] DEUTSCHE AUTOMOBIL TREUHAND GMBH (Hrsg.): *Leitfaden über den Kraftstoffverbrauch, die CO2-Emissioen und den Stromverbrauch*. : Deutsche Automobil Treuhand GmbH, Version: 2014. http://www.dat.de/uploads/media/LeitfadenCO2.pdf. – Abruf: 22.07.14

[17] ANALOG DEVICES (Hrsg.): *AD7798/7799*. AD: ANALOG DEVICES, Version: 2013. http://www.analog.com/static/imported-files/data_sheets/AD7798_7799.pdf. – Abruf: 23.07.2014

[18] ANALOG DEVICES (Hrsg.): *Chapter II - Inside an Instrumentation Amplifier*. : Analog Devices, Version: 2006. http://www.analog.com/static/imported-files/design_handbooks/5816856680166219537Chapter_II.pdf. – Abruf: 26.07.14

[19] ANALOG DEVICES (Hrsg.): *ADR45X0*. AD: ANALOG DEVICES, Version: 2012. http://www.analog.com/static/imported-files/data_sheets/ADR4520_4525_4530_4533_4540_4550.pdf. – Abruf: 02.08.2014

[20] LITTELFUSE (Hrsg.): *SA5.0A*. : Littelfuse, Version: 2009. http://www.littelfuse.com/data/en/data_sheets/littelfuse_tvs-diode_sa.pdf. – Abruf: 04.08.2014

[21] ANALOG DEVICES (Hrsg.): *AD623*. : Analog Devices, Version: 2008. http://www.analog.com/static/imported-files/data_sheets/AD623.pdf. – Abruf: 06.08.2014

[22] ANALOG DEVICES (Hrsg.): *AD8226*. : Analog Devices, Version: 2012. http://www.analog.com/static/imported-files/data_sheets/AD8226.pdf. – Abruf: 06.08.2014

[23] ANALOG DEVICES (Hrsg.): *AD624*. : Analog Devices, Version: 2013. http://www.analog.com/static/imported-files/data_sheets/AD624.pdf. – Abruf: 06.08.2014

[24] TIETZE, U. ; SCHENK, Ch.: *Halbleiter-Schaltungstechnik*. Springer Verlag, 2009. – ISBN 978–3642016219

[25] TREMBLAY, Olivier ; DESSAINT, Louis-A.: *Experimental Validation of a Battery Dynamic Model for EV*. World Electric Vehicle Journal, 2009. ISSN 2032–6653

[26] VW (Hrsg.): *ELEKTROMOBILITÄT Der e-up! made by Volkswagen*. : VW, Version: 2013. http://www.volkswagenag.com/content/vwcorp/info_center/de/publications/2013/09/VIAVISION7.bin.html/binarystorageitem/file/VIAVISION_D.pdf. – Abruf: 22.08.2014

[27] THEUERKAUF, Judith: *Schreiben im Ingenieurstudium*. Schöningh, 2012. – ISBN 978–3–8252–3644–1

[28] ESPEC CORP. (Hrsg.): *Environmental Stress Chamber AR Series - Communication Function*. TI: ESPEC CORP., Version: 2009

[29] WEYDANZ, Wolfgang ; JOSSEN, Andreas: *Moderne Akkumulatoren richtig einsetzen*. Reichardt Verlag, 2006. – ISBN 978–3–939359–11–1

[30] ERDÖL-VEREINIGUNG / UNION PÉTROLIÈRE (Hrsg.): *Erdöl-Energieverbrauch und Reserven*. : Erdöl-Vereinigung / Union Pétrolière, Version: 2007. http://www.erdoel-vereinigung.ch/UserContent/Shop/EV07_Energieverbrauch_d_RZ.pdf. – Abruf: 05.07.2014

[31] HOCHSCHULE REUTLINGEN (Hrsg.): *Die Batterie als Schlüsseltechnologie für die Elektromobilität der Zukunft*. : Hochschule Reutlingen, Version: 2012. http://www.esb-business-school.de/fileadmin/_research/dokumente/Diskussionsbeitraege/2012-3-Reutlinger-

Diskussionsbeitraege-Mark-Mngmt-E-Mobility-Batterie.pdf. – Abruf: 05.07.2014

[32] VDE PRÜF- UND ZERTIFIZIERUNGSINSTITUT GMBH (Hrsg.): *Prüfanforderungen an Li-Batterien für Elektrofahrzeuge*. : VDE Prüf- und Zertifizierungsinstitut GmbH, Version: 2012. http://www.vde.com/de/Technik/e-energy/Testing/Documents/VDE-Pr%C3%BCfanforderungen%20an%20Li-Batterien%20f%C3%BCr%20Elektrofahrzeuge.pdf. – Abruf: 17.07.14

Anhang

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

17. Juni 2014

Bachelorarbeit: Martin Kusche

Aufbau und Inbetriebnahme eines Zykliersystems für Batteriezellen mit ARM-Mikrocontroller

Motivation

Das Forschungsprojekt "Drahtlose Zellensensoren für Fahrzeugbatterien - BATSEN" an der HAW Hamburg untersucht verschiedene Batterietypen. Um das Batterieverhalten modellieren zu können, ist es zuvor notwendig, eine Reihe von Lade- und Entladevorgänge durchzuführen und dabei möglichst viele Messwerte, wie z.B. Zellspannung, Strom und Temperatur, aufzuzeichnen.

Diese zeitaufwändige Zyklierung wird durch ein spezielles Zykliersystem aus zwei Vorarbeiten automatisiert. Im praktischen Erprobungsbetrieb sind mehrere Probleme erkannt worden, zu nennen ist insbesondere ist die hohe Empfindlichkeit des verwendeten ADC gegen Überspannungen oder fehlerhaften Masseführungen.

Aufgabe

Herr Kusche erhält die Aufgabe, die o.g. Probleme systematisch zu untersuchen und Verbesserungsmöglichkeiten zu entwickeln. Die Lösungen - wie z.B. Schutzschaltungen - sollen in beiden Batteriezykliergeräte eingesetzt werden. Die praktische Nutzung ist umfassend zu erproben und mit neuen Teillösungen zu unterstützen (Einbindung in das HAW-Netz, Fernsteuerung, Kopplung mit Temperaturschrank). Im Zusammenhang mit der Erprobung sind Kurzanleitungen für wichtige Teilarbeiten mit dem Zykliersystem zu erstellen.
Der Schwerpunkt der Arbeit liegt darüber hinaus bei der Erprobung von großen Lithiumzellen unter verschiedenen Betriebsszenarien. Dabei sollen typische Fahrsituationen aus E-Fahrzeugen herangezogen werden, wie z.B. der Neue Europäische Fahrzyklus für Elektrofahrzeuge. Hierbei ist der Aspekt des Temperatureinflusses auf die Batterien zu betrachten.

Gliederung

Die Aufgabe der Bachelorarbeit gliedert sich wie folgt:

1) **Analyse der Rahmenbedingungen und Vorarbeiten**

   - Einarbeitung durch Recherche und Literatur
   - Einarbeitung in die Hard- und Software der Vorarbeiten
   - Recherche von Batterieuntersuchungen und -Testverfahren für das Anwendungsgebiet E-Mobilität
   - Analyse von Verbesserungs- und Erweiterungs-Möglichkeiten

2) **Entwicklung von Schutzschaltungen für die Hardware des Gerätes**

- Analyse bestehender Fehlermöglichkeiten und Empfindlichkeiten
- galvanische Trennung der Daten-Schnittstellen
- Korrektur der ADC-Schaltung, Entwurf und Untersuchung von Zusatzmodulen (Aufsetzplatine), Anpassung von Kalibrierungen an die neue ADC-Schaltung

3) **Kommunikationsschnittstellen - Inbetriebnahme und Verbesserung**

- Fernbedienung für das Zykliersystem im HAW-Netzwerk
- Kopplung von Kühlkammer und Zykliergerät prüfen und vorbereiten

4) **Ausführliche Dokumentation über die Benutzung**

- Anleitung zum Kalibrieren
- Anleitung zum Verbindungsaufbau
- Beschreibung von Default-Werten und der Konfigurationsdatei
- Beschreibung von Beispielen

5) **Planung und Durchführung von Batterieuntersuchungen**

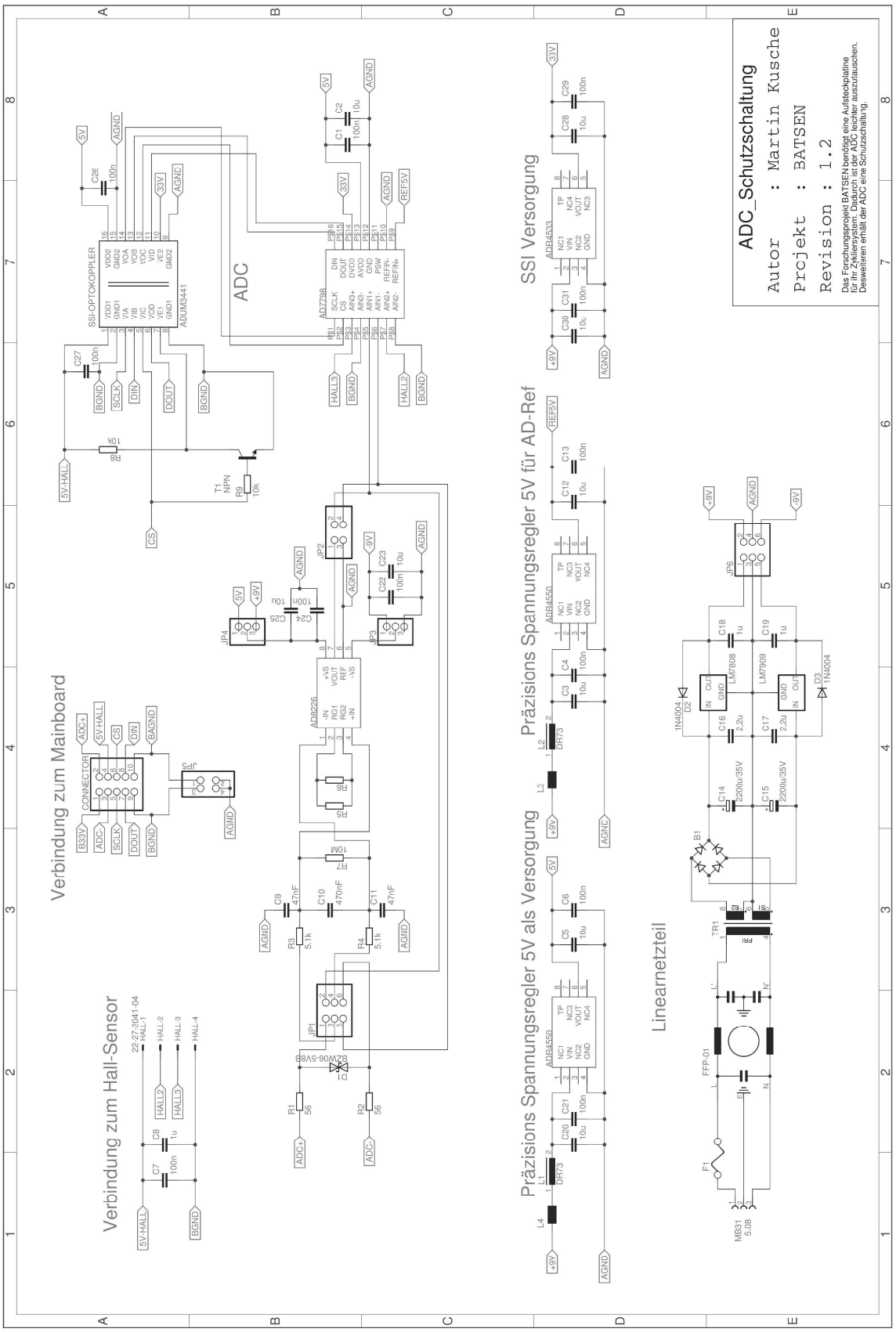
- Messkonzept nach dem Neuen Europäischen Fahrzyklus (NEFZ)
- Messkonzept zur Erstellung von Zellenprofilen (Zellencharakterisierung)
- Aufbau einer Messumgebung (u.a. eine Batteriebox im Metallkoffer)
- Aufstellung eines Messplans
- Zyklierung ausgewählter Zellen
- Auswertung der Messreihen und Aufbereitung

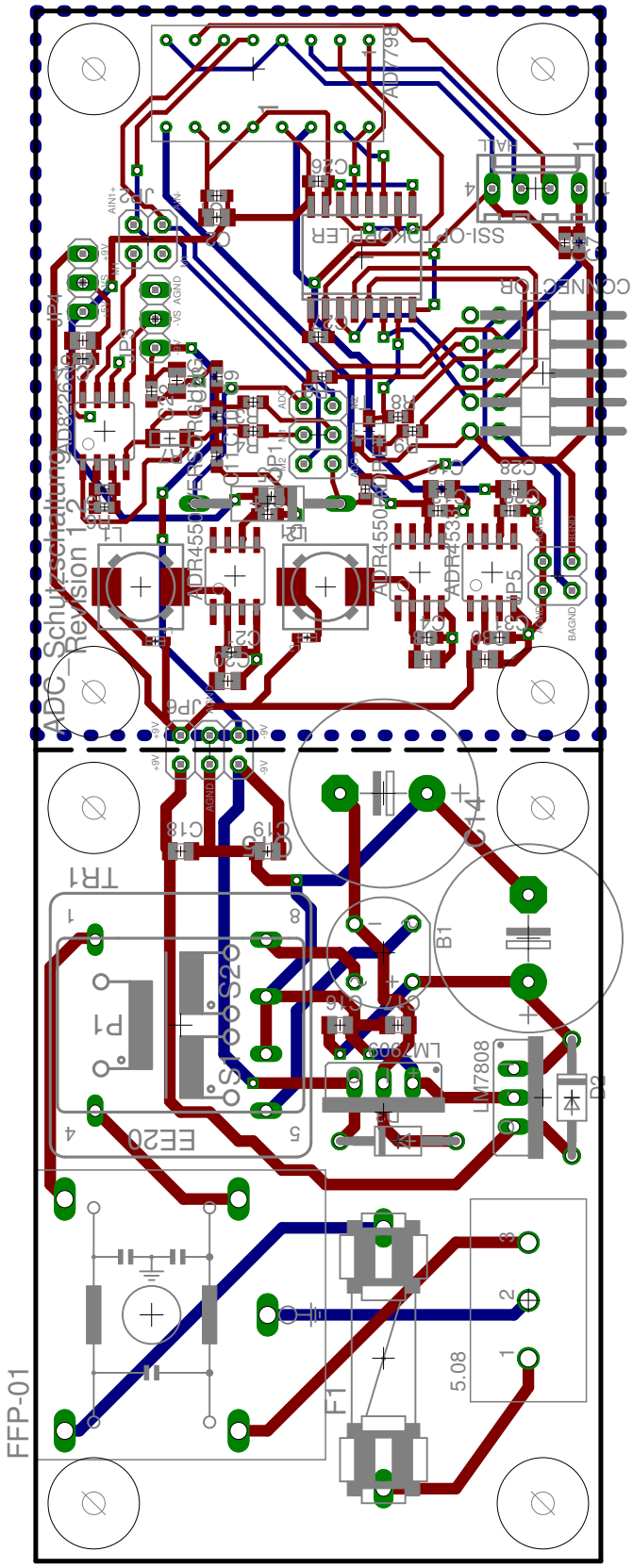
6) **Auswertung und Bewertung**

- Diskussion und Bewertung der Messreihen
- Bewertung erzielten Ergebnisse im Hinblick auf das Anwendungsgebiet der E-Mobilität
- Vorschläge zur Weiterentwicklung des Zykliersystems und der Messplanung

Dokumentation

Die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Die gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungen, die Grundkonzeption, auftretende Probleme und wesentliche Entwurfsentscheidungen sollen beschrieben werden. Die Ergebnisse sind systematisch zu erfassen und auszuwerten.


ADC_Schutzschaltung
Autor : Martin Kusche
Prcjekt : BATSEN
Revision : 1.2
Das Forschungsprojekt BATSEN benötigt eine Aufsteckplatine für ihr Zyklersystem. Dadurch ist der ADC leichter auszutauschen. Desweiteren enthält der ADC eine Schutzschaltung.
Verbindung zum Mainboard
Verbindung zum Hall-Sensor
ADC
SSI Versorgung
Präzisions Spannungsregler 5V für AD-Ref
Präzisions Spannungsregler 5V als Versorgung
Linearnetzteil
SSI-OPTOKOPPLER
ADUM3441
AD7798
ADR4533
ADR4550
AD8226
LM7808
LM7909
1N4004
1N4004
BZW06-5V8B
CONNECTOR
22-27-2041-04
MB31 5.08
FFP-01


ADC_Schutzschaltung
Revision 1.0
FFP-01
TR1
LM7808
ADR4550
ADR4533
ADR4550
SSI-CONTROLLER
CONNECTOR
AD7798
EE20
5.08

Teileliste der ADC Schutzschaltung 1.2

Netzteil

Bezeichnung	Händler	Artikelnummer	Anzahl	Einzelpreis	Gesamtpreis	Bemerkung
B1	Reichelt	B380C1500RUND	1	0,17 €	0,17 €	Rund-Gleichrichter
C14,C15	Reichelt	RAD FR 2.200/35	2	1,10 €	2,20 €	Elko radial, 35V, 2200µF
C16,C17	Reichelt	X7R-G0805 2,2/25	2	0,09 €	0,18 €	SMD-Vielschichtkondensator G0805 - 2,2µF 25V
C18,C19	Reichelt	X7R-G0805 1,0/25	2	0,05 €	0,10 €	SMD-Vielschichtkondensator G0805 - 1,0µF 25V
D2,D3	vom BATSEN Projekt		2	0 €	0,00 €	1N4004 - Freilaufdioden
JP6	vom BATSEN Projekt		1	0 €	0,00 €	2x3 Pinhead 2,54mm
LM 7809	vom BATSEN Projekt		1	0 €	0,00 €	9V Festspannungsregler pos.
LM 7909	vom BATSEN Projekt		1	0 €	0,00 €	9V Festspannungsregler neg.
MB31 5.08	vom BATSEN Projekt		1	0 €	0,00 €	230V Anschlussklemme 3polig
F1	Reichelt	PL 112100	1	0,21 €	0,21 €	Sicherungshalter
F1	Reichelt	K-FLINK 0,05A	1	0,59 €	0,59 €	Sicherung
FFP-01	Reichelt	FFP-01 55002000	1	3,85 €	3,85 €	AC Filter für PCB Montage, 0,6A
TR1	Reichelt	EE 20/10 209	1	3,25 €	3,25 €	Trafo 0,5VA, 2x 9V, 2x 28mA
Gesamtpreis					10,55 €	

ADC Schutzschaltung

Bezeichnung	Händler	Artikelnummer	Anzahl	Einzelpreis	Gesamtpreis	Bemerkung
AD7798	sample von Analog Devices		1	0 €	0,00 €	ADC
AD8226	sample von Analog Devices		1	0 €	0,00 €	Instrumentenverstärker
ADR4550	sample von Analog Devices		2	0 €	0,00 €	Präzisions Spannungsregler 5V
ADR4533	sample von Analog Devices		1	0 €	0,00 €	Präzisions Spannungsregler 3,3V
ADUM3441	sample von Analog Devices		1	0 €	0,00 €	Optokoppler
C9, C11	vom BATSEN Projekt		2	0 €	0,00 €	Kondensator Tiefpass 47nF
C10	vom BATSEN Projekt		1	0 €	0,00 €	Kondensator Tiefpass 470nF
C1, C4, C6, C7, C13, C21, C22, C24, C26, C27, C29, C31	vom BATSEN Projekt		12	0 €	0,00 €	Keramikondesatoren 100nF Blockkondensatoren
C2, C3, C5, C12, C20, C23, C25, C28, C30	vom BATSEN Projekt		9	0 €	0,00 €	Keramikondesatoren 10uF Blockkondensatoren
D1	vom BATSEN Projekt		1	0 €	0,00 €	Supressordiode BZW06-5V8B
JP1	vom BATSEN Projekt		1	0 €	0,00 €	2x3 Pinhead 2,54mm
JP2, JP5	vom BATSEN Projekt		2	0 €	0,00 €	2x2 Pinhead 2,54mm
JP3, JP4	vom BATSEN Projekt		2	0 €	0,00 €	1x3 Pinhead 2,54mm
HALL-Connector	vom BATSEN Projekt		1	0 €	0,00 €	4 Pin - Connector
L1, L2	vom BATSEN Projekt		2	0 €	0,00 €	Drosseln
L3, L4	vom BATSEN Projekt		2	0 €	0,00 €	Drosseln
R1, R2	vom BATSEN Projekt		2	0 €	0,00 €	Vorwiderstand ADC 56Ohm
R3, R4	vom BATSEN Projekt		2	0 €	0,00 €	Widerstand Tiefpass 5.1kOhm
R8, R9	vom BATSEN Projekt		2	0 €	0,00 €	Widerstand Inverter 10kOhm
T1	vom BATSEN Projekt		1	0 €	0,00 €	NPN-SMD Transistor Inverter
Jumper	vom BATSEN Projekt		11	0 €	0,00 €	zum Einstellen
Gesamtpreis					0,00 €	

```
function [NEFZ,time] = genNZ
% diese Funktion erstellt die den Neuen Europäischen Normzyklus (NEFZ)
% der NEFZ besteht aus 4 Grundstadtfahrzyklen und 1 Außerstädtischer Fahrzyklus
% nach Richtlinie 91/441/EG
  Zyklus1=genSFZ();
  Zyklus2=genSFZ();
  Zyklus3=genSFZ();
  Zyklus4=genSFZ();
  Zyklus5=genAFZ();
  NEFZ=[Zyklus1 Zyklus2 Zyklus3 Zyklus4 Zyklus5];
  time=0:0.001:(size(NEFZ,2)/1000)-0.001;

 % time=0:0.001:9-0.001;     % Zeit in ms Auflösung
 % NEFZ=zeros(size(time));
 % for i=3000 : 6000-0.001 ; NEFZ(i)=50; end

end

function velocity = genSFZ()
% diese Funktion erstellt den Grundstadtfahrzyklus
% nach Richtlinie 91/441/EG
  time=0:0.001:195-0.001;     % Zeit in ms Auflösung
  velocity=zeros(size(time)); % Geschwindigkeitsvektor anlegen
  % nun folgen die einzelnen Betriebszustände
  for i=11000 : 15000-0.001 ; velocity(i)=velocity(i-1)+15/4000; end
  for i=15000 : 23000-0.001 ; velocity(i)=velocity(i-1); end
  for i=23000 : 28000-0.001 ; velocity(i)=velocity(i-1)-15/5000; end
  for i=49000 : 61000-0.001 ; velocity(i)=velocity(i-1)+32/12000; end
  for i=61000 : 85000-0.001 ; velocity(i)=velocity(i-1); end
  for i=85000 : 96000-0.001 ; velocity(i)=velocity(i-1)-32/11000; end
  for i=117000:143000-0.001 ; velocity(i)=velocity(i-1)+50/26000; end
  for i=143000:155000-0.001 ; velocity(i)=velocity(i-1); end
  for i=155000:163000-0.001 ; velocity(i)=velocity(i-1)-15/8000; end
  for i=163000:176000-0.001 ; velocity(i)=velocity(i-1); end
  for i=176000:188000-0.001 ; velocity(i)=velocity(i-1)-35/12000; end
end

function velocity = genAFZ()
% diese Funktion erstellt den Außerstädtischen Fahrzyklus
% nach Richtlinie 91/441/EG
  time=0:0.001:400-0.001;           % Zeit in ms Auflösung
  velocity=zeros(size(time)); % Geschwindigkeitsvektor anlegen
  % nun folgen die einzelnen Betriebszustände
  for i=20000 : 61000-0.001 ; velocity(i)=velocity(i-1)+70/41000; end
  for i=61000 :111000-0.001 ; velocity(i)=velocity(i-1); end
  for i=111000:119000-0.001 ; velocity(i)=velocity(i-1)-20/8000; end
  for i=119000:188000-0.001 ; velocity(i)=velocity(i-1); end
  for i=188000:201000-0.001 ; velocity(i)=velocity(i-1)+20/13000; end
  for i=201000:251000-0.001 ; velocity(i)=velocity(i-1); end
  for i=251000:286000-0.001 ; velocity(i)=velocity(i-1)+30/35000; end
  for i=286000:316000-0.001 ; velocity(i)=velocity(i-1); end
  for i=316000:336000-0.001 ; velocity(i)=velocity(i-1)+20/20000; end
  for i=336000:346000-0.001 ; velocity(i)=velocity(i-1); end
  for i=346000:380000-0.001 ; velocity(i)=velocity(i-1)-120/34000; end
end
```

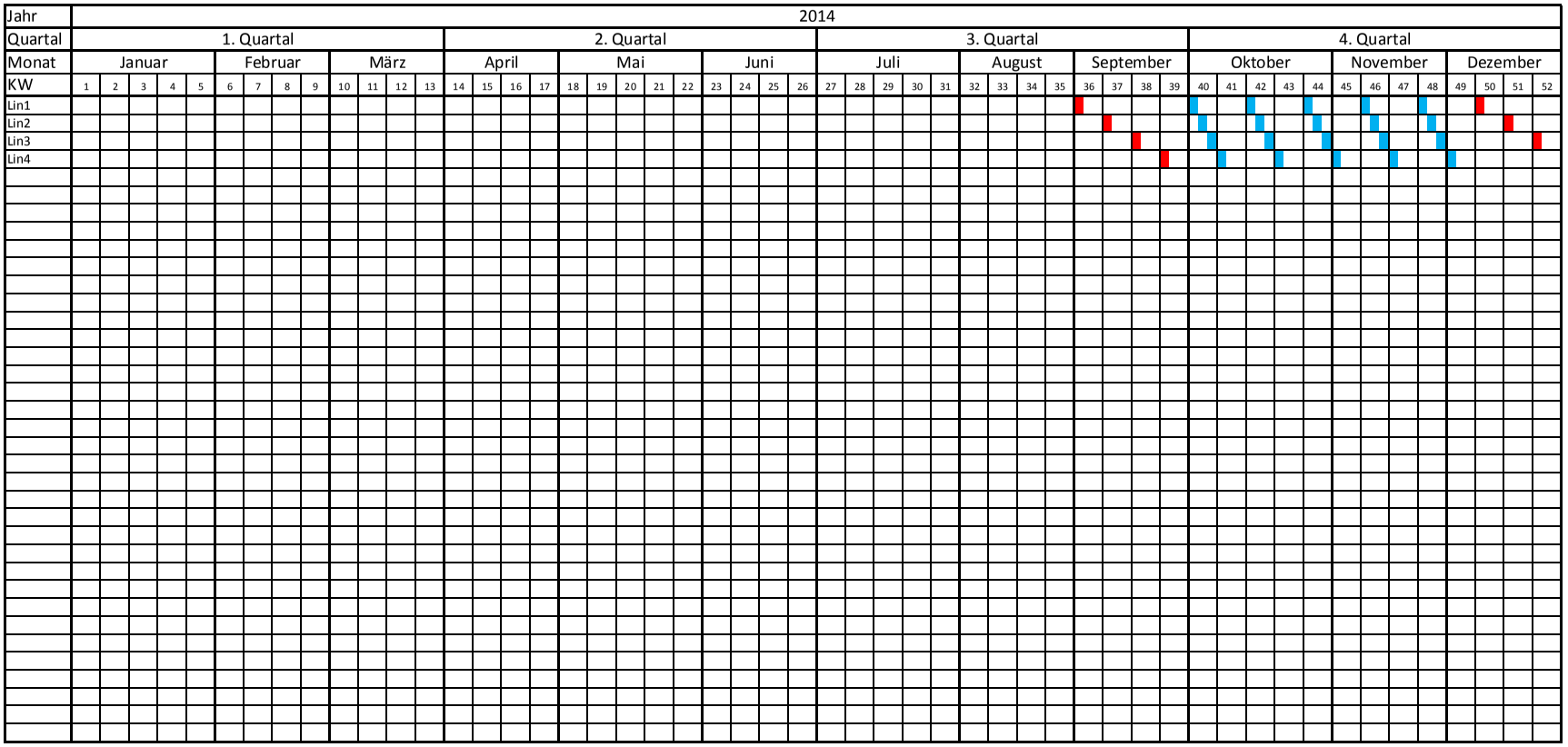
```
%% Automodelle
%-------------------------------------------------
%Twizy 45
Masse=548;    % Fahrzeuggewicht
Breite=1.228; % in m
Hoehe =1.451; % in m
CR=0.011;     % Rollwiderstandskoeffizient
CW=0.21;      % Luftwiderstandsbeiwert
WM=0.9;       % Wirkungsgrad Motor
WA=0.9;       % Wirkungsgrad Akku
%-------------------------------------------------
%VW e-up!
%204 Zellen in 17 Module (Batteriespannung 296V bis 418V)
Masse=1214;   % Fahrzeuggewicht
Breite=1.645; % in m
Hoehe =1.477; % in m
CR=0.011;     % Rollwiderstandskoeffizient
CW=0.308;     % Luftwiderstandsbeiwert (aus Datenblatt)
WM=0.9;       % Wirkungsgrad Motor
WA=0.9;       % Wirkungsgrad Akku
%-------------------------------------------------
%Citroen C-Zero
Masse=1140;   % Fahrzeuggewicht
Breite=1.475; % in m
Hoehe =1.608; % in m
CR=0.011;     % Rollwiderstandskoeffizient
CW=0.26;      % Luftwiderstandsbeiwert
WM=0.9;       % Wirkungsgrad Motor
WA=0.9;       % Wirkungsgrad Akku
%-------------------------------------------------

%% Verbrauchsberechnungen
%-------------------------------------------------
%VW e-up!
%204 Zellen in 17 Module (Batteriespannung 296V bis 418V)
Masse=1214;   % Fahrzeuggewicht
Breite=1.645; % in m
Hoehe =1.477; % in m
CR=0.011;     % Rollwiderstandskoeffizient
CW=0.308;     % Luftwiderstandsbeiwert (aus Datenblatt)
WM=0.9;       % Wirkungsgrad Motor
WA=0.9;       % Wirkungsgrad Akku
%-------------------------------------------------
%initialisierung
[NEFZ, time] = genNZ;             % NEFZ Verlauf
VektorSize   = size(NEFZ);        % Datensatzgrösse
Energie      = zeros(VektorSize); % Init Energie Datensatz
Leistung     = zeros(VektorSize); % Init Leistung Datensatz
RecuFlag     = zeros(VektorSize); % Init Rekuperationsphase Datensatz
StopFlag     = zeros(VektorSize); % Init Stopphase Datensatz

% Berechnung Fahrzeug spezifische Variablen
A=0.8*Breite*Hoehe; % Stirnfläche in m²
VF=1/(WM*WA);       % Verlust Faktor (Mehrverbrauchsfaktor)


%Leistungs Berechnung in kW
for i=1 : size(NEFZ,2)-1;           % Durchlauf des NEFZ Datensatz
  v1=NEFZ(i).*1000/3600;            % aktueller Punkt aus NEFZ Datensatz (in m/s)
```

```
60      v2=NEFZ(i+1).*1000/3600;            % nächster Punkt aus NEFZ Datensatz  (in m/s)
61      PRW=CR*Masse*9.81*v1*0.001*VF;      % Leistung durch den Rollwiderstand (in kW)
62      PLW=CW*A*1.29*0.5*v1^3*0.001*VF;    % Leistung durch den Luftwiderstand (in kW)
63      PKI=0.5*Masse*(v2^2 - v1^2)*VF;     % Leistung zur Beschleunigung (Kinetische
        Energie in kW)
64      % Energieberechnung
65      if v1 < v2                          % FALL1: Beschleunigung
66        Leistung(i)= PRW + PLW + PKI;
67      elseif v1 == v2 && v1 ~= 0          % FALL2: konstante Geschwindigkeit
68        Leistung(i)= PRW + PLW;
69      elseif v1 == v2 && v1 == 0          % FALL3: Stopp
70        StopFlag(i)=1;
71      elseif v1 > v2                      % Fall4: Bremsvorgang
72        Leistung(i) = PRW + PLW;
73        RecuFlag(i)=1;
74      end
75    end
76
77    % Berechnung der Strecke (Zeit in ms, Strecke in km)
78    WegVektor = cumtrapz(NEFZ)./3600000;
79    WegGesamt = WegVektor(end);
80
81    % aus Leistung zur Energie in kWh
82    Energie=cumtrapz(Leistung)./(1000.*3600);
83    EnergieGesamt=Energie(end);
84
85    % Ausgabe
86    fprintf('Verbrauch auf 100 Km = %.2f kW\n',EnergieGesamt*(100/WegGesamt));
87
88    % Plot
89    figure;
90    %hold on;
91    % Figure1
92    subplot(4,1,1);
93    plot(time,NEFZ);
94    title('Neuer Europäischer Fahrzyklus');
95    xlabel('time [s]');
96    ylabel('Geschwindigkeit [km/h]');
97    grid on;
98    % Figure2
99    subplot(4,1,2);
100
101   plot(time,Leistung);
102   title('benötigte Leistung');
103   xlabel('time [s]');
104   ylabel('Leistung [kW]');
105   grid on;
106
107   % Figure3
108   subplot(4,1,3);
109   plot(time,Energie);
110   title('Verbrauchte Energie');
111   xlabel('time [s]');
112   ylabel('Energie [kWh]');
113   grid on;
114
115   % Figure4
116   subplot(4,1,4);
117   plot(time,WegVektor);
```



Legende:

blau - NEFZ Messung

rot - Zellprofilmessung Messung

Identität	Hersteller	Bezeichnung	Typ	Kapazität	Nominale Spannung	Ladeschluss Spannung	Dauer Entladestrom	max. Ladestrom
Lin1	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin2	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin3	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C
Lin4	Headway	40152SE	LiFePO4	15Ah	3,2V	3,65V	5C	4C

```
%% lineare Kalibbrierung

Fluke_LOW  = 1.20115; % in V
Fluke_HIGH = 3.2010; % in V
ADC_VRef   = 5;
ADC_Bit    = 16;
ADC_GAIN   = 1;

%Berechnung aus Datenblatt des ADCs entnommen (Seite 23)
Fluke_LOW_Binaer  = (2^ADC_Bit * Fluke_LOW * ADC_GAIN)/ADC_VRef;
Fluke_HIGH_Binaer = (2^ADC_Bit * Fluke_HIGH * ADC_GAIN)/ADC_VRef;

%Gemessene Werte mit dem ADC in Binärdaten
CELL1_LOW = [9046 9041 9041 9043 9043 9041 9040 9044 9046 9045 9048 9048 9045 9045
9047 9046 9045 9046 9049 9049 9045];
CELL1_HIGH =[38824 38823 38823 38825 38825 38825 38826 38827 38826 38827 38827 38830
38831 38822 38824 38823 38820 38826 38829 38829 38828];
CELL_L_MEAN(1) = mean(CELL1_LOW);
CELL_H_MEAN(1) = mean(CELL1_HIGH);
LOW_Falure(1)  = max(CELL1_LOW) - min(CELL1_LOW);
High_Falure(1) = max(CELL1_HIGH) - min(CELL1_HIGH);

CELL2_LOW = [3611 3611 3612 3611 3612 3612 3611 3611 3611 3611 3611 3611 3611 3611
3611 3611 3610 3611 3611 3611 3611];
CELL2_HIGH =[40376 40376 40373 40371 40371 40372 40373 40375 40376 40375 40373 40371
40372 40373 40375 40376 40375 40373 40372 40372 40372];
CELL_L_MEAN(2) = mean(CELL2_LOW);
CELL_H_MEAN(2) = mean(CELL2_HIGH);
LOW_Falure(2)  = max(CELL2_LOW) - min(CELL2_LOW);
High_Falure(2) = max(CELL2_HIGH) - min(CELL2_HIGH);

CELL3_LOW = [3611 3611 3610 3611 3611 3611 3611 3612 3611 3611 3610 3611 3611 3611
3611 3611 3610 3611 3610 3611 3611];
CELL3_HIGH =[40376 40374 40372 40371 40374 40375 40376 40377 40376 40374 40373 40373
40372 40376 40376 40376 40374 40373 40373 40372 40372];
CELL_L_MEAN(3) = mean(CELL3_LOW);
CELL_H_MEAN(3) = mean(CELL3_HIGH);
LOW_Falure(3)  = max(CELL3_LOW) - min(CELL3_LOW);
High_Falure(3) = max(CELL3_HIGH) - min(CELL3_HIGH);

CELL4_LOW =[3609 3609 3609 3609 3609 3608 3609 3609 3609 3609 3609 3609 3609 3609
3609 3609 3609 3609 3609 3609 3609];
CELL4_HIGH =[40373 40371 40371 40369 40369 40369 40372 40372 40372 40372 40370 40370
40370 40372 40373 40374 40372 40369 40369 40369 40369];
CELL_L_MEAN(4) = mean(CELL4_LOW);
CELL_H_MEAN(4) = mean(CELL4_HIGH);
LOW_Falure(4)  = max(CELL4_LOW) - min(CELL4_LOW);
High_Falure(4) = max(CELL4_HIGH) - min(CELL4_HIGH);

CELL5_LOW =[3609 3609 3609 3609 3609 3609 3609 3609 3609 3609 3609 3609 3609 3608
3609 3609 3609 3609 3609 3609 3609];
CELL5_HIGH =[40375 40375 40374 40372 40371 40371 40374 40375 40374 40374 40373 40370
40372 40372 40373 40374 40374 40372 40371 40371];
CELL_L_MEAN(5) = mean(CELL5_LOW);
CELL_H_MEAN(5) = mean(CELL5_HIGH);
LOW_Falure(5)  = max(CELL5_LOW) - min(CELL5_LOW);
High_Falure(5) = max(CELL5_HIGH) - min(CELL5_HIGH);

CELL6_LOW =[3609 3609 3610 3610 3609 3610 3610 3610 3609 3610 3610 3610 3609 3610
```

```
    3610 3609 3609 3609 3609 3610 3610];
50  CELL6_HIGH =[40370 40372 40373 40373 40373 40372 40371 40371 40372 40374 40373 40375
    40372 40370 40369 40370 40373 40374 40374 40374];
51  CELL_L_MEAN(6) = mean(CELL6_LOW);
52  CELL_H_MEAN(6) = mean(CELL6_HIGH);
53  LOW_Falure(6)  = max(CELL6_LOW) - min(CELL6_LOW);
54  High_Falure(6) = max(CELL6_HIGH) - min(CELL6_HIGH);
55
56  CELL7_LOW =[3610 3610 3609 3609 3609 3609 3609 3609 3609 3610 3610 3609 3609 3610
    3609 3609 3609 3609 3609 3610 3610];
57  CELL7_HIGH =[40371 40372 40373 40375 40376 40376 40375 40372 40373 40372 40374 40374
    40376 40374 40375 40373 40371 40372 40373 40376];
58  CELL_L_MEAN(7) = mean(CELL7_LOW);
59  CELL_H_MEAN(7) = mean(CELL7_HIGH);
60  LOW_Falure(7)  = max(CELL7_LOW) - min(CELL7_LOW);
61  High_Falure(7) = max(CELL7_HIGH) - min(CELL7_HIGH);
62
63  CELL8_LOW =[3611 3611 3610 3610 3610 3610 3610 3611 3610 3610 3611 3610 3609 3609
    3609 3610 3610 3610 3610 3609];
64  CELL8_HIGH =[40375 40374 40373 40371 40372 40373 40376 40375 40374 40373 40370 40372
    40371 40373 40374 40375 40374 40372 40371 40371];
65  CELL_L_MEAN(8) = mean(CELL8_LOW);
66  CELL_H_MEAN(8) = mean(CELL8_HIGH);
67  LOW_Falure(8)  = max(CELL8_LOW) - min(CELL8_LOW);
68  High_Falure(8) = max(CELL8_HIGH) - min(CELL8_HIGH);
69
70  CELL9_LOW =[3610 3610 3610 3610 3610 3610 3609 3610 3610 3611 3610 3610 3610 3611
    3610 3611 3610 3610 3610 3611];
71  CELL9_HIGH =[40375 40376 40377 40375 40373 40372 40370 40374 40375 40375 40375 40373
    40372 40371 40372 40374 40375 40375 40375 40374 40374];
72  CELL_L_MEAN(9) = mean(CELL9_LOW);
73  CELL_H_MEAN(9) = mean(CELL9_HIGH);
74  LOW_Falure(9)  = max(CELL9_LOW) - min(CELL9_LOW);
75  High_Falure(9) = max(CELL9_HIGH) - min(CELL9_HIGH);
76
77  CELL10_LOW =[3610 3611 3610 3610 3611 3610 3610 3610 3610 3610 3610 3610 3610 3611
    3611 3610 3609 3610 3610 3610 3610];
78  CELL10_HIGH =[40371 40372 40373 40374 40375 40374 40373 40372 40370 40371 40374 40374
     40375 40373 40373 40371 40371 40373 40373 40373 40373];
79  CELL_L_MEAN(10) = mean(CELL10_LOW);
80  CELL_H_MEAN(10) = mean(CELL10_HIGH);
81  LOW_Falure(10)  = max(CELL10_LOW) - min(CELL10_LOW);
82  High_Falure(10) = max(CELL10_HIGH) - min(CELL10_HIGH);
83
84  CELL11_LOW =[3609 3610 3610 3609 3610 3610 3610 3609 3609 3610 3610 3609 3608 3609
    3609 3609 3609 3609 3609 3609 3609];
85  CELL11_HIGH =[40374 40372 40370 40369 40370 40372 40374 40374 40374 40370 40371 40371
     40372 40374 40376 40374 40372 40372 40369 40371 40371];
86  CELL_L_MEAN(11) = mean(CELL11_LOW);
87  CELL_H_MEAN(11) = mean(CELL11_HIGH);
88  LOW_Falure(11)  = max(CELL11_LOW) - min(CELL11_LOW);
89  High_Falure(11) = max(CELL11_HIGH) - min(CELL11_HIGH);
90
91  CELL12_LOW =[3608 3610 3608 3609 3608 3608 3609 3609 3608 3609 3609 3609 3609 3609
    3608 3609 3609 3609 3609 3608 3608];
92  CELL12_HIGH =[40374 40374 40373 40370 40370 40371 40373 40373 40376 40374 40373 40371
     40372 40370 40373 40373 40375 40373 40371 40370 40370];
93  CELL_L_MEAN(12) = mean(CELL12_LOW);
94  CELL_H_MEAN(12) = mean(CELL12_HIGH);
```

```
95  LOW_Falure(12)  = max(CELL12_LOW) - min(CELL12_LOW);
96  High_Falure(12) = max(CELL12_HIGH) - min(CELL12_HIGH);
97
98  %%Calculation of gain and offset
99  Diff = Fluke_HIGH_Binaer - Fluke_LOW_Binaer;
100 Multiplikator = 5/2^ADC_Bit * 1000000;
101
102 for i= 1:12
103   Gain(i)=Diff/(CELL_H_MEAN(i)-CELL_L_MEAN(i));
104   Offset(i)=Fluke_LOW_Binaer-CELL_L_MEAN(i)*Gain(i);
105   fprintf('Gain%d:%d     Offset%d:%d     LowAbweichung:%d     HighAbweichung:%d\n',i,
      round(Gain(i)*1000*Multiplikator),i,round(Offset(i)*Multiplikator),LOW_Falure(i),
      High_Falure(i));
106 end
107
108 %% Überprüfung
109 % Übernahme der Ausgegebenen Daten
110 % alle Werte werden positiv angegeben
111 Gain        = 80589;
112 Offset      = 471845;
113
114 Spannung  = [0.25039 0.50027 0.74883 1.0025 1.2507 1.5006 1.7522 2.0028 2.2513 2.5019
     2.75 3.0026 3.251 3.5 3.75 4]; % in V
115 ADC_CODE  = [1728 3490 5202 7058 9620 12408 15366 18400 21557 24800 27941 31196 34375
     37548 40739 43917]; % zur Spannung gehörender ADC Code
116 Input     = [0.25 0.5 0.75 1 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.255 3.5 3.75 4];  %
    ideale Kennlinie in V
117
118 % implementierte Kalibriergleichung
119 AusgabeKorekt = ADC_CODE .* Gain ./ 1000 + Offset;
120
121 subplot(2,1,1); % Kalibrierte Übertragunsfunktion
122 plot(Input,Input,'*',Spannung,AusgabeKorekt./1000000,'r');
123 title('Ergebniss Kalibrierung');
124 legend('ideale Übertragungskennlinie','kalibrierte Kennlinie');
125 xlabel('ADC Eingang in V');
126 ylabel('ADC Ausgabe Codiert');
127 grid on;
128 subplot(2,1,2); % Fehler nach Kalibrierung
129 plot(Spannung,(Spannung-(AusgabeKorekt./1000000))*1000,'g');
130 title('Fehler nach Kalibrierung');
131 legend('quadratischer Korrekturfehler');
132 xlabel('ADC Eingang in V');
133 ylabel('Fehler in mV');
134 grid on;
135
```

```
%% quadratische Kalibrierung

Fluke_LOW    = 1.2002;  % in V
FLUKE_MIDDLE = 2.0005;  % in V
Fluke_HIGH   = 3.2019;  % in V
ADC_VRef     = 5;
ADC_Bit      = 16;
ADC_GAIN     = 1;

%Berechnung aus Datenblatt des ADCs entnommen (Seite 23)
Fluke_LOW_Binaer  = (2^ADC_Bit * Fluke_LOW * ADC_GAIN)/ADC_VRef;
Fluke_MID_Binaer  = (2^ADC_Bit * FLUKE_MIDDLE * ADC_GAIN)/ADC_VRef;
Fluke_HIGH_Binaer = (2^ADC_Bit * Fluke_HIGH * ADC_GAIN)/ADC_VRef;

%Gemessene Werte mit dem ADC in Binärdaten
CELL1_LOW    = [9051 9059 9052 9056 9056 9060 9064 9059 9060 9064 9067 9060 9061 9062
 9060 9065 9054 9061 9058 9059 9056 9053 9053 9050 9050 9052 9057 9056 9052 9053 9058
 9058 9055 9058 9055 9054 9056 9053 9051];
CELL1_MIDDLE = [23497 23501 23504 23502 23502 23504 23500 23505 23502 23504 23507
23505 23508 23508 23508 23505 23512 23514 23508 23509 23510 23505 23501 23499 23501
23503 23502 23496 23497 23500 23496 23500 23501 23500 23504 23503 23499 23501 23495
23495];
CELL1_HIGH   = [38826 38833 38833 38827 38837 38838 38841 38835 38841 38845 38841
38841 38846 38847 38846 38838 38845 38847 38844 38846 38847 38847 38847 38847 38849
38847 38848 38850 38849 38856 38859 38855 38851 38851 38853 38857 38856 38862 38855
38855];
CELL_L_MEAN(1) = mean(CELL1_LOW);
CELL_M_MEAN(1) = mean(CELL1_MIDDLE);
CELL_H_MEAN(1) = mean(CELL1_HIGH);
LOW_Falure(1)  = max(CELL1_LOW) - min(CELL1_LOW);
MID_Falure(1)  = max(CELL1_MIDDLE) - min(CELL1_MIDDLE);
High_Falure(1) = max(CELL1_HIGH) - min(CELL1_HIGH);

CELL2_LOW    = [9048 9045 9043 9044 9050 9043 9045 9046 9003 9050 9050 9046 9046 9048
 9049 9049 9050 9052 9057 9050 9056 9051 9052 9056 9050 9055 9055 9055 9054 9055 9057
 9052 9055 9057 9061 9059 9061 9056 9056];
CELL2_MIDDLE = [23510 23512 23510 23517 23512 23512 23516 23517 23514 23515 23514
23515 23521 23519 23519 23520 23523 23519 23523 23527 23526 23522 23517 23527 23522
23519 23519 23525 23521 23520 23523 23525 23527 23525 23524 23522 23524 23521 23523
23523];
CELL2_HIGH   = [38856 38853 38850 38851 38856 38854 38850 38854 38852 38854 38852
38853 38853 38851 38855 38851 38849 38852 38849 38851 38850 38846 38849 38849 38849
38858 38855 38849 38856 38856 38854 38852 38854 38851 38852 38855 38849 38851 38854
38854];
CELL_L_MEAN(2) = mean(CELL2_LOW);
CELL_M_MEAN(2) = mean(CELL2_MIDDLE);
CELL_H_MEAN(2) = mean(CELL2_HIGH);
LOW_Falure(2)  = max(CELL2_LOW) - min(CELL2_LOW);
MID_Falure(2)  = max(CELL2_MIDDLE) - min(CELL2_MIDDLE);
High_Falure(2) = max(CELL2_HIGH) - min(CELL2_HIGH);

CELL3_LOW    = [9067 9063 9059 9058 9061 9063 9062 9063 9062 9058 9064 9065 9067 9068
 9065 9070 9067 9069 9072 9066 9068 9073 9072 9071 9055 9050 9050 9051 9047 9043 9045
 9050 9040 9040 9043 9046 9046 9044 9037];
CELL3_MIDDLE = [23509 23511 23511 23509 23510 23514 23513 23513 23515 23513 23513
23504 23498 23498 23507 23504 23508 23498 23505 23505 23506 23503 23505 23506 23504
23504 23509 23507 23505 23504 23509 23507 23513 23509 23510 23510 23509 23507 23513
23513];
CELL3_HIGH   = [38853 38853 38858 38857 38856 38861 38856 38858 38858 38859 38859
```

```
    38858 38862 38862 38856 38860 38857 38860 38858 38861 38862 38863 38860 38862 38863
    38863 38871 38869 38867 38867 38862 38869 38866 38862 38863 38865 38868 38866 38866
    38866];
39  CELL_L_MEAN(3) = mean(CELL3_LOW);
40  CELL_M_MEAN(3) = mean(CELL3_MIDDLE);
41  CELL_H_MEAN(3) = mean(CELL3_HIGH);
42  LOW_Falure(3)  = max(CELL3_LOW) - min(CELL3_LOW);
43  MID_Falure(3)  = max(CELL3_MIDDLE) - min(CELL3_MIDDLE);
44  High_Falure(3) = max(CELL3_HIGH) - min(CELL3_HIGH);
45
46  CELL4_LOW    = [9111 9106 9109 9112 9111 9105 9114 9110 9111 9109 9107 9111 9107 9111
     9106 9103 9105 9106 9105 9107 9107 9100 9104 9102 9110 9100 9097 9095 9092 9096 9092
     9093 9094 9094 9089 9086 9089 9094 9093 9093];
47  CELL4_MIDDLE = [23503 23501 23500 23502 23502 23502 23505 23499 23502 23505 23515
    23515 23522 23517 23515 23510 23513 23512 23512 23511 23509 23512 23508 23509 23512
    23513 23516 23519 23515 23514 23513 23513 23510 23514 23516 23513 23514 23516 23515
    23515];
48  CELL4_HIGH   = [38848 38852 38858 38859 38855 38858 38858 38858 38861 38860 38859
    38860 38860 38859 38862 38861 38869 38868 38864 38869 38872 38874 38868 38870 38870
    38868 38867 38869 38874 38874 38874 38873 38868 38865 38866 38867 38863 38864 38866
    38866];
49  CELL_L_MEAN(4) = mean(CELL4_LOW);
50  CELL_M_MEAN(4) = mean(CELL4_MIDDLE);
51  CELL_H_MEAN(4) = mean(CELL4_HIGH);
52  LOW_Falure(4)  = max(CELL4_LOW) - min(CELL4_LOW);
53  MID_Falure(4)  = max(CELL4_MIDDLE) - min(CELL4_MIDDLE);
54  High_Falure(4) = max(CELL4_HIGH) - min(CELL4_HIGH);
55
56  CELL5_LOW    = [9071 9070 9072 9070 9065 9066 9068 9070 9067 9067 9067 9066 9062 9063
     9064 9067 9063 9059 9056 9060 9061 9061 9064 9065 9062 9063 9070 9078 9087 9085 9086
     9089 9090 9100 9099 9099 9103 9101 9098 9098];
57  CELL5_MIDDLE = [23492 23492 23490 23495 23492 23491 23492 23490 23485 23487 23491
    23487 23486 23490 23490 23489 23486 23486 23488 23489 23486 23487 23494 23485 23487
    23490 23489 23484 23487 23490 23488 23484 23486 23488 23488 23492 23491 23491 23491
    23491];
58  CELL5_HIGH   = [38835 38839 38840 38846 38842 38842 38844 38849 38849 38842 38844
    38847 38847 38846 38844 38845 38850 38854 38855 38853 38855 38847 38850 38850 38852
    38852 38848 38849 38845 38848 38848 38849 38846 38847 38852 38846 38850 38848 38855
    38855];
59  CELL_L_MEAN(5) = mean(CELL5_LOW);
60  CELL_M_MEAN(5) = mean(CELL5_MIDDLE);
61  CELL_H_MEAN(5) = mean(CELL5_HIGH);
62  LOW_Falure(5)  = max(CELL5_LOW) - min(CELL5_LOW);
63  MID_Falure(5)  = max(CELL5_MIDDLE) - min(CELL5_MIDDLE);
64  High_Falure(5) = max(CELL5_HIGH) - min(CELL5_HIGH);
65
66  CELL6_LOW    = [9066 9064 9065 9070 9066 9068 9062 9061 9063 9066 9064 9063 9061 9055
     9057 9058 9058 9060 9059 9060 9058 9056 9061 9070 9070 9061 9064 9066 9065 9063 9064
     9065 9061 9067 9069 9069 9069 9069 9065 9065];
67  CELL6_MIDDLE = [23515 23518 23519 23521 23520 23519 23516 23515 23515 23514 23515
    23502 23506 23503 23501 23499 23499 23499 23499 23501 23504 23503 23499 23497 23500
    23500 23504 23505 23504 23503 23509 23506 23509 23514 23513 23515 23513 23516 23513
    23513];
68  CELL6_HIGH   = [38844 38841 38841 38840 38840 38842 38842 38837 38842 38841 38835
    38839 38838 38844 38843 38845 38841 38843 38842 38842 38841 38846 38849 38852 38852
    38853 38848 38852 38851 38853 38855 38855 38858 38851 38855 38855 38853 38856 38854
    38854];
69  CELL_L_MEAN(6) = mean(CELL6_LOW);
70  CELL_M_MEAN(6) = mean(CELL6_MIDDLE);
```

```
71  CELL_H_MEAN(6) = mean(CELL6_HIGH);
72  LOW_Falure(6)  = max(CELL6_LOW) - min(CELL6_LOW);
73  MID_Falure(6)  = max(CELL6_MIDDLE) - min(CELL6_MIDDLE);
74  High_Falure(6) = max(CELL6_HIGH) - min(CELL6_HIGH);
75
76  CELL7_LOW    = [9059 9058 9058 9059 9061 9059 9061 9061 9063 9057 9059 9058 9057 9059
     9061 9063 9059 9060 9060 9065 9066 9065 9060 9062 9064 9060 9059 9065 9062 9061 9066
     9068 9064 9063 9060 9062 9068 9069 9067 9067];
77  CELL7_MIDDLE = [23517 23515 23520 23522 23524 23515 23519 23520 23522 23519 23521
    23522 23524 23528 23527 23528 23528 23523 23519 23518 23515 23511 23511 23506 23507
    23512 23508 23512 23513 23514 23520 23516 23517 23516 23517 23521 23522 23523 23525
    23525];
78  CELL7_HIGH   = [38869 38870 38865 38866 38867 38865 38860 38860 38860 38861 38859
    38853 38854 38854 38852 38855 38854 38853 38854 38856 38851 38852 38851 38853 38854
    38847 38852 38854 38856 38856 38850 38852 38850 38851 38852 38851 38850 38849 38851
    38851];
79  CELL_L_MEAN(7) = mean(CELL7_LOW);
80  CELL_M_MEAN(7) = mean(CELL7_MIDDLE);
81  CELL_H_MEAN(7) = mean(CELL7_HIGH);
82  LOW_Falure(7)  = max(CELL7_LOW) - min(CELL7_LOW);
83  MID_Falure(7)  = max(CELL7_MIDDLE) - min(CELL7_MIDDLE);
84  High_Falure(7) = max(CELL7_HIGH) - min(CELL7_HIGH);
85
86  CELL8_LOW    = [9051 9061 9057 9057 9055 9053 9055 9058 9062 9057 9063 9064 9064 9063
     9066 9066 9065 9067 9064 9061 9065 9056 9062 9062 9062 9057 9061 9059 9065 9058 9058
     9054 9055 9054 9052 9055 9055 9058 9059 9059];
87  CELL8_MIDDLE = [23525 23522 23524 23525 23522 23524 23523 23520 23531 23530 23529
    23527 23529 23526 23521 23512 23508 23509 23503 23497 23500 23503 23499 23500 23502
    23505 23504 23502 23505 23505 23499 23502 23506 23507 23507 23506 23507 23506 23505
    23505];
88  CELL8_HIGH   = [38881 38880 38876 38873 38875 38874 38867 38866 38867 38872 38871
    38866 38864 38864 38861 38864 38860 38863 38864 38863 38863 38860 38863 38859 38862
    38861 38859 38860 38863 38860 38861 38864 38867 38867 38864 38868 38865 38864 38861
    38861];
89  CELL_L_MEAN(8) = mean(CELL8_LOW);
90  CELL_M_MEAN(8) = mean(CELL8_MIDDLE);
91  CELL_H_MEAN(8) = mean(CELL8_HIGH);
92  LOW_Falure(8)  = max(CELL8_LOW) - min(CELL8_LOW);
93  MID_Falure(8)  = max(CELL8_MIDDLE) - min(CELL8_MIDDLE);
94  High_Falure(8) = max(CELL8_HIGH) - min(CELL8_HIGH);
95
96  CELL9_LOW    = [9035 9040 9039 9036 9036 9037 9039 9039 9040 9045 9040 9041 9043 9037
     9041 9040 9040 9040 9039 9041 9044 9040 9040 9038 9040 9039 9043 9043 9047 9042 9043
     9046 9050 9050 9054 9047 9052 9050 9055 9055];
97  CELL9_MIDDLE = [23532 23532 23532 23532 23533 23529 23530 23529 23529 23529 23530
    23531 23533 23530 23529 23533 23530 23529 23529 23533 23532 23525 23526 23529 23528
    23528 23525 23519 23522 23523 23521 23522 23519 23522 23522 23517 23519 23523 23522
    23522];
98  CELL9_HIGH   = [38869 38864 38861 38861 38863 38863 38860 38861 38857 38862 38861
    38861 38861 38863 38862 38858 38855 38857 38857 38858 38856 38852 38852 38854 38855
    38854 38853 38854 38857 38855 38858 38856 38857 38853 38854 38859 38864 38862 38859
    38859];
99  CELL_L_MEAN(9) = mean(CELL9_LOW);
100 CELL_M_MEAN(9) = mean(CELL9_MIDDLE);
101 CELL_H_MEAN(9) = mean(CELL9_HIGH);
102 LOW_Falure(9)  = max(CELL9_LOW) - min(CELL9_LOW);
103 MID_Falure(9)  = max(CELL9_MIDDLE) - min(CELL9_MIDDLE);
104 High_Falure(9) = max(CELL9_HIGH) - min(CELL9_HIGH);
105
```

```
106  CELL10_LOW    = [9082 9084 9082 9083 9086 9087 9088 9085 9086 9086 9084 9087 9088
     9090 9089 9090 9089 9088 9087 9087 9090 9092 9090 9091 9091 9091 9092 9090 9090 9092
     9087 9091 9090 9092 9086 9084 9086 9083 9085 9085];
107  CELL10_MIDDLE = [23514 23521 23516 23514 23516 23519 23525 23517 23518 23519 23526
     23523 23521 23525 23528 23527 23523 23523 23528 23536 23533 23530 23532 23535 23535
     23531 23526 23531 23532 23529 23529 23530 23530 23527 23531 23528 23531 23526 23532
     23532];
108  CELL10_HIGH   = [38849 38855 38859 38857 38860 38857 38853 38856 38854 38856 38856
     38853 38851 38853 38850 38852 38854 38854 38850 38849 38853 38852 38852 38851 38854
     38855 38857 38857 38852 38854 38853 38854 38858 38855 38854 38854 38854 38854 38860
     38860];
109  CELL_L_MEAN(10) = mean(CELL1_LOW);
110  CELL_M_MEAN(10) = mean(CELL1_MIDDLE);
111  CELL_H_MEAN(10) = mean(CELL1_HIGH);
112  LOW_Falure(10)  = max(CELL10_LOW) - min(CELL10_LOW);
113  MID_Falure(10)  = max(CELL10_MIDDLE) - min(CELL10_MIDDLE);
114  High_Falure(10) = max(CELL10_HIGH) - min(CELL10_HIGH);
115
116  CELL11_LOW    = [9083 9084 9085 9088 9085 9083 9075 9075 9073 9076 9074 9073 9078
     9077 9081 9083 9082 9088 9087 9090 9088 9090 9083 9079 9077 9078 9082 9083 9085 9087
     9085 9082 9085 9085 9085 9083 9088 9088 9089 9089];
117  CELL11_MIDDLE = [23499 23501 23505 23509 23510 23508 23512 23511 23508 23514 23511
     23509 23513 23515 23510 23509 23508 23511 23510 23509 23512 23506 23508 23515 23510
     23507 23516 23516 23517 23520 23524 23523 23522 23518 23523 23517 23519 23520 23519
     23519];
118  CELL11_HIGH   = [38839 38840 38839 38841 38843 38846 38849 38847 38846 38842 38840
     38839 38843 38839 38837 38834 38839 38840 38843 38842 38844 38838 38840 38838 38845
     38843 38843 38840 38839 38840 38838 38843 38844 38847 38846 38843 38837 38840 38840
     38840];
119  CELL_L_MEAN(11) = mean(CELL11_LOW);
120  CELL_M_MEAN(11) = mean(CELL11_MIDDLE);
121  CELL_H_MEAN(11) = mean(CELL11_HIGH);
122  LOW_Falure(11)  = max(CELL11_LOW) - min(CELL11_LOW);
123  MID_Falure(11)  = max(CELL11_MIDDLE) - min(CELL11_MIDDLE);
124  High_Falure(11) = max(CELL11_HIGH) - min(CELL11_HIGH);
125
126  CELL12_LOW    = [9067 9065 9065 9065 9060 9059 9059 9061 9066 9061 9059 9060 9062
     9057 9064 9062 9065 9060 9064 9065 9064 9062 9062 9062 9062 9060 9061 9061 9057 9058
     9057 9060 9057 9062 9056 9060 9055 9052 9054 9054];
127  CELL12_MIDDLE = [23522 23518 23516 23511 23515 23513 23508 23508 23508 23513 23512
     23512 23514 23514 23512 23514 23511 23510 23501 23493 23496 23496 23497 23496 23498
     23493 23497 23496 23495 23495 23491 23494 23490 23494 23492 23489 23493 23488 23490];
128  CELL12_HIGH   = [38849 38847 38848 38848 38854 38851 38850 38849 38855 38856 38854
     38854 38854 38855 38853 38847 38849 38849 38849 38847 38846 38843 38845 38847 38848
     38847 38846 38843 38848 38847 38849 38844 38844 38847 38845 38843 38846 38844 38842];
129  CELL_L_MEAN(12) = mean(CELL12_LOW);
130  CELL_M_MEAN(12) = mean(CELL12_MIDDLE);
131  CELL_H_MEAN(12) = mean(CELL12_HIGH);
132  LOW_Falure(12)  = max(CELL12_LOW) - min(CELL12_LOW);
133  MID_Falure(12)  = max(CELL12_MIDDLE) - min(CELL12_MIDDLE);
134  High_Falure(12) = max(CELL12_HIGH) - min(CELL12_HIGH);
135
136  Multiplikator = 5/2^ADC_Bit * 1000000;
137  Y = [Fluke_LOW_Binaer ;Fluke_MID_Binaer; Fluke_HIGH_Binaer];
138
139  for i= 1:12
140    A = [CELL_L_MEAN(i)*CELL_L_MEAN(i), CELL_L_MEAN(1), 1; CELL_M_MEAN(i)*CELL_M_MEAN(i
     ),CELL_M_MEAN(i),1;CELL_H_MEAN(i)*CELL_H_MEAN(i),CELL_H_MEAN(i),1 ];
141    X = inv(A)*Y;
```

```
142     Faktor1 = X(1)*Multiplikator*1000000;
143     Faktor2 = X(2)*Multiplikator*1000;
144     Faktor3 = X(3)*Multiplikator;
145     fprintf('%dA:%d     %dB:%d  %dC:%d   LowAbweichung:%d  MidAbweichung:%d
        HighAbweichung:%d\n',i,round(Faktor1),i,round(Faktor2),i,round(Faktor3),LOW_Falure(
        i),MID_Falure(i),High_Falure(i));
146   end
147
148   %% Überprüfung
149   % Übernahme der Ausgegebenen Daten
150   % alle Werte werden positiv angegeben
151   A      = 151;
152   B      = 87834;
153   C      = 418673;
154   Spannung  = [0.25039 0.50027 0.74883 1.0025 1.2507 1.5006 1.7522 2.0028 2.2513 2.5019
       2.75 3.0026 3.251 3.5 3.75 4]; % in V
155   ADC_CODE  = [1728 3490 5202 7058 9620 12408 15366 18400 21557 24800 27941 31196 34375
       37548 40739 43917]; % zur Spannung gehörender ADC Code
156   Input     = [0.25 0.5 0.75 1 1.25 1.5 1.75 2 2.25 2.5 2.75 3 3.255 3.5 3.75 4];  %
      ideale Kennlinie in V
157
158   % implementierte Kalibriergleichung
159   AusgabeKorekt = (ADC_CODE.*B - ADC_CODE.*ADC_CODE/1000*A)/1000 + C;
160
161   subplot(2,1,1); % Kalibrierte Übertragunsfunktion
162   plot(Input,Input,'*',Spannung,AusgabeKorekt./1000000,'r');
163   title('Ergebniss Kalibrierung');
164   legend('ideale Übertragungskennlinie','kalibrierte Kennlinie');
165   xlabel('ADC Eingang in V');
166   ylabel('ADC Ausgabe Codiert');
167   grid on;
168   subplot(2,1,2); % Fehler nach Kalibrierung
169   plot(Spannung,(Spannung-(AusgabeKorekt./1000000))*1000,'g');
170   title('Fehler nach Kalibrierung');
171   legend('quadratischer Korrekturfehler');
172   xlabel('ADC Eingang in V');
173   ylabel('Fehler in mV');
174   grid on;
```

KURZANLEITUNG V3

FÜR DEN

BATTERIE ZYKLIERPRÜFSTAND

INHALT

1 Übersicht über die Bedienoberfläche

1.) Das Frontpanel bietet mehrere Bedienschnittstellen und Anschlussmöglichkeiten. Das Frontpanel ist in der Abb. 1 abgebildet:

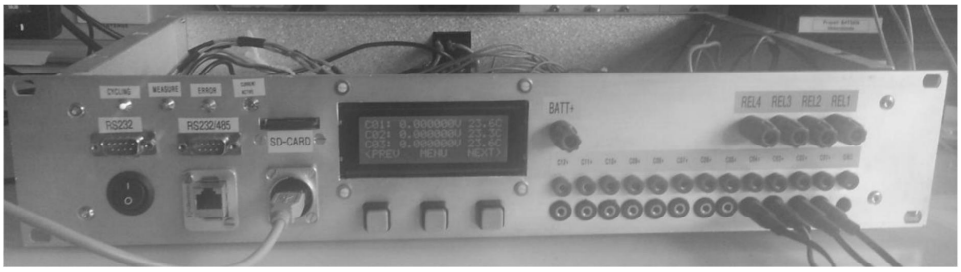


Abbildung 1: Ansicht Frontpanel

2.) Über den Ein-/Ausschalter (Abb. 2) lässt sich der Zyklierprüfstand in Betrieb nehmen. Die Stromversorgung wird der Elektronik zugeschaltet. An der Rückseite befindet sich ein zusätzlicher Schalter für die Netzspannung.

Abbildung 2: Linker Teil des Frontpanels

3.) Im SD-Karten Slot befindet sich eine SD-Karte, auf der die Konfigurationsdatei (Kap. 4) abgelegt sein muss. Die Messdaten werden auf diese SD-Karte gespeichert. Die maximale Größe der SD-Karte ist auf 4 GB beschränkt und SDHC Karten werden nicht unterstützt. Die Speicherkarte muss im FAT16 oder FAT32 formatiert sein.

4.) Die vier LEDs am Panel signalisieren den aktuellen Status des Zyklierprüfstandes (Tab. 1):

Tabelle 1: LEDs am Frontpanel

LED	Funktion
Cycling	Aktuell im Zyklierbetrieb (blinkend)
Measure	Signalisierung Aktion Messdatenaufnahme
ERROR	Fehler in der Software
Current Active	Lastkreis treibt Strom

5.) Tritt ein Fehler in der Software auf, so werden alle Lasten getrennt und die Messung aus sicherheitstechnischen Gründen gestoppt. Der Fehler wird mit der LED ERROR am Panel signalisiert.

6.) Die Anschlüsse RS232 & RS232/485 sind in der Software nicht aktiviert.

7.) Über ein USB-Kabel kann die Kommunikation über UART mit einem PC stattfinden (Kap. 3.1).

8.) Über ein ETHERNET-Anschluss kann über LAN eine Verbindung aufgebaut werden (Kap. 3.2).

Abbildung 3: Mittlerer Teil des Frontpanels

9.) Über das LC-Display (Abb. 3) erfolgt eine Informationsausgabe. Das Display kann über drei Taster bedient werden (Kap. 5).

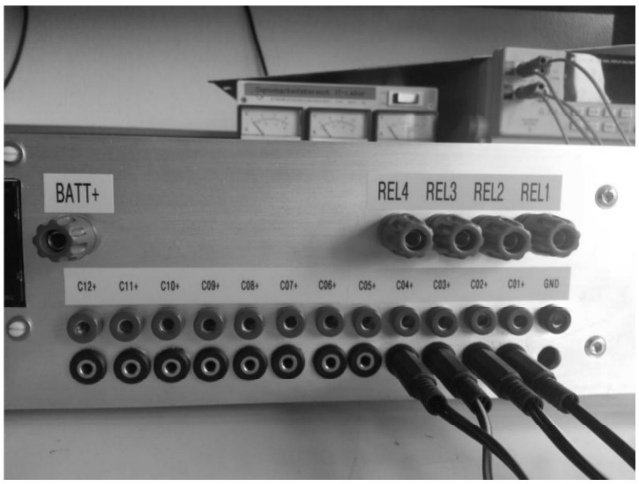


Abbildung 4: Rechter Teil des Frontpanels

10.) Im rechten Teil des Frontpanels befinden sich die Anschlüsse für den Batterieprüfling. Es sind die Anschlussvorgaben zu beachten (Kap. 2).

2 ANSCHLUSSVORGABEN FÜR DEN BATTERIEPRÜFLING

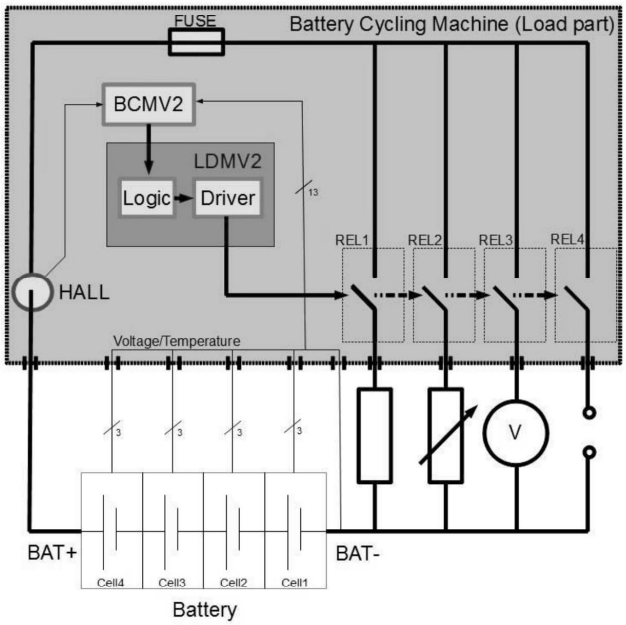


Abbildung 5: Interne Beschaltung der Lastrelais am Bsp. mit 4 Zellen

1.) Der Batterieprüfling wird zellenweise an die Messkontakte des Zyklierprüfstandes (rote Buchsen C01+ - C12+) angeschlossen. Ein Beispiel zeigt Abb. 5.

2.) Die Aufzählung der Zellen (1- 12) geht von der Masse der Batterie aus. Die Masse des Batterieprüflings wird als Null-Potential am GND-Messkontakt (schwarzen Buchse GND) angeschlossen. Es sind maximal 5 V Zellspannung erlaubt.

3.) Der Pluspol der gesamten Batterie wird an den Leistungsanschluss “BATT+“ des Zyklierpüfplatzes angeschlossen.

4.) Externe Geräte (Ladegerät, Entladegerät, etc.) werden an die Lastrelaisanschlüsse (REL1 - REL4) wahlweise mit dem positiven Pol angeschlossen.

5.) Masse des Batterieprüflings und externer Geräte werden außerhalb des Zyklierprüfplatzes zusammengeschaltet. Der Zyklierprüfplatz schaltet mit den Lastrelais das obere Spannungspotential auf eines der Geräte, wodurch der Stromfluss möglich wird.

6.) Der interne Lastkreis des Prüfstandes ist auf max. 30 A ausgelegt. Eine 30 A Sicherung ist im internen Lastkreis verbaut. Die neue größere Version kann an Relais 1 bis zu 500 A schalten. Hierfür ist ebenfalls eine Sicherung verbaut.

7.) Die NTC-Temperetursensoren werden über einen 3,5 mm Mono-Klinkenanschluss je Zelle angeschlossen und direkt an die jeweilige Batteriezelle montiert.

3 ANBINDUNG ÜBER USB UND ETHERNET

3.1. USB BZW. VIRTUELLE UART SCHNITTSTELLE ÜBER HTERM

1.) Über die virtuelle UART-Schnittstelle lässt sich mit einem Terminal Programm wie z.B. hTerm über Befehle der Zyklierprüfplatz steuern. Dabei muss das Terminal mit folgenden Einstellungen (Tab. 2) versehen werden:

Tabelle 2: Einstellungen UART-Schnittstelle

Baudrate	115200
Data	8 bit
Stop	1 bit
Parity	None
Newline	CR & LF
Option	DTR

2.) Es werden USB-Treiber für den Zyklierprüfstand benötigt. Die Treiber können auf der Homepage von FTDI gefunden werden. Es wird der FTDI-Treiber in der Version 2.06.00 benötigt.

3.) Am Anfang befindet sich der Zyklierprüfstand im Menü “Main“ (Abb. 6). Über den Befehl “help“ kann eine Auflistung der verfügbaren Befehle aufgerufen werden.

```
*********************************************
***         Battery Cycle Machine          ****
*********************************************

 Enter Command (type <help> to see list of commands):
Main: > help
Available commands
------------------
help    : Display list of commands
h       : alias for help
?       : alias for help
browser: SD-Card browser
config  : Configuration
start   : Start measurement cycling
stop    : Stop measurement cycling
```

Abbildung 6: Terminalausgabe im Menü 'Main'

4.) Tab. 3 listet den Befehlssatz im Menü “Main“ auf.

Tabelle 3: Befehlssatz im Kontextmenü ’Main’

Befehl	Beschreibung
help	Listet alle möglichen Befehle auf
h	Abkürzung für ’help’
?	Abkürzung für ’help’
browser	Aufrufen des Kontextmenü SD-Karten Browser (Tab. 4)
config	Aufrufen des Konfigurationsmenüs (Tab. 5)
alive	Abfrage, ob Zyklusmessung aktiv
start	Start der Zyklusmessung
stop	Manueler Stop der Zyklusmessung

5.) Über den Befehl “start“ kann eine zyklische Messdatenerfassung gestartet werden. Die Messung läuft über die in der Konfigurationsdatei eingestellte Gesamtdauer “Measuretime“

6.) Über den Befehl “stop“ kann eine zyklische Messdatenerfassung abgebrochen werden.

7.) Tab. 4 listet den Befehlssatz im Menü “SD-Card“ auf.

Tabelle 4: Befehlssatz im Kontextmenü ’SD-Card’

Befehl	Beschreibung
help	Listet alle möglichen Befehle auf
h	Abkürzung für ’help’
?	Abkürzung für ’help’
reinit	Re-Initialisierung der SD-Karte
ls	Auflistung der auf SD-Karte vorhandenen Dateien
chdir	Wechsel des Verzeichnisses auf der SD-Karte
cd	Abkürzung für ’chdir’
pwd	Wiedergabe des aktuellen Verzeichnisses
cat	Aufzeigen des Inhalts einer Datei (nur im Idle Mode)
cre	Erzeuge eine Datei
del	Lösche eine Datei
exit	Zurück zum Kontextmenü ’Main’

8.) Tab. 5 listet den Befehlssatz im Menü “Config“ auf.

Tabelle 5: Befehlssatz im Kontextmenü 'Config'

Befehl	Beschreibung
help	Listet alle möglichen Befehle auf
h	Abkürzung für 'help'
?	Abkürzung für 'help'
print	Wiedergabe der aktuellen Konfiguration
quantity	Setzte die Anzahl der zu messenden Zellen (1 - 12)
cyclestep	Setzte Zeitabstand zwischen zyklischer Messung in Sekunden (0-85)
gain	Setze kanalabhängig Verstärkungsfaktor für ADC (z.B. 'gain 1 76273500')
offset	Setze kanalabhängig Offset für ADC (z.B. 'offset 1 120')
calibrate	Null-Punkt Kalibrierung der Strommessung
ip	Setze IP-Adresse ('192.168.000.128')
relais	Manuel Lastrelais aktivieren (z.B. 'relais 1')
relsel	Wähle für die Ablaufsteuerung an entsprechender Stelle und Zeitpunkt ein Lastrelais (z.B. 'relsel 1 wait 120 3')
relqua	Setze die Anzahl der Lastrelaisumschaltzeiten (1 - 99)
meastime	Setze Laufzeit der gesamten Messung in Minuten (0-99999)
save	Speichere aktuelle Konfiguration auf die SD-Karte (überschreibt config.txt)
clock	Setzte die aktuelle Systemzeit und Datum (z.B. clock 10.12.2013 13:45)
vlmitis	Setze die obere und untere Zellspannungsgrenze in mV (z.B. vlimits 2500 3500)
maxcurr	Setzte die maximal erlaubte Stromstärke in mA (z.B. maxcurr 1500)
maxcycletime	Setze die maximale Zyklierdauer in Minuten (z.B. maxcycletime 1000)
maxtemp	Setze die maximale Zelltemperatur in °C (z.B. maxtemp 50)
maxcharge	Setze die maximale Ladungsmenge in mAh (z.B. maxcharge 500)
load	Lade die Konfiguration von der SD-Karte
exit	Zurück zum Kontextmenü 'Main'

9.) Über den Befehl *relsel* können verschiedene Zyklierschritte konfiguriert werden. Auf diese Weise können zeit- und auch spannungsabhängige Schaltereignisse vorgegeben werden. Eine Liste der möglichen Schritttypen, welche an die 2. Stelle des *relsel* Befehls geschrieben werden, ist nachfolgend dargestellt.

Zyklierschritt	Beschreibung
wait	Relative Wartezeit in Minuten
charge_cellV	Warten, bis eine der Zellen die angegebene Zellspannung erreicht hat
discharge_cellV	Warten, bis eine der Zellen die angegebene Zellspannung unterschritten hat
charge_batV	Warten, bis die Batterie die angegebene Gesamtspannung überschritten hat
discharge_batV	Warten, bis die Batterie die angegebene Gesamtspannung unterschritten hat

→ Beispiel: `relsel 3 charge_cellV 3000 1` → 3.Zyklierschritt: Zellspannungsladung bis 3 V und Relais 1 wird geschaltet.

10.) Über die Befehle geänderten Konfigurationsparameter werden nicht automatisch auf der SD-Karte gespeichert.Eine über das Menü geänderte Konfiguration kann über den Befehl “save“ auf der SD-Karte gespeichert werden. Die alte “config.txt“ wird dabei überschrieben oder falls nicht vorhanden, neu erstellt.

3.2. Ethernet Schnittstelle über VPN Ansprechen

1.) Zunächt muss unter https://connect.haw-hamburg.de der VPN Client „Cisco AnyConnect Secure Mobility Client” heruntergeladen werden.

2.) Nach der Installation wird der VPN Client gestartet. Der einzutragende VPN Server lautet: *„connect.haw-hamburg.de“*

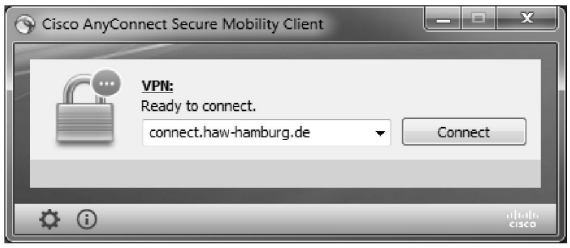


Abbildung 7: Start des VPN-Clients

3.) Zum Schluss findet die Authentifizierung mit dem HAW-Account statt.

Abbildung 8: Authentifizierung mit dem HAW-Account

4.) Nun ist eine VPN Verbindung zum HAW-Netzwerk aufgebaut. Es kann jetzt wie in Kapitel „3.3. Ethernet Schnittstelle über RealTerm“ verfahren werden.

3.3. ETHERNET SCHNITTSTELLE ÜBER REALTERM

1.) Unter http://realterm.sourceforge.net/ befindet sich die open source Software „RealTerm“. Es ist ein Serial/TCP Terminal Programm.

2.) Nach der Installation von „RealTerm“ wird es gestartet. Als erstes wird die „Display“ Einstellung vorgenommen. Dabei ist wichtig bei „newLine mode“ ein Häckchen zu setzen.

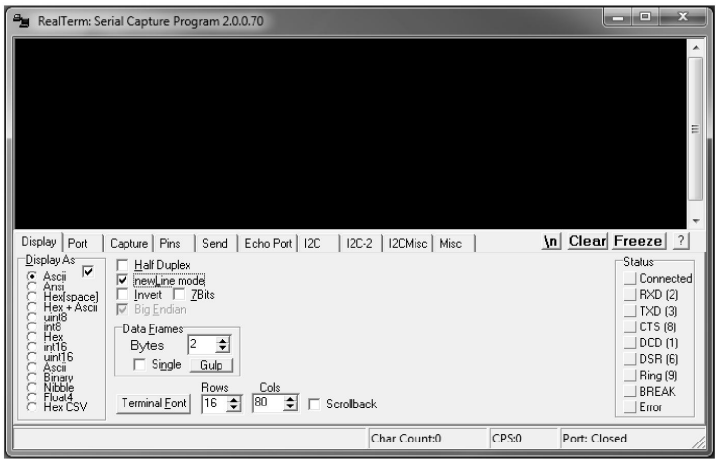


Abbildung 9: Display-Einstellungen von RealTerm

3.) Jetzt wird unter den Port-Einstellungen eine Verbindung aufgebaut. Als Port wird nun die IP-Adresse gefolgt vom Port eingetragen (141.22.14.199:56936). Mit dem Button „Change“ wird die Einstellung übernommen und die Verbindung aufgebaut. Sollte dies nicht sofort geschehen, Wmuss der Button „open“ durch zweifaches betätigen deaktviert und wieder aktiviert werden.

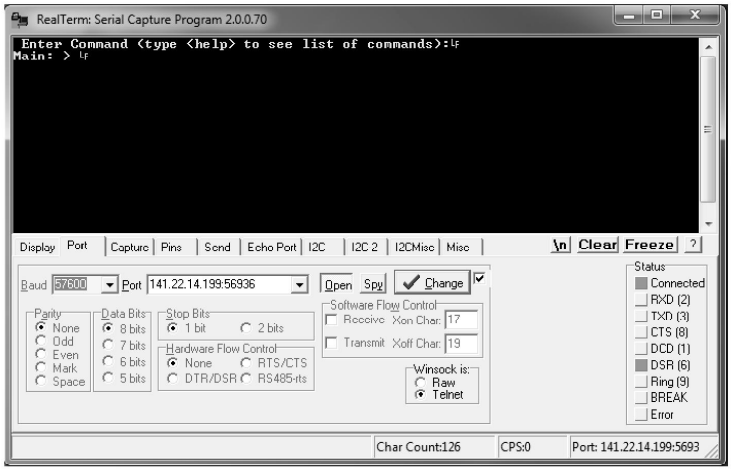


Abbildung 10: Port-Einstellungen von RealTerm

4.) Um nun einen Befehl über Ethernet zu übertragen muss zum Karteireiter „Send“ gewechselt werden. Dort sollten alle Häckchen, außer bei „+crc“, unter EOL (End of Line) gesetzt werden. Nun können Befehle eintragen werden und mit dem Button „Send ASCII“ übertragen werden.

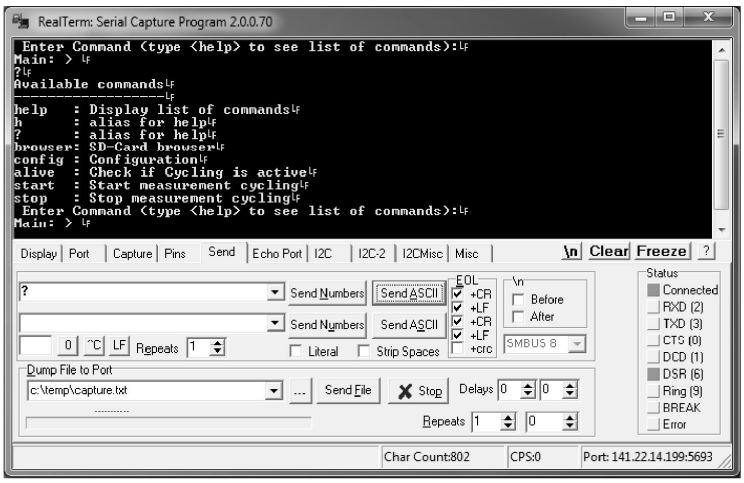


Abbildung 11: Übertragen eines Befehles über Ethernet mit RealTerm

3.4. ETHERNET SCHNITTSTELLE ÜBER MATLAB

1.) Über einen PC mit MatLab kann eine lokale TCP/IP Verbindung mit Zyklierprüfstand aufgebaut werden. Hierzu werden die entsprechenden MatLab Sourcefiles benötigt.

2.) Über den Befehl “connect(ip)“ kann mit Übergabe der Konfigurierten IP-Adresse des Zyklierprüfplatzes eine Verbindung in MatLab aufgebaut werden.

3.) Die Standard IP-Adresse lautet 141.22.14.199

4.) Der zu verwendende Port ist: 56936

5.) Folgende MatLab-Dateien werden benötigt, die während der Projektarbeit des Zyklierprüfstandes entstanden sind:

1. main.m
2. send_cmd.m
3. receive_cmd.m
4. connect.m

6.) Über MatLab und der Funktion “send_cmd(cons; sprintf(“cmd\n“))“ können Befehle (cmd) an die Zykliermaschine über Ethernet geschickt werden. Dabei können dieselben Befehle verwendet werden wie über UART (Kap. 3.1).

7.) In der Konsole von MatLab erfolgt die gleiche Ausgabe wie über ein Terminalprogram (z.B hTerm).

8.) Das Terminierungszeichen der Verbindung wurde auf ’LF’ festgesetzt.

9.) Die Unterstützung über Terminals wie Telnet ist nicht gewährleistet und nicht erprobt. Die Verwendung von MatLab wird empfohlen.

4 KONFIGURATIONSDATEI

1.) Über eine Konfigurationsdatei kann der Ablauf einer Zykliermessung ebenfalls konfiguriert werden. Die Datei muss den Dateinamen “config.txt“ besitzen und im ROOT-Verzeichnis der SD-Karte abgelegt sein.

2.) Ist keine Konfigurationsdatei vorhanden oder kann diese nicht geöffnet werden, so werden für alle Parameter Standardwerte in die Konfigurationsdatei geladen. Diese besteht aus einer Zyklierung mit nur einem Zyklierschritt, bei der keines der Relais geschaltet wird. Die Sicherheitsparameter werden auf die maximal Werte gelegt und alle 12 Zellen werden periodisch im Abstand von fünf Sekunden gemessen.

3.) Folgende Werte können in der Konfigurationsdatei eingestellt werden:

1. Die Anzahl der zu messenden Zellen
2. Zeitabstand zwischen den Messvorgängen in Sekunden
3. ADC Verstärkungsfaktor je Kanal
4. ADC Offsetwert je Kanal
5. Die IP-Adresse für die Ethernet Schnittstelle
6. Die gesamte Dauer der Messung in Minuten
7. Sicherheitsparameter der Zyklierung: Dauer, Zellspannung, Zelltemperatur und maximale Stromstärke
8. Zu schaltende Lastrelais zu den jeweiligen Umschaltzeitpunkten

4.) Die Formatierung der Konfigurationsdatei erfolgt auf Grundlage der Terminalbefehle aus Tabelle 5. Sie werden Zeile für Zeile in die Konfigurationsdatei geschrieben und es ist möglich Kommentarzeilen mit zwei Quertrichen beginnend (//....) einzufügen.

5.) Liegt keine Konfigurationsdatei auf der SD-Karte vor, kann vom Zyklierprüfplatz eine neue, blanke Konfigurationsdatei erzeugt werden (z.B. über Bedienoberfläche UART, Kap. 3). Darauf gilt es die Konfigurationsdatei durch ersetzten der 0-Werte anzupassen.

6.) Für das Editieren der Konfigurationsdatei wird notepad++ empfohlen. Anwendungen wie Wordpad oder der Editor von Windows bieten keine geeignete Darstellung der Datei und machen die Datei nach dem Abspeichern unbrauchbar für den Zyklierprüfstand.

7.) Die Werte der SD-Karte werden beim Einschalten des Zyklierprüfplatzes eingelesen und ins Programm geladen. Eine nachträgliche Änderung ist über UART oder Ethernet möglich.

Auf der folgenden Seit ist ein Beispiel der Konfigurationsdatei abgebildet.

```
//*********************************
// Konfigurationsdatei config.txt
//             Beispiel
//*********************************

//6 Zellen, 1 sek Messperiode
quantity 6
cyclestep 1

//Kalibrierwerte
gain 01 78060963
gain 02 78206692
gain 03 77986939
gain 04 78092611
gain 05 77817476
gain 06 77995647

offset 01 103
offset 02 128
offset 03 076
offset 04 094
offset 05 071
offset 06 085

//IP-Adresse
ip 141.22.14.199

//Zeitgrenzen
meastime 4740
maxcycletime 4140

//Grenzwerte zur Sicherheit
vlimits 1750 2350
maxcurr 20000
maxtemp 50
maxcharge 110000

//Zyklierablauf
//20 Grad
relsel 1 wait 60 0
relsel 2 wait 60 2
relsel 3 wait 120 0
relsel 4 wait 60 1
relsel 5 wait 180 0

//0 Grad
relsel 6 wait 60 2
relsel 7 wait 120 0
relsel 8 wait 60 1
relsel 9 wait 180 0
…
```

Kommentarzeile

vorhandene Terminalbefehle zur Konfiguration

Abbildung 12: Beispiel einer Konfigurationsdatei config.txt

5 LCD-DISPLAY UND INBETRIEBNAHME

(a) LED-Zustand: START (b) LED-Zustand: CLOCK (c) LED-Zustand: MEAS

(d) LED-Zustand: CELLA (e) LED-Zustand: BATTERY (f) LED-Zustand: IP

(g) LED-Zustand: CONF (h) LED-Zustand: CURRENT STEP

Abbildung 13: Display MENU

1.) Über das LC-Display des Zyklierprüfstandes können Informationen über die Messwerte und die aktuellen Zyklierparameter abgerufen werden.

2.) Die Funktion der Taster wird dabei in der untersten Zeile des Display dargestellt.

3.) Mit dem mittleren Taster kann durch die einzelnen Anzeigen Abb. 8a – 8h geschaltet werden. Der Zustand h kann nur während einer Zyklierung erreicht werden.

4.) Im Menü 'Measurment Cycling' (Abb. 8c) kann über den linken Taster "START" die zyklische Messdatenerfassung direkt gestartet werden. Ebenfalls kann dies über den Terminalbefehl *„start"* erfolgen. Danach wird die konfigurierte Zyklierung abgearbeitet und die Messwerte werden im konfigurierten Abstand aufgezeichnet und auf der SD-Karte gespeichert. Ebenfalls können die Messdaten über die Schnittstellen UART und Ethernet während der Zyklierung betrachtet werden. Ebenso kann über den rechten Taster "STOP" die zyklische Messdatenerfassung gestoppt werden. Die Eingabe "STOP" muss darauf mit "YES" bestätigt werden.

6.) In der Anzeige der aktuellen Zellenspannungen und -temperaturen (Abb. 8d) kann über den linken und rechten Taster zwischen den Zellen geblättert werden. Im Abschnit „BATTERY" (Abb. 8e) können die Parameter Batteriespannung, Stromstärke und geflossene Ladungsmenge betrachtet werden.

7.) Bei Überschreitung einer der Sicherheitsparameter wird die Zyklierung direkt beendet und es wird eine Warnmeldung auf dem Display ausgegeben. Nachfolgend sind drei der Warnmeldungen beim Überschreiten der Spannung, der Ladungsmenge oder der Temperatur dargestellt.

Abbildung 14:Warnmeldungen bei Überschreitung einer der Grenzwerte

6 BEISPIELMESSUNGEN

BEISPIELMESSUNG A

1.) Zunächst muss die Batterie in den Temperaturschrank und das Zykliersystem in den 19“ Datenschrank gestellt werden. Nun wird sich vergwissert, dass außer einer Stromversorgung und dem LAN-Kabel nichts an das Zykliersystem angeschlossen ist.

2.) Das Zykliersystem kann jetzt eingeschalten werden und die Batterie kann wie in „Abbildung 15: Verkabelung einer Blei-Säure Batterie“, angeschlossen werden.

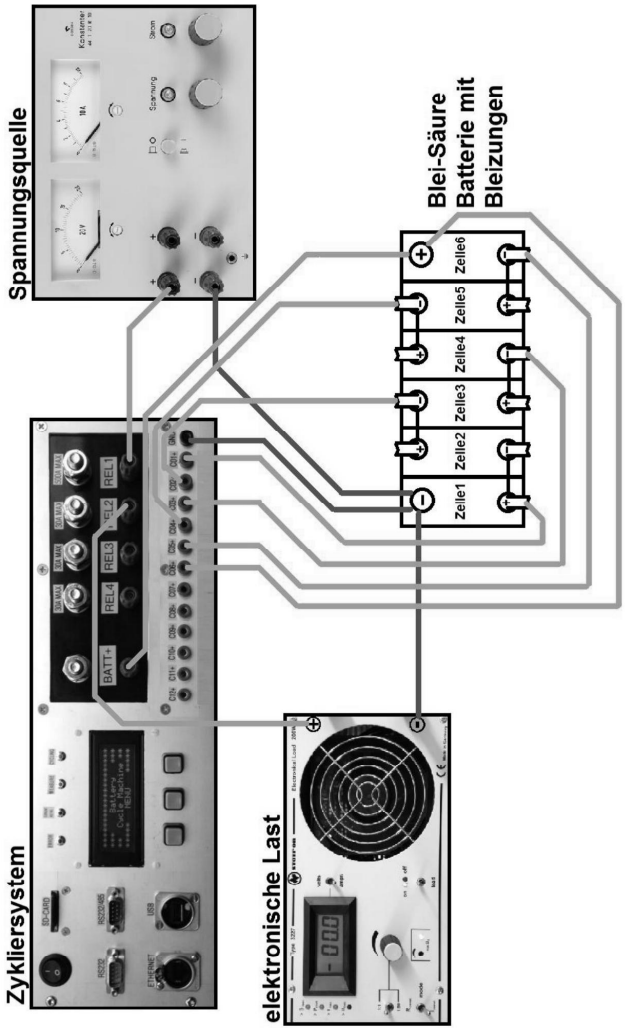


Abbildung 15: Verkabelung einer Blei-Säure Batterie

3.) Nun muss die Konfigurationsdatei „config.txt“ auf der SD-Karte richtig eingestellt werden. Es soll zunächst 6 Stunden geladen werden und anschließend nach einer 6 stündigen Pause die Batterie 6 Stunden entladen werden.

```
//************************************
//Konfigurationsdatei
//Erstellungsdatum: 15.08.2014
//************************************
quantity  6
cyclestep 1

//Zeitgrenzen
meastime 1900
maxcycletime 1900

//Grenzwerte zur Sicherheit
vlimits 1750 2350
maxcurr 10000
maxtemp 40
maxcharge 100000

//Zyklierablauf
relsel 01 wait 1 0
relsel 02 wait 360 1
relsel 03 wait 360 0
relsel 04 wait 360 2
relsel 05 wait 360 0
```

Abbildung 16: Einstellung einer Konfigurationsdatei „config.txt“ für eine Blei-Säure Batterie

4.) Jetzt kann die Messung gestartet werden. Nach einer Minute sollte der Ladevorgang beginnen.

BEISPIELMESSUNG B

1.) Zunächst muss die Batterie in den Temperaturschrank und das Zykliersystem in den 19“ Datenschrank gestellt werden. Nun wird sich vergwissert, dass außer einer Stromversorgung und dem LAN-Kabel nichts an das Zykliersystem angeschlossen ist.

2.) Das Zykliersystem kann jetzt eingeschalten werden und die Batterie kann wie in „Abbildung 17: Verkabelung nach dem NEFZ-Messkonzept“ angeschlossen werden.

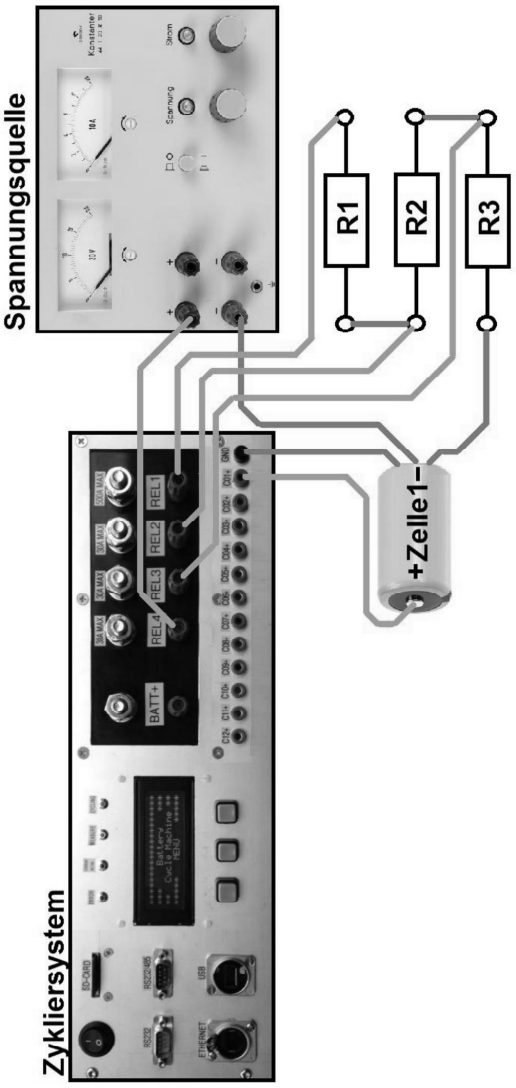


Abbildung 17: Verkabelung nach dem NEFZ-Messkonzept^

3.) Nun muss die Konfigurationsdatei „config.txt“ auf der SD-Karte richtig eingestellt werden.

```
//Konfigurationsdatei
quantity  1
cyclestep 1
meastime 420
maxcycletime 420
vlimits 2700 3600
maxcurr 20000
maxtemp 40
maxcharge 10000
relsel 01 wait 1 0
relsel 02 charge_cellV 3600 4
relsel 03 wait 30 0
relsel 04 wait 1 3
relsel 05 wait 1 2
relsel 06 wait 1 0
relsel 07 wait 1 3
relsel 08 wait 1 2
relsel 09 wait 1 0
relsel 10 wait 1 3
relsel 11 wait 1 2
relsel 12 wait 1 0
relsel 13 wait 1 3
relsel 14 wait 1 2
relsel 15 wait 1 0
relsel 16 wait 2 2
relsel 17 wait 2 3
relsel 18 wait 2 1
relsel 19 wait 30 0
relsel 20 wait 1 3
relsel 21 wait 1 2
relsel 22 wait 1 0
relsel 23 wait 1 3
relsel 24 wait 1 2
relsel 25 wait 1 0
relsel 26 wait 1 3
relsel 27 wait 1 2
relsel 28 wait 1 0
relsel 29 wait 1 3
relsel 30 wait 1 2
relsel 31 wait 1 0
relsel 32 wait 2 2
relsel 33 wait 2 3
relsel 34 wait 2 1
relsel 35 wait 30 0
```

Abbildung 18: Einstellung einer Konfigurationsdatei „config.txt“ für das NEFZ-Messkonzept

4.) Jetzt kann die Messung gestartet werden. Nach einer Minute sollte der Ladevorgang beginnen.

```
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 #endif /* TIMER_H_ */
26
```

```
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                                      Stellaris Ware
11
12  last modified:                      2014/01/13
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 File for Timer
25
26  */
27
28  /*****************************************************************************
29  *
30  * Includings
31  *
32  *****************************************************************************/
33  #include <string.h>
34  #include "driverlib/rom.h"
35  #include "third_party/fatfs/src/ff.h"
36  #include "third_party/fatfs/src/diskio.h"
37  #include "inc/hw_ints.h"
38  #include "inc/hw_memmap.h"
39  #include "inc/hw_types.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/sysctl.h"
46  #include "driverlib/timer.h"
47  #include <stdio.h>
48
49  /*****************************************************************************
50  *
51  * Own Includings
52  *
53  *****************************************************************************/
54  #include "utils/uartstdio.h"
55  #include "header/mysdcard.h"
56  #include "header/config.h"
57  #include "header/clocktimer.h"
58  #include "header/myadc.h"
59  #include "header/temperature.h"
```

```
60  #include "header/ethernet.h"
61  #include "header/control.h"
62  #include "header/relais.h"
63  #include "header/display.h"
64
65
66  // Global variables for systemclock data
67  volatile unsigned long clock_msec = 00;
68  volatile unsigned long clock_sec = 00;
69  volatile unsigned long clock_min = 00;
70  volatile unsigned long clock_hour = 18;
71  volatile unsigned long clock_day = 06;
72  volatile unsigned long clock_month = 03;
73  volatile unsigned long clock_year = 2013;
74
75  // Global variables for loadrelay timing control
76  volatile unsigned long timer2minute = 0;
77  volatile unsigned int power_relay_active = 0;
78  volatile unsigned int cycle_active = false;
79  volatile unsigned int measurement_active = false;
80
81  // Global variable for cycle control
82  volatile unsigned int current_cycle_step = 1;
83  volatile unsigned int wait_start_time = 0;
84  //Minute counter for maximum cycle time
85  volatile int max_time_counter = 0;
86  //global variable for battery charge
87  volatile int gBat_charge = 0;
88
89
90  //*********external variables**************
91
92  // TCP-Socket für die aktuelle Steuerungs-Verbindung
93  extern struct tcp_pcb* control_connection;
94
95  //Feed watchdog variable
96  extern volatile tBoolean g_bFeedWatchdog;
97
98  //defines the string variables for cycling types
99  extern const char *cycle_types[];
100
101 //Variable for the current
102 extern volatile unsigned int g_iCurrent;
103 //global variable for the current cell voltages
104 extern volatile unsigned int g_iCellVoltages[12];
105 //global variable for the current cell voltages
106 extern volatile int g_iCellTemperature[12];
107
108 //Flag for Display ReINIT
109 extern volatile int g_bDispInit;
110
111 //*****************************************************************************
112 //Extern variable for name of file for measurment saving
113 //*****************************************************************************
114 extern char MEAS_FILE[128];
115
116
117 //*****************************************************************************
118 // Set up and enable the timers.  Configure a timer
```

```
119 // for the measure interrupt handler.
120 // Additionally set up Timer for display refreshment at 15Hz.
121 // and minute counter.
122 //*****************************************************************************
123 void init_clock_timer(){
124
125     UARTprintf("Initializing Timer...");
126
127         //
128         // Enable the peripherals used by this example.
129         //
130         //Timer for Measurment-Interrupt
131         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
132         //Timer for LED-Display refreshment
133         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
134         //Timer for loadrelay timing control
135         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
136         //Timer for toggle cycle active LED
137         SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER3);
138
139         //
140         // Configure the 32-bit periodic timers.
141         //
142         //Configure Timer0A for defined time "Cyclestep" from config
143         TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
144         TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() * config.cyclestep);
145
146         //Configure Timer1A at 15 Hz
147         TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
148         TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/15);
149
150         //configure Timer2A interrupt for every minute
151         TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
152         TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()*60);
153
154
155         //configure Timer3A as 32 Bit Timer for LED toggling every 500ms
156         TimerConfigure(TIMER3_BASE, TIMER_CFG_PERIODIC);
157         TimerLoadSet(TIMER3_BASE, TIMER_A, SysCtlClockGet()/2);
158
159
160
161         //
162         // Enables Trigger for internal ADCs
163         //
164         TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
165
166         //
167         // Setup the interrupt for the timer timeout.
168         //
169         IntEnable(INT_TIMER0A);
170         TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
171
172         IntEnable(INT_TIMER1A);
173         TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
174
175         IntEnable(INT_TIMER2A);
176         TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
177
```

```
178             IntEnable(INT_TIMER3A);
179             TimerIntEnable(TIMER3_BASE, TIMER_TIMA_TIMEOUT);
180
181         UARTprintf("done\n");
182     }
183
184     //*****************************************************************************
185     // Function to reinitialize the Timer0A when variable "Cyclestep" reconfigured
186     //*****************************************************************************
187     void reinit_timer0(void){
188         TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() * config.cyclestep);
189     }
190
191
192     //*****************************************************************************
193     //
194     // The interrupt handler for the periodical measurement. Called depended on
195     // configured "Cyclestep".
196     //
197     //*****************************************************************************
198     void Timer0IntHandler(void)
199     {
200
201         //
202         // Clear the timer interrupt.
203         //
204         TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
205
206         // Do the measurement
207         do_measure();
208     }
209
210     //*****************************************************************************
211     //
212     // The interrupt handler for switching power relays and automatic stop of measurement.
213     // Called every minute.
214     //
215     //*****************************************************************************
216     void Timer2IntHandler(void)
217     {
218         int i;
219         //
220         // Clear the timer interrupt.
221         //
222         TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
223
224         //Increment minute counter
225         timer2minute++;
226
227         if(timer2minute == config.measuretime)//Stop measurment and cycling if
            measuretime reached, If measuretime = 0 INFINITE MEASUREMENT
228            Cmd_stop(0,NULL);
229
230         if(cycle_active){//follow cycle steps if cycling is active
231
232             //CYCLING STATEMACHINE
233             switch(config.cycle_steps[current_cycle_step].type){
234
235                 case WAIT:{
```

```
236
237                 if(timer2minute == (wait_start_time + config.cycle_steps[
                    current_cycle_step].param)){//check if relative waiting time reached
238                     current_cycle_step++;//next step
239                     if(current_cycle_step == (config.step_quantity + 1))//if step
                        quantity exceeded stop cycling
240                         stop_cycling();
241                     else{//SWITCH POWERRELAY for the next step
242                         power_relay_active = switch_power_relais(config.cycle_steps[
                            current_cycle_step].relay);
243                         wait_start_time = timer2minute;//set the waiting start time
                            for the wait step
244                     }
245                 }
246                 break;
247             }
248             case CHARGE_CELLV:{
249
250                 for(i = 0; i < config.quantity_cells; i++){//check if one of the
                    cells has reached charge end voltage
251                     if(g_iCellVoltages[i] >= (config.cycle_steps[current_cycle_step].
                        param * 1000)){//µV
252                         current_cycle_step++;
253                         if(current_cycle_step == (config.step_quantity + 1))//if
                            step quantity exceeded stop cycling
254                             stop_cycling();
255                         else{//SWITCH POWERRELAY for the next step
256                             power_relay_active = switch_power_relais(config.
                                cycle_steps[current_cycle_step].relay);
257                             wait_start_time = timer2minute;//set the waiting start
                                time for the wait step
258                         }
259                         break;
260                     }
261                 }
262                 break;
263             }
264             case DISCHARGE_CELLV:{
265
266                 for(i = 0; i < config.quantity_cells; i++){//check if one of the
                    cells has reached min voltage
267                     if(g_iCellVoltages[i] <= (config.cycle_steps[current_cycle_step].
                        param * 1000)){//µV
268                         current_cycle_step++;
269                         if(current_cycle_step == (config.step_quantity + 1))//if
                            step quantity exceeded stop cycling
270                             stop_cycling();
271                         else{//SWITCH POWERRELAY for the next step
272                             power_relay_active = switch_power_relais(config.
                                cycle_steps[current_cycle_step].relay);
273                         wait_start_time = timer2minute;//set the waiting start time
                            for the wait step
274                         }
275                         break;
276                     }
277                 }
278                 break;
279             }
280             case CHARGE_BATV:{
```

```
281
282                 int bat_voltage = 0;
283                 for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
284                     bat_voltage += g_iCellVoltages[i];
285                 //check if battery voltage has reached charge end voltage
286                 if(bat_voltage >= (config.cycle_steps[current_cycle_step].param *
                    1000)){//µV
287                     current_cycle_step++;
288                     if(current_cycle_step == (config.step_quantity + 1))//if step
                        quantity exceeded stop cycling
289                         stop_cycling();
290                     else{//SWITCH POWERRELAY for the next step
291                         power_relay_active = switch_power_relais(config.cycle_steps[
                            current_cycle_step].relay);
292                         wait_start_time = timer2minute;//set the waiting start time
                            for the wait step
293                     }
294                 }
295                 break;
296             }
297             case DISCHARGE_BATV:{
298
299                 int bat_voltage = 0;
300                 for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
301                     bat_voltage += g_iCellVoltages[i];
302                 //check if battery voltage has reached discharge end voltage
303                 if(bat_voltage <= (config.cycle_steps[current_cycle_step].param *
                    1000)){//µV
304                     current_cycle_step++;
305                     if(current_cycle_step == config.step_quantity)//if step quantity
                        exceeded stop cycling
306                         stop_cycling();
307                     else{//SWITCH POWERRELAY for the next step
308                         power_relay_active = switch_power_relais(config.cycle_steps[
                            current_cycle_step].relay);
309                         wait_start_time = timer2minute;//set the waiting start time
                            for the wait step
310                     }
311                 }
312                 break;
313             }
314             default:{//default statement
315                 stop_cycling();//safety turn off
316             }
317         }
318     }
319 }
320
321 //*****************************************************************************
322 // The interrupt handler for the LED toggling if cycling is active
323 //*****************************************************************************
324 void Timer3IntHandler(void){
325     //
326     // Clear the timer interrupt.
327     //
328     TimerIntClear(TIMER3_BASE, TIMER_TIMA_TIMEOUT);
329
330      if(cycle_active)
331         //Toggle cycle acitve LED
```

```
332          GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4, ~GPIOPinRead(GPIO_PORTJ_BASE,
             GPIO_PIN_4));
333       else
334          GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4, 0);
335   }
336
337
338
339   //*****************************************************************************
340   // Update the Systemclock and toggle cycle LED + check of maximum cycle time
341   //*****************************************************************************
342   void TimerClock(void)
343   {
344          clock_msec += 10;
345          if(clock_msec == 1000){
346              clock_msec = 0;
347              clock_sec++;
348          }
349
350          if (clock_sec == 60){
351              clock_sec = 0;
352              clock_min++;
353
354              if(cycle_active){
355                  max_time_counter++;
356                  if(max_time_counter == config.max_cycle_time){//If absolute maximum
                     measurement time reached shut down relays and cycling
357                      stop_cycling();
358                      handlerState = ERRTIME;//Display Error Message
359                      UARTprintf("\nMAXIMUM CYCLE TIME EXCEEDED!\n");
360                      if(control_connection)
361                          telnet_write("\nMAXIMUM CYCLE TIME EXCEEDED!\n");
362                  }
363              }
364
365              //raise Flag for Display ReINIT every minute
366              g_bDispInit = 1;
367          }
368
369          if(clock_min == 60){
370              clock_hour++;
371              clock_min = 0;
372          }
373          if(clock_hour == 24){
374              clock_day++;
375              clock_hour = 0;
376          }
377
378          if((clock_month == 1) || (clock_month == 3) || (clock_month == 5) || (
             clock_month == 7) || (clock_month == 8) || (clock_month == 10) || (
             clock_month == 12)){
379              if(clock_day == 32) {
380                  clock_month++;
381                  clock_day = 1;
382              }
383          }
384          if((clock_month == 4) || (clock_month == 6) || (clock_month == 9) || (
             clock_month == 11)){
385              if(clock_day == 31) {
```

```
386                 clock_month++;
387                 clock_day = 1;
388             }
389         }
390 
391         if(clock_month == 2){
392             int d_temp = 29;
393             if(clock_year%4 == 0)
394                 d_temp = 30;
395             if(clock_year%4 != 0)
396                 d_temp = 29;
397             if(clock_day == d_temp) {
398                 clock_month++;
399                 clock_day = 1;
400             }
401         }
402 
403         if(clock_month == 13){
404             clock_year++;
405             clock_month = 1;
406         }
407 }
408 
409 //*****************************************************************************
410 //
411 // This function implements the cycling safety switch off, measurement continues
412 //
413 //*****************************************************************************
414 void stop_cycling(void)
415 {
416     power_relay_active = switch_power_relais(0);//Deactivate power relays
417     cycle_active = false;//Deactivate cycling
418     wait_start_time = 0;//reset cycling variables
419     max_time_counter = 0;
420     current_cycle_step = 1;
421 
422 }
423 
424 
425 //*****************************************************************************
426 //
427 // This function implements the "start" command. It starts the measurement
428 // by activating Timer 0A.
429 //
430 //*****************************************************************************
431 int
432 Cmd_start(int argc, char *argv[])
433 {
434     int i = 0;
435 
436     //Start cycling only if not already running
437     if(!cycle_active && !measurement_active){
438 
439         //Generate Filename
440         sprintf(MEAS_FILE, "%02d%02d%02d%02d.txt", clock_month, clock_day, clock_hour
            , clock_min);
441 
442         //Generate header info
443         char header[64];
```

```
444         add_to_file(MEAS_FILE, "****************************************");
445
446         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);
447         add_to_file(MEAS_FILE, header);
448
449         sprintf(header, "\nQuantity Cells: %02d\nCyclestep: %02d sec", config.
            quantity_cells, config.cyclestep);
450         add_to_file(MEAS_FILE, header);
451
452         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);
453         add_to_file(MEAS_FILE, header);
454
455         sprintf(header, "\nMaximum cycle time: %05d min", config.max_cycle_time);
456         add_to_file(MEAS_FILE, header);
457
458         sprintf(header, "\nCell voltage limits: %05d mV , %05dmV", (int)(config.
            min_voltage/1000), (int)(config.max_voltage/1000));
459         add_to_file(MEAS_FILE, header);
460
461         sprintf(header, "\nMaximum current: %05d mA", config.max_current);
462         add_to_file(MEAS_FILE, header);
463
464         sprintf(header, "\nMaximum cell temperature: %05d °C", (int)(config.max_temp/
            10));
465         add_to_file(MEAS_FILE, header);
466
467         sprintf(header, "\nMaximum battery charge: %05d mAh", config.max_charge/3600);
468         add_to_file(MEAS_FILE, header);
469
470         sprintf(header, "\nCELL  | ADC Gain  |  ADC Offset");
471         add_to_file(MEAS_FILE, header);
472
473         for(i = 1; i <= 12; i++){
474             sprintf(header, "\n%02d:    %03d     %08d     %08d", i, config.A[i],
                config.B[i], config.C[i]);
475             add_to_file(MEAS_FILE, header);
476         }
477
478         sprintf(header, "\nCycle steps:  %02d", config.step_quantity);
479         add_to_file(MEAS_FILE, header);
480
481         sprintf(header, "\nStepID     Steptype        Parameter  Relayselect");
482         add_to_file(MEAS_FILE, header);
483
484         for(i = 1; i <= 100; i++){
485             if(i <= config.step_quantity){
486                 sprintf(header,"\n%02i         %-15s %05d      %01d", i, cycle_types[
                    config.cycle_steps[i].type - 1], config.cycle_steps[i].param, config.
                    cycle_steps[i].relay);
487                 add_to_file(MEAS_FILE, header);
488             } else{
489                 sprintf(header,"\n%02i         none            00000      0", i);
490                 add_to_file(MEAS_FILE, header);
491             }
492         }
493
```

```
494
495             add_to_file(MEAS_FILE, "\n*****************************************");
496             add_to_file(MEAS_FILE, "\nCE,YYYY,MM,DD,HH,MM,SS,VOL(uV),TEM,CUR(mA),R");
497             add_to_file(MEAS_FILE, "\n*****************************************\n");
498
499             //Enable Timer0A and Cycling Mode
500             TimerEnable(TIMER0_BASE, TIMER_A);
501             cycle_active = true;
502             measurement_active = true;
503
504             //turn on power relay for first cycle step and turn on LED
505             current_cycle_step = 1;
506             power_relay_active = switch_power_relais(config.cycle_steps[1].relay);
507
508             // Enable Timer 3 for LED toggle every 500ms
509             TimerEnable(TIMER3_BASE, TIMER_A);
510
511             UARTprintf("\nMeasurement cycling started");
512             if(control_connection){
513                 telnet_write("Measurement cycling started\n");
514             }
515
516             //Enable Timer2A for cycle control
517             TimerEnable(TIMER2_BASE, TIMER_A);
518         }
519         //
520         // Return success.
521         //
522         return(0);
523     }
524
525     //*****************************************************************************
526     //
527     // This function implements the "stop" command. It stops the measurement
528     // by deactivating Timer 0A and Timer 2A.
529     //
530     //*****************************************************************************
531     int
532     Cmd_stop(int argc, char *argv[])
533     {
534         int i = 0;
535         //Disable Timer0A and Timer2A
536         TimerDisable(TIMER0_BASE, TIMER_A);
537         TimerDisable(TIMER2_BASE, TIMER_A);
538
539         UARTprintf("\nMeasurement cycling stopped");
540         if(control_connection){
541             telnet_write("Measurement cycling stopped\n");
542         }
543
544         //Disable Cycling mode and deselect powerrelays
545         cycle_active = false;
546         measurement_active = false;
547
548         //Disable LED toggle and switch off LED
549         TimerDisable(TIMER3_BASE, TIMER_A);
550         GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);
551
552         //reset measurement and cycling values
```

```
553       wait_start_time = 0;
554       max_time_counter = 0;
555       timer2minute = 0;
556       current_cycle_step = 1;
557
558       gBat_charge = 0;
559       for(i = 1; i < 13; i++){
560           g_iCellVoltages[i]= 0;
561           g_iCellTemperature[i]= 0;
562       }
563       g_iCurrent = 0;
564
565
566       //switch off power relay
567       power_relay_active = switch_power_relais(0);
568
569       //Turn off Measurement-LED
570       GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  0x00);
571
572       //
573       // Return success.
574       //
575       return(0);
576   }
577
578
579
580
```

```
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 //defines for all the available step types
17 #define WAIT  1
18 #define CHARGE_CELLV 2
19 #define DISCHARGE_CELLV 3
20 #define CHARGE_BATV 4
21 #define DISCHARGE_BATV 5
22
23 //scruct defining a single cycle step
24 typedef struct
25 {
26     int type;
27     int param;
28     int relay;
29
30 } cycle_step_t;
31
32 // sizeof(config_t) must be an even number! Otherwise the program
33 // will fail to save and/or reload the configuration!
34 typedef struct
35 {
36     long quantity_cells;
37     long cyclestep;
38     int A[13];
39     int B[13];
40     int C[13];
41     int ip_a;
42     int ip_b;
43     int ip_c;
44     int ip_d;
45     int zerocurrentch1;
46     int zerocurrentch2;
47
48     int measuretime;
49     int max_cycle_time;
50
51     int step_quantity;
52     //includes all variables for cylcing
53     cycle_step_t cycle_steps[101];
54
55     //LIMIT values
56     int max_current;
57     int min_voltage;
58     int max_voltage;
59     int max_temp;
```

```
60      int max_charge;
61
62  } config_t;
63
64
65  extern config_t config;
66
67  /*****************************************************************************
68  *
69  * Function Declarations
70  *
71  *****************************************************************************/
72  void config_read(void);
73  int config_write(void);
74  void init_config(void);
75
76  int config_cmd_line(int argc, char *argv[]);
77  int Cmd_config_exit(int argc, char *argv[]);
78  int Cmd_gain       (int argc, char *argv[]);
79  int Cmd_offset     (int argc, char *argv[]);
80  int Cmd_calibrate  (int argc, char *argv[]);
81  int Cmd_print      (int argc, char *argv[]);
82  int Cmd_quantity   (int argc, char *argv[]);
83  int Cmd_cyclestep  (int argc, char *argv[]);
84  int Cmd_relsel     (int argc, char *argv[]);
85  int Cmd_relqua     (int argc, char *argv[]);
86  int Cmd_meastime   (int argc, char *argv[]);
87  int Cmd_save       (int argc, char *argv[]);
88  int Cmd_ip         (int argc, char *argv[]);
89  int Cmd_set_clock  (int argc, char *argv[]);
90  int Cmd_set_limits (int argc, char *argv[]);
91  int Cmd_set_current(int argc, char *argv[]);
92  int Cmd_max_time   (int argc, char *argv[]);
93  int Cmd_max_temp   (int argc, char *argv[]);
94  int Cmd_max_charge (int argc, char *argv[]);
95  int Cmd_load       (int argc, char *argv[]);
96
97  // Für die interne Nutzung
98  void config_erase(void);
99  void config_print_uart(void);
100 void config_write_struct(const char *conf_name, const char *conf_val);
101
102 void calibrate_adc(void);
103
104 #endif /* CONFIG_H_ */
105
106
```

```
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                                      Stellaris Ware
12
13  last modified:                      2014/09/08
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 for configure operations
26
27  */
28
29  //*****************************************************************************
30  *
31  * Includings
32  *
33  *****************************************************************************/
34  #include <stdio.h>
35  #include <stdlib.h>
36  #include <string.h>
37  #include "math.h"
38  #include "third_party/fatfs/src/ff.h"
39  #include "third_party/fatfs/src/diskio.h"
40  #include <utils/uartstdio.h>
41  #include <driverlib/ethernet.h>
42  #include <utils/lwiplib.h>
43  //*****************************************************************************
44  *
45  * Own Includings
46  *
47  *****************************************************************************/
48  #include "header/myadc.h"
49  #include "header/config.h"
50  #include "header/mysdcard.h"
51  #include "header/mycmdline.h"
52  #include "header/ethernet.h"
53  #include "header/control.h"
54  #include "header/relais.h"
55  #include "header/myadc.h"
56  #include "header/clocktimer.h"
57  #include "header/rtc.h"
58  #include "header/display.h"
59
```

```
60
61
62  // Define Name of Configurationfile
63  const char* CONFIG_FILE = "config.txt";
64
65
66  //*****************************************************************************
67  //
68  // Defines the size of the buffer that holds the input data.
69  //
70  //*****************************************************************************
71  #define CONF_BUF_SIZE 4096
72
73  //*****************************************************************************
74  //
75  // Defines the size of the buffer that holds the input data.
76  //
77  //*****************************************************************************
78  #define CONFIG_INPUT_DATA_SIZE    64
79
80  //*****************************************************************************
81  //
82  // A temporary data buffer used for input data
83  //
84  //*****************************************************************************
85  static char g_cTmpInputData[CONFIG_INPUT_DATA_SIZE];
86
87  //*****************************************************************************
88  //
89  // A global Flag initiating the configuration from sd card read
90  //
91  //*****************************************************************************
92  char g_ConfDone = 0;
93
94  //*****************************************************************************
95  //
96  // This is the table that holds the command names, implementing functions,
97  // and brief description.
98  //
99  //*****************************************************************************
100 tCmdLineEntry g_sConfigCmdTable[] =
101 {
102     { "help",       Cmd_help,     "     : Display list of commands" },
103     { "h",          Cmd_help,  "        : alias for help" },
104     { "?",          Cmd_help,  "        : alias for help" },
105     { "print",      Cmd_print,     "    : Print current Configuration" },
106     { "quantity",   Cmd_quantity,     " : Set quantity of cells (01 - 12)" },
107     { "cyclestep",  Cmd_cyclestep,     ": Set measurement period in sec (0000 -
        9999)" },
108     { "gain",       Cmd_gain,     "     : Set gain (eg. 'gain 01 76273500')" },
109     { "offset",     Cmd_offset,     "   : Set offset (eg. 'offset 01 -00100')" },
110     { "calibrate",  Cmd_calibrate,     ": Calibrate current sensor to zero" },
111     { "ip",         Cmd_ip,     "       : Set IP address (eg. 'ip 192.168.000.128')"
        },
112     { "relais",     Cmd_relais,     "   : Manualy activates choosen powerrelais
        (e.g. 'relais 1')"},
113     { "relsel",     Cmd_relsel,     "   : Set Relayselect (eg. 'relsel 01 wait 00001
        1')"},
114     { "relqua",     Cmd_relqua,     "   : Set quantity of Relay switches (01 - 99)"},
```

```
115       { "meastime",   Cmd_meastime, " : Set time of full measurement and cycling
          procedure in min (eg. 'time 1440')"},
116       { "save",       Cmd_save,     "     : Save configuration to SD-Card" },
117       { "clock",      Cmd_set_clock,  "    : Set the system clock, format: dd.mm.yyyy
          hh:mm (eg. 'clock 10.12.2013 13:45')" },
118       { "vlimits",    Cmd_set_limits,  "  : Set voltage min and max limits in mV (eg.
          'vlimits 500 4000')" },
119       { "maxcurr",    Cmd_set_current, "  : Set absolute maximum current limit in mA
          (eg. 'maxcurrent 1500')" },
120       { "maxcycletime",Cmd_max_time,    ": Set absolute maximum cycling time limit in
          min (eg. 'maxtime 1000')" },
121       { "maxtemp",    Cmd_max_temp,    "  : Set absolute maximum temperature limit in
          °C (eg. 'maxtemp 50')" },
122       { "maxcharge",  Cmd_max_charge,  ": Set absolute maximum charge taken through
          the battery in mAh (eg. 'maxcharge 500')" },
123       { "load",       Cmd_load,     "     : Load configuration from SD-Card and
          re-init" },
124       { "exit",       Cmd_config_exit,"     : Exit configuration operations" },
125       { 0, 0, 0 }
126   };
127
128   //holds the strings for the relsel comman0d
129   const char *cycle_types[] = {"wait","charge_cellV","discharge_cellV","charge_batV",
      "discharge_batV"};
130
131
132   //*********external variables*****************
133   extern tCmdLineEntry g_sMainCmdTable;
134   extern tCmdLineEntry *g_psCmdTable;
135
136   // string for main location
137   extern const char *g_cMainLocalBuf;
138
139   // Global location buffer
140   extern char *g_cLocalBuf;
141
142   //define string for config location
143   const char *g_cConfigLocalBuf = "Config";
144
145   // String buffer for print operations to UART and Ethernet
146   extern char print_buffer[1024];
147
148   // TCP-Socket für die aktuelle Steuerungs-Verbindung
149   extern struct tcp_pcb* control_connection;
150
151   // MAC Adresse für weitere Verwendung im Programm
152   extern unsigned char pucMACArray[8];
153
154   //Flag zeigt an, ob Konfigurationsdatein eingelsen
155   extern char g_ConfDone;
156
157
158   //*****************************************************************************
159   //Extern variables for Clock
160   //*****************************************************************************
161   extern volatile unsigned long clock_msec;
162   extern volatile unsigned long clock_sec;
163   extern volatile unsigned long clock_min;
164   extern volatile unsigned long clock_hour;
```

```
165  extern volatile unsigned long clock_day;
166  extern volatile unsigned long clock_month;
167  extern volatile unsigned long clock_year;
168
169
170  /*
171   * Initialisieren der Konfiguration.
172   */
173  void init_config(void){
174
175       UARTprintf("Initializing Configuration:\n");
176
177       // set Command line pointer to the beginning of the Config command structure
178       g_psCmdTable = &g_sConfigCmdTable[0];
179       // set location buffer to Browser
180       g_cLocalBuf = (char*)g_cConfigLocalBuf;
181
182       config_read();
183
184       // set Command line pointer to the beginning of the main command structure
185       g_psCmdTable = &g_sMainCmdTable;
186       // set location buffer to main
187       g_cLocalBuf = (char*)g_cMainLocalBuf;
188
189       UARTprintf("Initializing Configuration...done\n");
190
191  }
192
193
194
195  /*
196   * Löschen der Konfiguration
197   */
198  void config_erase(void)
199  {
200      delete_file(CONFIG_FILE);
201      create_file(CONFIG_FILE);
202  }
203
204  /*
205   * Auslesen der Konfiguration.
206   */
207  void config_read(void)
208  {
209      char buffer[ CONF_BUF_SIZE ];
210      int i, err_flag = 0;
211
212      //Load default values
213      config.quantity_cells = 12;
214      config.cyclestep = 5;
215
216      config.A[0]  = 0;
217      config.A[1]  = 161;
218      config.A[2]  = 156;
219      config.A[3]  = 161;
220      config.A[4]  = 161;
221      config.A[5]  = 166;
222      config.A[6]  = 159;
223      config.A[7]  = 157;
```

```
224       config.A[8]  = 161;
225       config.A[9]  = 154;
226       config.A[10] = 161;
227       config.A[11] = 156;
228       config.A[12] = 161;
229
230       config.B[0]  = 0;
231       config.B[1]  = 88321;
232       config.B[2]  = 88062;
233       config.B[3]  = 88318;
234       config.B[4]  = 88264;
235       config.B[5]  = 88565;
236       config.B[6]  = 88225;
237       config.B[7]  = 88096;
238       config.B[8]  = 88265;
239       config.B[9]  = 87973;
240       config.B[10] = 88321;
241       config.B[11] = 88111;
242       config.B[12] = 88334;
243
244       config.C[0]  = 0;
245       config.C[1]  = 413468;
246       config.C[2]  = 415377;
247       config.C[3]  = 413559;
248       config.C[4]  = 414109;
249       config.C[5]  = 411739;
250       config.C[6]  = 414200;
251       config.C[7]  = 415176;
252       config.C[8]  = 413973;
253       config.C[9]  = 416040;
254       config.C[10] = 413468;
255       config.C[11] = 415067;
256       config.C[12] = 413393;
257
258       config.ip_a = 192;
259       config.ip_b = 168;
260       config.ip_c = 000;
261       config.ip_d = 128;
262
263       config.step_quantity = 1;
264
265       config.cycle_steps[0].type = 0;
266       config.cycle_steps[0].param = 0;
267       config.cycle_steps[0].relay = 0;
268       config.cycle_steps[1].type = WAIT;
269       config.cycle_steps[1].param = 0;
270       config.cycle_steps[1].relay = 0;
271       for(i = 2; i <= 100; i++){
272           config.cycle_steps[i] = (cycle_step_t) {0};
273       }
274
275       config.measuretime = 0;
276       config.max_cycle_time = 0;
277
278       config.max_current = 100;
279       config.min_voltage = 0;
280       config.max_voltage = 5000000;
281       config.max_temp = 300;
282       config.max_charge = 100 * 3600;
```

```
283
284       //If Error reading CONFIG_GILE, load default Values
285       if( read_into_buffer( CONFIG_FILE, buffer ) ) {
286
287           UARTprintf("\nERROR opening %s", CONFIG_FILE);
288           UARTprintf("\nLoading default Values\n\n");
289
290           g_ConfDone = 1;
291           config_print_uart();
292           return;
293       }
294
295
296       //Read config data from SD-Card
297       char *tmp1,*tmp2;
298       tmp1 = tmp2 = buffer;//set pointers to the start of the file
299
300       while(*tmp1){//parse till end of config file
301
302           if(*tmp1 == '\n'){//Line Feed reached, execute command
303
304               *tmp1 = 0;//terminate string
305
306               if(strlen(tmp2) != 0){//skip empty lines
307                   if((*tmp2) == '/' && (*(tmp2 + 1)) == '/');//Ignore comment lines
308                   else if(CmdLineProcess(tmp2) == CMDLINE_BAD_CMD){//process line and
                       check for error
309                       UARTprintf("\nBad command in config file line: %s\nLoading
                           default values!\n", tmp2);
310                       err_flag = 1;
311                       break;
312                   }
313               }
314
315               tmp2 = (tmp1 + 1);//next element of the file
316           }
317
318           tmp1++;//next character
319
320           if((*tmp1) == 0){//end of config file reached
321               if(strlen(tmp2) != 0){//skip empty lines
322                   if((*tmp2) == '/' && (*(tmp2 + 1)) == '/');//Ignore comment lines
323                   else if(CmdLineProcess(tmp2) == CMDLINE_BAD_CMD){//execute last line
                       of the file
324                           UARTprintf("\nBad command in config file line: %s\nLoading
                               default values!\n", tmp2);
325                           err_flag = 1;
326                       }
327               }
328           }
329       }
330
331       if(err_flag){
332
333           handlerState = ERROR;
334
335           //Load default values
336           config.quantity_cells = 12;
337           config.cyclestep = 5;
```

```
338
339         config.A[0]  = 0;
340         config.A[1]  = 161;
341         config.A[2]  = 156;
342         config.A[3]  = 161;
343         config.A[4]  = 161;
344         config.A[5]  = 166;
345         config.A[6]  = 159;
346         config.A[7]  = 157;
347         config.A[8]  = 161;
348         config.A[9]  = 154;
349         config.A[10] = 161;
350         config.A[11] = 156;
351         config.A[12] = 161;
352
353         config.B[0]  = 0;
354         config.B[1]  = 88321;
355         config.B[2]  = 88062;
356         config.B[3]  = 88318;
357         config.B[4]  = 88264;
358         config.B[5]  = 88565;
359         config.B[6]  = 88225;
360         config.B[7]  = 88096;
361         config.B[8]  = 88265;
362         config.B[9]  = 87973;
363         config.B[10] = 88321;
364         config.B[11] = 88111;
365         config.B[12] = 88334;
366
367         config.C[0]  = 0;
368         config.C[1]  = 413468;
369         config.C[2]  = 415377;
370         config.C[3]  = 413559;
371         config.C[4]  = 414109;
372         config.C[5]  = 411739;
373         config.C[6]  = 414200;
374         config.C[7]  = 415176;
375         config.C[8]  = 413973;
376         config.C[9]  = 416040;
377         config.C[10] = 413468;
378         config.C[11] = 415067;
379         config.C[12] = 413393;
380
381         config.ip_a = 192;
382         config.ip_b = 168;
383         config.ip_c = 000;
384         config.ip_d = 128;
385
386         config.step_quantity = 1;
387
388         config.cycle_steps[0].type = WAIT;
389         config.cycle_steps[0].param = 0;
390         config.cycle_steps[0].relay = 0;
391         config.cycle_steps[1].type = WAIT;
392         config.cycle_steps[1].param = 0;
393         config.cycle_steps[1].relay = 0;
394         for(i = 2; i <= 100; i++){
395             config.cycle_steps[i] = (cycle_step_t) {0};
396         }
```

```
397
398             config.measuretime = 0;
399             config.max_cycle_time = 0;
400
401             config.max_current = 100;
402             config.min_voltage = 0;
403             config.max_voltage = 5000000;
404             config.max_temp = 300;
405             config.max_charge = 100 * 3600;
406         }
407
408         g_ConfDone = 1;//Configuration done
409         config_print_uart();
410
411         return;
412     }
413
414
415     /*
416      * Schreiben der aktuellen Konfiguration in die config.txt (SD-Karte).
417      */
418     int config_write(void)
419     {
420         int i = 0;
421
422         delete_file(CONFIG_FILE);
423         create_file(CONFIG_FILE);
424
425         char write_quantity_cells[3];
426         sprintf(write_quantity_cells, "%02i", config.quantity_cells);
427         config_write_struct("quantity ", write_quantity_cells );
428
429         char write_cyclestep[5];
430         sprintf(write_cyclestep, "%04i", config.cyclestep);
431         config_write_struct("cyclestep ", write_cyclestep );
432
433         add_to_file(CONFIG_FILE, "\n");
434
435         char write_adc_A[32];
436         for(i = 1; i <= 12; i++){
437             sprintf(write_adc_A, "%02i %08d", i, config.A[i]);
438             config_write_struct("gain ", write_adc_A );
439         }
440
441         add_to_file(CONFIG_FILE, "\n");
442
443         char write_B[32];
444         for(i = 1; i <= 12; i++){
445             sprintf(write_B, "%02i %08d", i, config.B[i]);
446             config_write_struct("gain ", write_B );
447         }
448
449         add_to_file(CONFIG_FILE, "\n");
450
451         char write_adc_C[32];
452         for(i = 1; i <= 12; i++){
453             sprintf(write_adc_C, "%02i %06d ", i, config.C[i]);
454             config_write_struct("gain ", write_adc_C );
455         }
```

```
456
457       add_to_file(CONFIG_FILE, "\n");
458
459       char write_ip_address[48];
460       sprintf(write_ip_address, "%03d.%03d.%03d.%03d", config.ip_a, config.ip_b, config
          .ip_c, config.ip_d);
461       config_write_struct("ip ", write_ip_address );
462
463       add_to_file(CONFIG_FILE, "\n");
464
465       char write_measure_time[32];
466       sprintf(write_measure_time,  "%05d", config.measuretime);
467       config_write_struct("meastime ", write_measure_time);
468
469       sprintf(write_measure_time, "%d", config.max_cycle_time);
470       config_write_struct("maxcycletime ", write_measure_time);
471
472       add_to_file(CONFIG_FILE, "\n");
473
474       char write_limits[32];
475       sprintf(write_limits, "%d %d", config.min_voltage, config.max_voltage);
476       config_write_struct("vlimits ", write_limits);
477
478       sprintf(write_limits, "%d", config.max_current);
479       config_write_struct("maxcurr ", write_limits);
480
481       sprintf(write_limits, "%d", (int)(config.max_temp/10));
482       config_write_struct("maxtemp ", write_limits);
483
484       sprintf(write_limits, "%d", (int)(config.max_charge/3600));
485       config_write_struct("maxcharge ", write_limits);
486
487       char write_relay_control[32];
488       for(i = 1; i <= config.step_quantity; i++){
489               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);
490               config_write_struct("relsel ", write_relay_control);
491       }
492
493       return 1;
494   }
495
496   /*
497    * Hilfsfunktion zum strukturieren der Konfigurationsdatei
498    */
499   void config_write_struct(const char *conf_name, const char *conf_val){
500
501       add_to_file(CONFIG_FILE, conf_name);
502       add_to_file(CONFIG_FILE, conf_val);
503       add_to_file(CONFIG_FILE, "\n");
504
505   }
506
507   /*
508    * Ausgabe der Konfiguration Ueber UART und Ethernet.
509    * Gibt tabellarisch alle Werte der aktuellen Konfiguration über
510    * die UART und ggbf. vorhandene Ethernet Verbindung aus.
511    */
```

```
512 void config_print_uart(void)
513 {
514     int i = 0;
515 
516     UARTprintf("\n  Quantity_cells: %d Cells", config.quantity_cells);
517     UARTprintf("\n  Cyclestep:      %d sec", config.cyclestep);
518     UARTprintf("\n  IP Address:     %03d.%03d.%03d.%03d", config.ip_a, config.ip_b,
        config.ip_c, config.ip_d);
519     UARTprintf("\n\n    Cell |    ADC A    |    ADC B   |    ADC C   \n");
520 
521     for(i = 1; i <= config.quantity_cells; i++){
522         UARTprintf("\n     %02d:    %03d     %08d     %08d", i, config.A[i], config.B
            [i], config.C[i]);
523     }
524 
525     UARTprintf("\n\nMeasuretime (min):  %d", config.measuretime);
526 
527     UARTprintf("\n\nMaximum cycling time (min):  %d", config.max_cycle_time);
528 
529     UARTprintf("\n\nVoltage limits (mV):  min:%d  max:%d", (int)(config.min_voltage/
        1000), (int)(config.max_voltage/1000));
530 
531     UARTprintf("\n\nMaximum current (mA):  %d", config.max_current);
532 
533     UARTprintf("\n\nMaximum cell temperature (°C):  %d", (int)(config.max_temp/10));
534 
535     UARTprintf("\n\nMaximum battery charge (mAh):  %d", config.max_charge/3600);
536 
537     UARTprintf("\n\nCycle steps:  %d", config.step_quantity);
538 
539     UARTprintf("\n\nStepID     Steptype        Parameter  Relayselect");
540 
541     for(i = 1; i <= config.step_quantity; i++){
542         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);
543         UARTprintf(print_buffer);
544     }
545 
546     UARTprintf("\n\n");
547 
548 
549 
550     if(control_connection){
551 
552         telnet_write("\n");
553         sprintf(print_buffer, "  Quantity_cells: %d Cells\n", config.quantity_cells);
554         telnet_write(print_buffer);
555         sprintf(print_buffer, "  Cyclestep:      %d sec\n", config.cyclestep);
556         telnet_write(print_buffer);
557         sprintf(print_buffer, "  IP Address:     %03d.%03d.%03d.%03d\n", config.ip_a,
             config.ip_b, config.ip_c, config.ip_d);
558         telnet_write(print_buffer);
559         telnet_write("\n");
560         sprintf(print_buffer, "    Cell |  ADC Gain | ADC Offset\n");
561         telnet_write(print_buffer);
562         telnet_write("\n");
563         telnet_write("\n");
564 
```

```
565           sprintf(print_buffer, "");
566           char buf[64];
567           for(i = 1; i <= config.quantity_cells; i++){
568               sprintf(buf, "  %02d:    %03d     %08d     %08d\n", i, config.A[i],
                  config.B[i], config.C[i]);
569               strcat(print_buffer, buf);
570           }
571
572           telnet_write(print_buffer);
573
574           telnet_write("\n");
575           telnet_write("\n");
576
577           sprintf(print_buffer, "  Measuretime (min): %d\n", config.measuretime);
578           telnet_write(print_buffer);
579
580           sprintf(print_buffer, "\n\nMaximum cycling time (min): %d", config.
              max_cycle_time);
581           telnet_write(print_buffer);
582
583           sprintf(print_buffer, "\n\nVoltage limits (mV):  min:%d   max:%d", (int)(
              config.min_voltage/1000), (int)(config.max_voltage/1000));
584           telnet_write(print_buffer);
585
586           sprintf(print_buffer,"\n\nMaximum current (mA): %d", config.max_current);
587           telnet_write(print_buffer);
588
589           sprintf(print_buffer,"\n\nMaximum cell temperature (°C): %d", (int)(config.
              max_temp/10));
590           telnet_write(print_buffer);
591
592           sprintf(print_buffer,"\n\nMaximum battery charge (mAh):  %d", config.
              max_charge/3600);
593           telnet_write(print_buffer);
594
595           telnet_write("\n");
596           telnet_write("\n");
597
598           sprintf(print_buffer, "  Cyclesteps  : %01d\n", config.step_quantity);
599           telnet_write(print_buffer);
600
601           telnet_write("\n");
602
603
604           sprintf(print_buffer,"\n\nStepID     Steptype       Parameter  Relayselect");
605           telnet_write(print_buffer);
606
607           for(i = 1; i <= config.step_quantity; i++){
608               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);
609               telnet_write(print_buffer);
610           }
611
612
613           telnet_write("\n");
614           telnet_write("\n");
615
616       }
```

```
617
618
619
620
621 }
622
623 //*****************************************************************************
624 //
625 // Implementation of an command line for configure operations based on stellarisware
    sd-card example
626 //
627 //*****************************************************************************
628 int config_cmd_line(int argc, char *argv[])
629 {
630
631     // set Command line pointer to the beginning of the Config command structure
632     g_psCmdTable = &g_sConfigCmdTable[0];
633
634     // set location buffer to Browser
635     g_cLocalBuf = (char*)g_cConfigLocalBuf;
636
637     return(0);
638 }
639
640 //*****************************************************************************
641 //
642 // Implementation of an command to exit to main command line entry
643 //
644 //*****************************************************************************
645 int Cmd_config_exit(int argc, char *argv[])
646 {
647     // set Command line pointer to the beginning of the main command structure
648     g_psCmdTable = &g_sMainCmdTable;
649
650     // set location buffer to main
651     g_cLocalBuf = (char*)g_cMainLocalBuf;
652
653     return(0);
654 }
655
656 //*****************************************************************************
657 //
658 // Implementation of an command to set the gain value
659 //
660 //*****************************************************************************
661 int Cmd_gain(int argc, char *argv[])
662 {
663
664     if(argc == 3){//takes only 2 parameters
665         int cell, gain;
666         //
667         // Copy the first input data into buffer.
668         //
669         strcpy(g_cTmpInputData, argv[1]);
670
671         cell = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
672
673         if(cell > 0 && cell < 13){//check borders
674             //
```

```
675             // Copy the second input data into buffer.
676             //
677             strcpy(g_cTmpInputData, argv[2]);
678 
679             gain = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
680 
681             if(gain > 0){
682                 //config.gain[cell] = gain;
683                 return 0;//success
684             }
685         }
686     }
687     return -1;//Bad command
688 }
689 
690 //*****************************************************************************
691 //
692 // Implementation of an command to set the offset value
693 //
694 //*****************************************************************************
695 int Cmd_offset(int argc, char *argv[])
696 {
697 
698     if(argc == 3){//takes only 2 parameters
699         int cell;
700 
701         //
702         // Copy the first input data into buffer.
703         //
704         strcpy(g_cTmpInputData, argv[1]);
705 
706         //
707         // Assign cell
708         //
709         cell = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
710 
711         if(cell > 0 && cell < 13){//check borders
712             //
713             // Copy the second input data into buffer.
714             //
715             strcpy(g_cTmpInputData, argv[2]);
716 
717 
718             //
719             // Configure negative offset to assigned cell
720             //
721 
722             if(g_cTmpInputData[0] == '-' ){
723                 //config.offset[cell] = (int) strtol(g_cTmpInputData + 1, (char
                    **)NULL, 10);
724             }
725 
726             //
727             // Configure positive offset to assigned cell
728             //
729             if(g_cTmpInputData[0] != '-' ){
730                 //config.offset[cell] = (int) strtol(g_cTmpInputData, (char **)NULL,
                    10);
731             }
```

```
732
733             return(0);
734         }
735     }
736     return -1; //bad command
737 }
738
739 //*****************************************************************************
740 //
741 // Implementation of an command to autocalibrate hall sensor offset
742 //
743 //*****************************************************************************
744 int Cmd_calibrate(int argc, char *argv[])
745 {
746
747     //
748     // Calibrate and set offset for current
749     //
750     config.zerocurrentch1 = 0;
751     config.zerocurrentch2 = 0;
752
753     config.zerocurrentch1 = (32770 - adc_get_single_value_ch2());
754     config.zerocurrentch2 = (32770 - adc_get_single_value_ch3());
755
756
757     return(0);
758
759 }
760
761 //*****************************************************************************
762 //
763 // Implementation of an command to set the up-address
764 //
765 //*****************************************************************************
766 int Cmd_ip(int argc, char *argv[])
767 {
768     if(argc == 2){//takes only one parameter
769         char *tmp1,*tmp2, i;
770         int ip, flag = 0;
771         //
772         // Copy the first input data into buffer.
773         //
774         strcpy(g_cTmpInputData, argv[1]);
775
776         tmp1 = tmp2 = g_cTmpInputData;
777
778         for (i = 0; i < 3; i++){
779             while(*tmp1++ != '.');//look for the separating dot sign
780             *(tmp1 - 1) = 0;//write trailing string null
781             if(i == 0){
782                 ip = (int) strtol(tmp2, (char **)NULL, 10);
783                 if(ip >= 0 && ip <= 255)//check borders
784                     config.ip_a = ip;//first argument
785                 else {flag = 1;break;}
786             }else if(i == 1){
787                 ip = (int) strtol(tmp2, (char **)NULL, 10);
788                 if(ip >= 0 && ip <= 255)//check borders
789                     config.ip_b = ip;//second argument
790                 else {flag = 1;break;}
```

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

```
849 }
850
851
852 //*****************************************************************************
853 //
854 // Implementation of an command to set the time between measurment ("Cyclestep")
855 //
856 //*****************************************************************************
857 int Cmd_cyclestep(int argc, char *argv[])
858 {
859     if(argc == 2){
860         //
861         // Copy the input data into buffer.
862         //
863         strcpy(g_cTmpInputData, argv[1]);
864
865         long data = (long) strtol(g_cTmpInputData, (char **)NULL, 10);
866
867         if(data > 0 && data <= 85){//check borders
868             config.cyclestep = data;
869
870             if(g_ConfDone)
871                 reinit_timer0();//Reinit timer only if not in configuration from sd
                    card file
872
873             return 0;//success
874         }
875     }
876     return -1; //Bad command
877 }
878
879 //*****************************************************************************
880 //
881 // Implementation of an command to set the measurement time
882 //
883 //*****************************************************************************
884 int Cmd_meastime(int argc, char *argv[])
885 {
886     if(argc == 2){
887         int time;
888         //
889         // Copy the input data into buffer.
890         //
891         strcpy(g_cTmpInputData, argv[1]);
892
893         time = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
894         if(time > 0){
895             config.measuretime = time;
896             return 0;
897         }
898     }
899     return -1;
900
901 }
902
903 //*****************************************************************************
904 //
905 // Implementation of an command to set the quantity loadrelay switches
906 //
```

```
907 //*****************************************************************************
908 int Cmd_relqua(int argc, char *argv[])
909 {
910     if(argc == 2){
911         int step_qua;
912         //
913         // Copy the input data into buffer.
914         //
915         strcpy(g_cTmpInputData, argv[1]);
916 
917         step_qua = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
918         if(step_qua > 0 && step_qua <= config.step_quantity){
919             int i;
920             config.step_quantity = step_qua;
921             //set Relsel values more than actual relay quantity to zero
922             for(i = (config.step_quantity + 1); i < 101; i++)
923                 config.cycle_steps[i] = (cycle_step_t) {0};
924 
925             return 0;
926         }
927     }
928     return -1;
929 }
930 
931 //*****************************************************************************
932 //
933 // Implementation of an command to set the selection of a loadrelay by given time
934 //
935 //*****************************************************************************
936 int Cmd_relsel(int argc, char *argv[])
937 {
938     if(argc == 5){//takes only 4 parameters
939         int stepId;
940 
941         //
942         // Copy the first input data into buffer.
943         //
944         strcpy(g_cTmpInputData, argv[1]);
945 
946         //
947         // Assign relayid
948         //
949         stepId = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
950         if(stepId > 0 && (stepId <= config.step_quantity + 1)){
951             char type, flag = false;
952 
953             //
954             // Copy the second input data into buffer.
955             //
956             strcpy(g_cTmpInputData, argv[2]);
957 
958             for(type = 1; type < 6; type++){
959                 if(strcmp(cycle_types[type - 1], g_cTmpInputData) == 0){
960                     flag = true;
961                     break;
962                 }
963             }
964             if(flag){
965                 int param;
```

```
966
967                 //
968                 // Copy the third input data into buffer.
969                 //
970                 strcpy(g_cTmpInputData, argv[3]);
971
972                 param = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
973                 if(param >= 0){
974                     int relay;
975
976                     if(type == WAIT)//check borders for each type
977                         if(param > 99999)
978                             return -1;
979                     else if(type == CHARGE_CELLV)
980                         if(param > 5000)
981                             return -1;
982                     else if(type == DISCHARGE_CELLV)
983                         if(param > 5000)
984                             return -1;
985                     else if(type == CHARGE_BATV)
986                         if(param > 60000)
987                             return -1;
988                     else if(type == DISCHARGE_BATV)
989                         if(param > 60000)
990                             return -1;
991
992                     //
993                     // Copy the fourth input data into buffer.
994                     //
995                     strcpy(g_cTmpInputData, argv[4]);
996                     relay = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
997
998                     if(relay >= 0 && relay < 5){
999                         config.cycle_steps[stepId].type = type;
1000                        config.cycle_steps[stepId].param = param;//µV
1001                        config.cycle_steps[stepId].relay = relay;
1002
1003                        if(stepId == config.step_quantity + 1)
1004                            config.step_quantity++;
1005
1006                        return 0;//success
1007                    }
1008
1009                }
1010            }
1011        }
1012    }
1013    return -1;
1014}
1015
1016
1017//*****************************************************************************
1018//
1019// Implementation of an command to save current configuration to SD-Card
1020//
1021//*****************************************************************************
1022int Cmd_save(int argc, char *argv[])
1023{
1024    config_write();
```

```
1025        return 0;
1026    }
1027
1028    //*****************************************************************************
1029    //
1030    // Implementation of an command to load the configuration file from SD-Card
1031    // and reinitialize
1032    //
1033    //*****************************************************************************
1034    int Cmd_load(int argc, char *argv[])
1035    {
1036        config_read();
1037        return 0;
1038    }
1039
1040    //*****************************************************************************
1041    //
1042    // Implementation of an command to set the current system clock and RTC
1043    //
1044    //*****************************************************************************
1045    int Cmd_set_clock(int argc, char *argv[])
1046    {
1047        if(argc == 3){//takes only 2 parameters
1048            strcpy(g_cTmpInputData, argv[1]);
1049            clock_day =    10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
1050
1051            clock_month =  10 * (g_cTmpInputData[3] - '0') + (g_cTmpInputData[4] - '0') ;
1052
1053            clock_year =   1000 * (g_cTmpInputData[6] - '0') + 100 * (g_cTmpInputData[7]
                - '0') + 10 * (g_cTmpInputData[8] - '0') + (g_cTmpInputData[9] - '0');
1054
1055            strcpy(g_cTmpInputData, argv[2]);
1056            clock_hour =   10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
1057
1058            clock_min =    10 * (g_cTmpInputData[3] - '0') + (g_cTmpInputData[4] - '0') ;
1059
1060            clock_sec =    0;
1061
1062            set_rtc_values(clock_year - 2000, clock_month, clock_day, clock_hour,
                clock_min, clock_sec);//Set Time in RTC
1063
1064            return 0;
1065        }
1066        return -1;
1067    }
1068
1069    //*****************************************************************************
1070    //
1071    // Implementation of an command to set the minimum and maximum voltages
1072    //
1073    //*****************************************************************************
1074    int Cmd_set_limits (int argc, char *argv[]){
1075
1076        if(argc == 3){//takes only 2 parameters
1077            int min, max;
1078
1079            strcpy(g_cTmpInputData, argv[1]);
1080
1081            min = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
```

```
1082
1083            strcpy(g_cTmpInputData, argv[2]);
1084
1085            max = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1086
1087            if(min >= 0 && min < max && max <= 5000){//check borders
1088
1089                config.min_voltage = min * 1000;//µV
1090                config.max_voltage = max * 1000;//µV
1091                return 0;//success
1092            }
1093        }
1094        return -1;//Bad command
1095    }
1096
1097    //***************************************************************************
1098    //
1099    // Implementation of an command to set the absolute maximum current allowed
1100    //
1101    //***************************************************************************
1102    int Cmd_set_current(int argc, char *argv[]){
1103
1104        if(argc == 2){//check if only one command
1105
1106            strcpy(g_cTmpInputData, argv[1]);
1107            int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1108
1109            if(data > 0 && data <= 500000){//check if in borders
1110                config.max_current = data;
1111                return 0;//success
1112            }
1113        }
1114        return -1;//return bad command
1115    }
1116
1117    //***************************************************************************
1118    //
1119    // Implementation of an command to set the absolute maximum cycling time allowed
1120    //
1121    //***************************************************************************
1122    int Cmd_max_time (int argc, char *argv[]){
1123
1124        if(argc == 2){//check if only one command
1125
1126            strcpy(g_cTmpInputData, argv[1]);
1127            int data = (int) strtol(g_cTmpInputData, (char **)NULL, 10);
1128
1129            if(data > 0 && data < 99999){//check if in borders
1130                config.max_cycle_time = data;
1131                return 0;//success
1132            }
1133        }
1134        return -1;//return bad command
1135    }
1136
1137    //***************************************************************************
1138    //
1139    // Implementation of an command to set the absolute maximum temperature allowed
1140    //
```

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

```
1200            for(i=1 ; i<=20 ; i++){
1201
1202                sprintf(buf,"%d\n", adc_get_single_value());
1203                add_to_file("adc.txt", buf);
1204            }
1205            UARTprintf("\n%d LOW done",j);
1206            UARTprintf("\nnext");
1207
1208        }
1209        UARTprintf("\nALL LOW DONE!");
1210
1211        for(j = 1; j <= 12; j++){
1212
1213
1214                switch_relais(j);
1215                sprintf(buf,"\n CELL HIGH %d\n", j);
1216                add_to_file("adc.txt", buf);
1217
1218                for(i=1 ; i<=20 ; i++){
1219
1220                    sprintf(buf,"%d\n", adc_get_single_value());
1221                    add_to_file("adc.txt", buf);
1222                    UARTprintf(buf);
1223                }
1224                UARTprintf("\n%d HIGH done",j);
1225                UARTprintf("\nnext");
1226
1227            }
1228            UARTprintf("\nALL HIGH DONE!");
1229
1230    }
1231
1232
1233
1234
```

```
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_ */
38
```

```
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 zunÃ¤chst 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     tcp_recved(tpcb, p->tot_len);
126     pbuf_free(p);
127
128
129     return ERR_OK;
130 }
131
132 /*
133  * Auswertung der im FIFO enthaltenen Befehle.
134  * Es wird zunachst geprüft, ob eine vollstandige Zeile
135  * im FIFO vorhanden ist. Wenn ja, wird diese an die Funktion
136  * Cmd_interprete() übergeben. Wenn nicht, beendet
137  * sich die Funktion einfach.
138  */
139 void control_tick(unsigned char *data_check, unsigned long data_length)
140 {
141     long i;
142     tBoolean hadCommand = true;
143     char g_cEthGetBuf[64] = {0};
144
145
146     while(hadCommand)
147     {
148
149         for(i = 0; i < data_length; i++)
150         {
151             char element = *data_check++;
152
153             if(element == '\n' || element =='\r' )
154             {
155
156                 // There is a new command, execute
157
158                 Cmd_interprete(g_cEthGetBuf);
159
160                 hadCommand = false;
161
162                 break;
163             }
164
165             g_cEthGetBuf[i] = element;
166         }
167     }
168 }
169
170 void telnet_write(const void *data){
171     tcp_write(control_connection, data, strlen(data), TCP_WRITE_FLAG_COPY |
    TCP_WRITE_FLAG_MORE);
172 }
173
174
175 void telnet_immediate_write(const void *data){
176
```

```
177       if(tcp_write(control_connection, data, strlen(data),TCP_WRITE_FLAG_COPY |
          TCP_WRITE_FLAG_MORE) != ERR_OK){
178           tcp_output(control_connection);//Clear queue
179
180           while(tcp_write(control_connection, data, strlen(data),TCP_WRITE_FLAG_COPY |
              TCP_WRITE_FLAG_MORE)){//Retry sending till ERR_OK is returned
181               //tcp_output(control_connection);//Clear queue;#
182               //UARTprintf("\nRetry sending!");
183           }
184
185       }else;
186           tcp_output(control_connection);//Clear queue
187
188   }
189
190
```

```
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
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void discover_init(void);
17 void discover_rx(void* arg, struct udp_pcb* upcb, struct pbuf* p, struct ip_addr*
   addr, u16_t port);
18 void ipToString(char* string, unsigned long ip, int maxLen);
19
20 #endif /* DISCOVER_H_ */
21
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               discover.c
4
5   Auhtor:                             Fabian Schwartau
6   Credits:                            Thomas Wisniewski
7                                       Matthias Schneider
8                                       Tobias Steinmann
9                                       Stellaris Ware
10
11  last modified:                      2013/03/07
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:                        Discover the Board via Ethernet Broadcast message
24                                      by answer with IP-address-package
25  */
26
27
28
29
30  #include <string.h>
31  #include <stdio.h>
32  #include "utils/lwiplib.h"
33  #include "utils/uartstdio.h"
34  #include "inc/hw_ints.h"
35  #include "inc/hw_memmap.h"
36  #include "inc/hw_types.h"
37  #include "inc/hw_uart.h"
38  #include "inc/lm3s9b92.h"
39  #include "stdio.h"
40  #include "lwip/udp.h"
41
42
43
44  void discover_init(void);
45  void discover_rx(void* arg, struct udp_pcb* upcb, struct pbuf* p, struct ip_addr*
    addr, u16_t port);
46  void ipToString(char* string, unsigned long ip, int maxLen);
47
48  // UDP-Socket für die UDP-Nachrichten
49  struct udp_pcb* g_upcb;
50
51  /*
52   * Initialisierung des UDP-Sockets.
53   */
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) & 0xff);
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) & 0xff,
        (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 }
112
113
```

```
1   /*
2    * display.h
3    *
4    *  Created on: 25.03.2013
5    *      Author: Thomas W.
6    *              Johannes R.
7    *      modified: 13.01.2014
8    */
9
10  #ifndef DISPLAY_H_
11  #define DISPLAY_H_
12
13  #define LEDPORTCTRL      GPIO_PORTH_BASE
14  #define LEDPORTDATA      GPIO_PORTJ_BASE
15
16  #define RS               GPIO_PIN_0
17  #define RW               GPIO_PIN_1
18  #define E                GPIO_PIN_2
19
20
21  #define D4               GPIO_PIN_0
22  #define D5               GPIO_PIN_1
23  #define D6               GPIO_PIN_2
24  #define D7               GPIO_PIN_3
25
26
27  // Displaymodes
28  typedef enum
29  {
30      ERROR,
31      ERRCURR,
32      ERRCHARGE,
33      ERRVOLT,
34      ERRTIME,
35      ERRTEMP,
36      START,
37      CLOCK,
38      MEAS,
39      MEASSTOP,
40      CELLA,
41      CELLB,
42      CELLC,
43      CELLD,
44      CURRENT,
45      IP,
46      CONF,
47      CURRSTEP
48  } display_handler_t;
49
50  // Instance of display mode
51  volatile display_handler_t handlerState;
52
53  /*******************************************************************************
54  *
55  * Function Declarations
56  *
57  *******************************************************************************/
58
59  void display_init(void);
```

```
60  void display_reinit(void);
61  void buttons_init(void);
62  void clear_display(void);
63  void write_to_display(char* inputText, unsigned char row, unsigned char col);
64  void set_position(unsigned char address);
65  void write_char(unsigned char inputData);
66  void LEDcommand(unsigned char command);
67
68  void GPIOD01Handler(void);
69  void GPIOF00Handler(void);
70  void GPIOC04Handler(void);
71  void GPIOH05Handler(void);
72
73
74  #endif /* DISPLAY_H_ */
75
```

```
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                                      Stellaris Ware
11
12  last modified:                      2014/01/13
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 LED Display
25
26  */
27
28
29
30  /*****************************************************************************
31  *
32  * Own Includings
33  *
34  *****************************************************************************/
35
36  #include <utils/lwiplib.h>
37  #include "inc/hw_ints.h"
38  #include "inc/hw_memmap.h"
39  #include "inc/hw_types.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/sysctl.h"
46  #include "driverlib/uart.h"
47  #include "utils/uartstdio.h"
48  #include "driverlib/timer.h"
49  #include "header/display.h"
50  #include "stdio.h"
51  #include "header/config.h"
52  #include "header/mycmdline.h"
53
54
55
56  //*****************************************************************************
57  //Extern variables for Clock
58  //*****************************************************************************
59  extern volatile unsigned long clock_msec;
```

```
60  extern volatile unsigned long clock_sec;
61  extern volatile unsigned long clock_min;
62  extern volatile unsigned long clock_hour;
63  extern volatile unsigned long clock_day;
64  extern volatile unsigned long clock_month;
65  extern volatile unsigned long clock_year;
66
67  //*****************************************************************************
68  //
69  // Extern variables for measure data
70  //
71  //*****************************************************************************
72  extern volatile unsigned int g_iCellVoltages[];
73  extern volatile int g_iCellTemperature[];
74  extern volatile int g_iCurrent;
75  extern volatile int gBat_charge;
76
77  extern volatile unsigned long timer2minute;
78  extern volatile unsigned int cycle_active;
79  extern volatile unsigned int measurement_active;
80  extern volatile unsigned int current_cycle_step;
81  extern const char *cycle_types[];
82
83  void display_init(void)
84  {
85      UARTprintf("Initializing Display...");
86
87      //
88      // Enable the peripherals used by this example.
89      //
90      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
91      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
92
93      GPIOPinTypeGPIOOutput(LEDPORTCTRL, RS | RW | E ); // set RS | R/W | E as digital
        output
94      GPIOPinTypeGPIOOutput(LEDPORTDATA, D4 | D5 | D6 | D7); // set  D4 | D5 | D6 | D7
        as digital output
95
96      GPIOPadConfigSet(LEDPORTCTRL, RS | RW | E, GPIO_STRENGTH_8MA,
        GPIO_PIN_TYPE_STD_WPU);//set outputs to max 8 mA for better safety
97      GPIOPadConfigSet(LEDPORTDATA, D4 | D5 | D6 | D7, GPIO_STRENGTH_8MA,
        GPIO_PIN_TYPE_STD_WPU);
98
99
100     //*************************************************************
101     //Display Reset
102     GPIOPinWrite(LEDPORTCTRL, RS | RW | E, 0x0); // set RS | R/W | E to logic "0"
103     GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7 , 0x0); // set  D4 | D5 | D6 | D7 to
        logic "0"
104
105     // ca. 20 ms delay für Display reset
106     unsigned long ul_delay_count = 200000;
107     while (ul_delay_count) ul_delay_count--;
108     //--------------------------------
109
110
111     //
112     //Function set: 8 bit Interface = 0x30
113     //
```

```
114       GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
115       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
116
117       // ca. 0.45 ms delay warten Enable High
118       ul_delay_count = 4500;
119       while (ul_delay_count) ul_delay_count--;
120
121       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
122
123       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);        // Reset pins
124
125       // ca. 10 ms delay warten für nächsten Befehl
126       ul_delay_count = 100000;
127       while (ul_delay_count) ul_delay_count--;
128
129       //
130       //Function set: 8 bit Interf445ace = 0x30
131       //
132       GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
133       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
134
135       // ca. 0.45 ms delay warten Enable High
136       ul_delay_count = 4500;
137       while (ul_delay_count) ul_delay_count--;
138
139       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
140
141       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
142
143       // ca. 5 ms delay warten für nächsten Befehl
144       ul_delay_count = 50000;
145       while (ul_delay_count) ul_delay_count--;
146
147
148       //
149       //Function set: 8 bit Interface = 0x30
150       //
151       GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
152       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
153
154       // ca. 0.45 ms delay warten Enable High
155       ul_delay_count = 4500;
156       while (ul_delay_count) ul_delay_count--;
157
158       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
159
160       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
161
162       // ca. 2 ms delay warten für nächsten Befehl
163       ul_delay_count = 20000;
164       while (ul_delay_count) ul_delay_count--;
165
166
167
168       //
169       //Function set: 4 bit Interface = 0x20
170       //
171       GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
172       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x2);
```

```
173
174       // ca. 0.45 ms delay warten Enable High
175       ul_delay_count = 4500;
176       while (ul_delay_count) ul_delay_count--;
177
178       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
179       GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);        // Reset pins
180
181       // ca. 1 ms delay warten für nächsten Befehl
182       ul_delay_count = 10000;
183       while (ul_delay_count) ul_delay_count--;
184
185       clear_display();
186
187       //Function set: 4 bit Interface, 2(4) rows, 5x7dot = 0x28
188       LEDcommand(0x28);
189       //Display on, cursor off, blink off = 0x0C
190       LEDcommand(0x0C);
191       //Entry Mode set: Increment, No shift = 0x06
192       LEDcommand(0x06);
193
194
195       //
196       // Enable Timer 1
197       //
198       TimerEnable(TIMER1_BASE, TIMER_A);
199
200
201       UARTprintf("done\n");
202   }
203
204   void display_reinit(void){
205
206           //*********************************************************
207           //Display Reset
208           GPIOPinWrite(LEDPORTCTRL, RS | RW | E, 0x0); // set RS | R/W | E to logic "0"
209           GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7 , 0x0); // set  D4 | D5 | D6 |
              D7 to logic "0"
210
211           // ca. 20 ms delay für Display reset
212           unsigned long ul_delay_count = 200000;
213           while (ul_delay_count) ul_delay_count--;
214           //--------------------------------
215
216
217           //
218           //Function set: 8 bit Interface = 0x30
219           //
220           GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
221           GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
222
223           // ca. 0.45 ms delay warten Enable High
224           ul_delay_count = 4500;
225           while (ul_delay_count) ul_delay_count--;
226
227           GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
228
229           GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);        // Reset pins
230
```

```
231         // ca. 10 ms delay warten für nächsten Befehl
232         ul_delay_count = 100000;
233         while (ul_delay_count) ul_delay_count--;
234
235         //
236         //Function set: 8 bit Interface = 0x30
237         //
238         GPIOPinWrite(LEDPORTCTRL, E, E);               // Enable the Display
239         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
240
241         // ca. 0.45 ms delay warten Enable High
242         ul_delay_count = 4500;
243         while (ul_delay_count) ul_delay_count--;
244
245         GPIOPinWrite(LEDPORTCTRL, E, 0x00);            // Disable the Display
246
247         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
248
249         // ca. 5 ms delay warten für nächsten Befehl
250         ul_delay_count = 50000;
251         while (ul_delay_count) ul_delay_count--;
252
253
254         //
255         //Function set: 8 bit Interface = 0x30
256         //
257         GPIOPinWrite(LEDPORTCTRL, E, E);               // Enable the Display
258         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
259
260         // ca. 0.45 ms delay warten Enable High
261         ul_delay_count = 4500;
262         while (ul_delay_count) ul_delay_count--;
263
264         GPIOPinWrite(LEDPORTCTRL, E, 0x00);            // Disable the Display
265
266         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
267
268         // ca. 2 ms delay warten für nächsten Befehl
269         ul_delay_count = 20000;
270         while (ul_delay_count) ul_delay_count--;
271
272         //
273         //Function set: 4 bit Interface = 0x20
274         //
275         GPIOPinWrite(LEDPORTCTRL, E, E);               // Enable the Display
276         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x2);
277
278         // ca. 0.45 ms delay warten Enable High
279         ul_delay_count = 4500;
280         while (ul_delay_count) ul_delay_count--;
281
282         GPIOPinWrite(LEDPORTCTRL, E, 0x00);            // Disable the Display
283         GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);       // Reset pins
284
285         // ca. 1 ms delay warten für nächsten Befehl
286         ul_delay_count = 10000;
287         while (ul_delay_count) ul_delay_count--;
288
289         clear_display();
```

```
290
291         //Function set: 4 bit Interface, 2(4) rows, 5x7dot = 0x28
292         LEDcommand(0x28);
293         //Display on, cursor off, blink off = 0x0C
294         LEDcommand(0x0C);
295         //Entry Mode set: Increment, No shift = 0x06
296         LEDcommand(0x06);
297 }
298
299
300 void buttons_init(void)
301 {
302     UARTprintf("Initializing Buttons...");
303
304     //
305     // Enable the GPIO that is used for the first push button (GPIOF_PIN0).
306     //
307     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
308     ROM_GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
309     ROM_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA,
310                          GPIO_PIN_TYPE_STD_WPU);
311
312     //
313     // Enable the GPIO that is used for the second push button (GPIOD_PIN1).
314     //
315     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
316     ROM_GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_1);
317     ROM_GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_STRENGTH_2MA,
318                          GPIO_PIN_TYPE_STD_WPU);
319
320     //
321     // Enable the GPIO that is used for the third push button (GPIOC_PIN4).
322     //
323     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
324     ROM_GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_4);
325     ROM_GPIOPadConfigSet(GPIO_PORTC_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
326                          GPIO_PIN_TYPE_STD_WPU);
327
328
329     //
330     // Enable the GPIO that is used for the forth push button (GPIOH_PIN5).
331     //
332     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
333     ROM_GPIOPinTypeGPIOInput(GPIO_PORTH_BASE, GPIO_PIN_5);
334     ROM_GPIOPadConfigSet(GPIO_PORTH_BASE, GPIO_PIN_5, GPIO_STRENGTH_2MA,
335                          GPIO_PIN_TYPE_STD_WPU);
336
337
338     //
339     // Enable interrupts
340     //
341     ROM_IntEnable(INT_GPIOF);
342     GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_RISING_EDGE);  //trigger on
        rising edge
343     GPIOPinIntEnable(GPIO_PORTF_BASE,GPIO_PIN_0); //enable interrupt on PF0
344
345     ROM_IntEnable(INT_GPIOD);
346     GPIOIntTypeSet(GPIO_PORTD_BASE,GPIO_PIN_1,GPIO_RISING_EDGE);  //trigger on
        rising edge
```

```
347       GPIOPinIntEnable(GPIO_PORTD_BASE,GPIO_PIN_1); //enable interrupt on PD1
348
349       ROM_IntEnable(INT_GPIOC);
350       GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_4,GPIO_RISING_EDGE);  //trigger on
          rising edge
351       GPIOPinIntEnable(GPIO_PORTC_BASE,GPIO_PIN_4); //enable interrupt on PC4
352
353       ROM_IntEnable(INT_GPIOH);
354       GPIOIntTypeSet(GPIO_PORTH_BASE,GPIO_PIN_5,GPIO_RISING_EDGE);  //trigger on
          rising edge
355       GPIOPinIntEnable(GPIO_PORTH_BASE,GPIO_PIN_5); //enable interrupt on PC4
356
357       // Set statemachine to state Start
358       handlerState = START;
359       UARTprintf("done\n");
360   }
361
362   //*****************************************************************************
363   //
364   // This is the handler for the GPIO PORT F PIN 0 interrupt. (Button1)
365   //
366   //*****************************************************************************
367   void GPIOF00Handler(void){
368
369       //Disable Interrupts
370       TimerDisable(TIMER1_BASE, TIMER_A);
371       ROM_IntDisable(INT_GPIOF);
372       GPIOPinIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);
373
374       clear_display();
375
376           // Switch state
377           if(handlerState == CELLA)
378           handlerState = CELLD;
379
380           else if(handlerState == CELLD)
381           handlerState = CELLC;
382
383           else if(handlerState == CELLC)
384           handlerState = CELLB;
385
386           else if(handlerState == CELLB)
387           handlerState = CELLA;
388
389           else if(handlerState == MEAS)
390               Cmd_interprete("start");
391
392           else if(handlerState == MEASSTOP){
393               Cmd_interprete("stop");
394               handlerState = MEAS;
395           }
396
397       //Enable Interrupts
398       ROM_IntEnable(INT_GPIOF);
399       TimerEnable(TIMER1_BASE, TIMER_A);
400   }
401
402   //*****************************************************************************
403   //
```

```
404 // This is the handler for the GPIO PORT D PIN 1 interrupt. (Button2)
405 //
406 //*****************************************************************************
407 void GPIOD01Handler(void){
408
409     //Disable Interrupts
410     TimerDisable(TIMER1_BASE, TIMER_A);
411     GPIOPinIntClear(GPIO_PORTD_BASE,GPIO_PIN_1);
412
413
414     clear_display();
415
416         // Switch state
417         switch(handlerState){
418             case ERROR:   handlerState = START; break;
419             case ERRCURR: handlerState = START; break;
420             case ERRVOLT: handlerState = START; break;
421             case ERRTIME: handlerState = START; break;
422             case ERRTEMP: handlerState = START; break;
423             case ERRCHARGE:handlerState = START; break;
424             case START: handlerState = CLOCK; break;
425             case CLOCK: handlerState = MEAS; break;
426             case MEAS:
427             case MEASSTOP: if(cycle_active)
428                                 handlerState = CURRSTEP;
429                            else
430                                 handlerState = CELLA;
431                            break;
432             case CURRSTEP: handlerState = CELLA; break;
433             case CELLA: handlerState = CURRENT;
434             case CELLB: handlerState = CURRENT;
435             case CELLC: handlerState = CURRENT;
436             case CELLD: handlerState = CURRENT; break;
437             case CURRENT: handlerState = IP; break;
438             case IP: handlerState = CONF; break;
439             case CONF: handlerState = START; break;
440
441         }
442
443     //Enable Interrupts
444     TimerEnable(TIMER1_BASE, TIMER_A);
445 }
446
447
448 //*****************************************************************************
449 //
450 // This is the handler for the GPIO PORT C PIN 4 interrupt. (Button3)
451 //
452 //*****************************************************************************
453 void GPIOC04Handler(void){
454     //Disable Interrupts
455     TimerDisable(TIMER1_BASE, TIMER_A);
456     ROM_IntDisable(INT_GPIOC);
457     GPIOPinIntClear(GPIO_PORTC_BASE,GPIO_PIN_4);
458
459     clear_display();
460
461         // Switch state
462         if(handlerState == CELLA)
```

```
463           handlerState = CELLB;
464
465           else if(handlerState == CELLB)
466           handlerState = CELLC;
467
468           else if(handlerState == CELLC)
469           handlerState = CELLD;
470
471           else if(handlerState == CELLD)
472           handlerState = CELLA;
473
474           else if(handlerState == MEAS)
475           handlerState = MEASSTOP;
476
477           else if(handlerState == MEASSTOP)
478           handlerState = MEAS;
479
480       //Enable Interrupts
481       ROM_IntEnable(INT_GPIOC);
482       TimerEnable(TIMER1_BASE, TIMER_A);
483   }
484
485   //*****************************************************************************
486   //
487   // This is the handler for the GPIO PORT H PIN 5 interrupt. (Button4)
488   // NO FUNCTION YET!
489   //
490   //*****************************************************************************
491   void GPIOH05Handler(void){
492       //Disable Interrupts
493       TimerDisable(TIMER1_BASE, TIMER_A);
494       ROM_IntDisable(INT_GPIOH);
495       GPIOPinIntClear(GPIO_PORTH_BASE,GPIO_PIN_5);
496
497       clear_display();
498
499       //Enable Interrupts
500       ROM_IntEnable(INT_GPIOH);
501       TimerEnable(TIMER1_BASE, TIMER_A);
502   }
503
504
505   //*****************************************************************************
506   //
507   // The  handler for the Timer1A interrupt.
508   // Refresh the Content of the LED-Display dependen on current state
509   //
510   //*****************************************************************************
511   void Timer1IntHandler(void){
512
513       unsigned long ulIPAddress;
514       char display_buffer[21];
515       int bat_voltage = 0;
516       int i = 0;
517
518
519       //Write content to Displays
520       switch(handlerState) {
521
```

```
522         case ERROR:
523
524             write_to_display("ERROR in config.txt!", 0, 0);
525             write_to_display("   default values   ", 1, 0);
526             write_to_display("       loaded       ", 2, 0);
527             write_to_display("***** CONTINUE *****", 3 ,0);
528             break;
529
530         case ERRCHARGE:
531
532             write_to_display(" Cycling canceled!  ", 0, 0);
533             write_to_display("  maximum charge    ", 1, 0);
534             write_to_display("      exceeded      ", 2, 0);
535             write_to_display("***** CONTINUE *****", 3 ,0);
536             break;
537
538         case ERRCURR:
539
540             write_to_display(" Cycling canceled!  ", 0, 0);
541             write_to_display("  maximum current   ", 1, 0);
542             write_to_display("      exceeded      ", 2, 0);
543             write_to_display("***** CONTINUE *****", 3 ,0);
544             break;
545
546         case ERRVOLT:
547
548             write_to_display(" Cycling canceled!  ", 0, 0);
549             write_to_display("   voltage limits   ", 1, 0);
550             write_to_display("      exceeded      ", 2, 0);
551             write_to_display("***** CONTINUE *****", 3 ,0);
552             break;
553
554         case ERRTIME:
555
556             write_to_display(" Cycling finished!  ", 0, 0);
557             write_to_display(" maximum cycle time ", 1, 0);
558             write_to_display("      reached       ", 2, 0);
559             write_to_display("***** CONTINUE *****", 3 ,0);
560             break;
561
562         case ERRTEMP:
563
564             write_to_display(" Cycling canceled!  ", 0, 0);
565             write_to_display("maximum temperature ", 1, 0);
566             write_to_display("      exceeded      ", 2, 0);
567             write_to_display("***** CONTINUE *****", 3 ,0);
568             break;
569
570         case START:
571
572             write_to_display("********************", 0, 0);
573             write_to_display("***    Battery   ***", 1, 0);
574             write_to_display("**  Cycle Machine **", 2, 0);
575             write_to_display("*****   MENU   *****", 3 ,0);
576             break;
577
578         case CLOCK:
579
580             sprintf(display_buffer, "      %02d:%02d:%02d      " , clock_hour,
```

```
                clock_min, clock_sec);
581             write_to_display(display_buffer, 0, 0);
582             sprintf(display_buffer, "     %02d.%02d.%04d     " , clock_day,
                clock_month, clock_year);
583             write_to_display(display_buffer, 1, 0);
584             write_to_display("        MENU        ", 3 ,0);
585             break;
586
587         case MEAS:
588
589             sprintf(display_buffer, "Measurement Cycling:");
590             write_to_display(display_buffer, 0, 0);
591
592             if(measurement_active && cycle_active)
593                 sprintf(display_buffer, "active      active  ");
594             else if(measurement_active && !cycle_active)
595                 sprintf(display_buffer, "active      not act.");
596             else
597                 sprintf(display_buffer, "not act.    not act.");
598             write_to_display(display_buffer, 1, 0);
599
600
601             write_to_display("START   MENU    STOP", 3 ,0);
602             break;
603
604         case MEASSTOP:
605
606             sprintf(display_buffer, "Measurement Cycling:");
607             write_to_display(display_buffer, 0, 0);
608
609             sprintf(display_buffer, "    Confirm STOP    ");
610
611             write_to_display(display_buffer, 1, 0);
612
613
614             write_to_display(" YES    MENU     NO ", 3 ,0);
615             break;
616
617         case CELLA:
618
619             for(i = 0; i <= 2; i++){
620                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
                    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
621                                                                          ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
622                 write_to_display(display_buffer, i, 0);
623             }
624
625             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
626             break;
627
628         case CELLB:
629
630             for(i = 3; i <= 5; i++){
631                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
                    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
632                                                                          ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
633                 write_to_display(display_buffer, i-3, 0);
```

```
634             }
635
636             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
637             break;
638
639         case CELLC:
640
641             for(i = 6; i <= 8; i++){
642                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
                    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
643                                                                          ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
644                 write_to_display(display_buffer, i-6, 0);
645             }
646
647             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
648             break;
649
650         case CELLD:
651
652             for(i = 9; i <= 11; 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, g_iCellTemperature[i]%10);
655                 write_to_display(display_buffer, i-9, 0);
656             }
657
658             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
659             break;
660
661         case CURRENT:{
662
663             bat_voltage = 0;
664             for(i = 0; i < config.quantity_cells; i++)//calculate battery voltage
665                 bat_voltage += g_iCellVoltages[i];
666             bat_voltage /= 1000;//mV
667
668             sprintf(display_buffer, "Voltage:   %02d.%03d V ",bat_voltage/1000,
                bat_voltage%1000);
669             write_to_display(display_buffer, 0, 0);
670
671             if(g_iCurrent < 0)
672                 sprintf(display_buffer, "Current: -%03d.%03d A ", abs(g_iCurrent/1000
                    ), abs(g_iCurrent%1000));
673             else
674                 sprintf(display_buffer, "Current:  %03d.%03d A ", g_iCurrent/1000,
                    g_iCurrent%1000);
675             write_to_display(display_buffer, 1, 0);
676
677             if(gBat_charge < 0)
678                 sprintf(display_buffer, "Charge : -%03d.%03d Ah", abs(gBat_charge/
                    3600000), abs((gBat_charge/3600)%1000));
679             else
680                 sprintf(display_buffer, "Charge :  %03d.%03d Ah", gBat_charge/3600000
                    , (gBat_charge/3600)%1000);
681             write_to_display(display_buffer, 2, 0);
682
683             write_to_display("        MENU        ", 3 ,0);
```

```
684             break;
685         }
686         case IP:
687 
688             sprintf(display_buffer, "     IP-Address:    ");
689             write_to_display(display_buffer, 0, 0);
690             ulIPAddress = lwIPLocalIPAddrGet();
691             sprintf(display_buffer, "   %03d.%03d.%03d.%03d  "  , ulIPAddress & 0xff,
692                     (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
693                     (ulIPAddress >> 24) & 0xff);
694             write_to_display(display_buffer, 1, 0);
695             write_to_display("        MENU        ", 3 ,0);
696             break;
697 
698         case CONF:
699 
700             sprintf(display_buffer, "Quantity cells:%02d   ", config.quantity_cells);
701             write_to_display(display_buffer, 0, 0);
702             sprintf(display_buffer, "Cyclestep(sec):%04d "  , config.cyclestep);
703             write_to_display(display_buffer, 1, 0);
704             if(config.measuretime > 0)
705                 sprintf(display_buffer, "Time left(min):%05d"  , (config.measuretime
                    - timer2minute));
706             else
707                 sprintf(display_buffer, "Time left(min):  inf");
708             write_to_display(display_buffer, 2, 0);
709             write_to_display("        MENU        ", 3 ,0);
710             break;
711 
712         case CURRSTEP:
713 
714             sprintf(display_buffer, "Current step %d/%d", current_cycle_step, config.
                step_quantity);
715             write_to_display(display_buffer, 0, 0);
716             sprintf(display_buffer, "type %s"  , cycle_types[config.cycle_steps[
                current_cycle_step].type - 1]);
717             write_to_display(display_buffer, 1, 0);
718             if(config.cycle_steps[current_cycle_step].type == 1)
719                 sprintf(display_buffer, "%d min relay %d"  ,config.cycle_steps[
                    current_cycle_step].param ,config.cycle_steps[current_cycle_step].
                    relay);
720             else
721                 sprintf(display_buffer, "%d mV relay %d"  ,config.cycle_steps[
                    current_cycle_step].param ,config.cycle_steps[current_cycle_step].
                    relay);
722             write_to_display(display_buffer, 2, 0);
723             write_to_display("        MENU        ", 3 ,0);
724             break;
725     }
726 
727     //
728     // Clear the timer interrupt.
729     //
730    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
731 }
732 
733 
734 //*****************************************************************************
735 //
```

```
736 // Clear the LED-Display
737 //
738 //*****************************************************************************
739 void clear_display(void){
740
741
742     LEDcommand(0x01);
743
744
745 }
746
747 //*****************************************************************************
748 //
749 // Write string to Display by row and column
750 //
751 //*****************************************************************************
752 void write_to_display(char* inputText, unsigned char row, unsigned char col) {
753     unsigned char address_d = 0;        // address of the data in the screen.
754
755     //define address
756     switch(row)
757         {
758             case 0: address_d = 0x80 + col;     // at zeroth row
759             break;
760             case 1: address_d = 0xC0 + col;     // at first row
761             break;
762             case 2: address_d = 0x94 + col;     // at second row
763             break;
764             case 3: address_d = 0xD4 + col;     // at third row
765             break;
766             default: address_d = 0x80 + col;    // returns to first row if invalid
                row number is detected
767             break;
768         }
769
770     //set position by address
771     set_position(address_d);
772
773     // Place a string, letter by letter.
774     while(*inputText)
775         write_char(*inputText++);
776 }
777
778 //*****************************************************************************
779 //
780 // Set the position by address
781 //
782 //*****************************************************************************
783 void set_position(unsigned char address) {
784
785     unsigned long ul_delay_count;
786
787     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
788     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0xf0) >> 4);//data
789
790     // ca. 1 µs delay warten Enable High
791     ul_delay_count = 4500;
792     while (ul_delay_count) ul_delay_count--;
793
```

```
794       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
795       //10ns hold time
796       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
797
798
799       GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
800       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0x0f));//data
801
802       // ca. 1 µs delay warten Enable High
803       ul_delay_count = 4500;
804       while (ul_delay_count) ul_delay_count--;
805
806       GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
807       //10ns hold time
808       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
809
810   }
811
812   //*****************************************************************************
813   //
814   // Write character to Display
815   //
816   //*****************************************************************************
817   void write_char(unsigned char inputData) {
818
819       unsigned long ul_delay_count;
820
821       GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);                // Enable the Display
          and writeoperation
822       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0xf0) >> 4);//data
823
824       // ca. 1 µs delay warten Enable High
825       ul_delay_count = 4500;
826       while (ul_delay_count) ul_delay_count--;
827
828       GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);            // Disable the Display
829       //10 ns hold time
830       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
831
832
833       GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);                // Enable the Display
          and writeoperation
834       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0x0f));//data
835
836       // ca. 1 µs delay warten Enable High
837       ul_delay_count = 4500;
838       while (ul_delay_count) ul_delay_count--;
839
840       GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);            // Disable the Display
841       //10ns hold time
842       GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
843
844   }
845
846   //*****************************************************************************
847   //
848   // Send Command to Display
849   //
850   //*****************************************************************************
```

```
851  void LEDcommand(unsigned char command){
852
853      unsigned long ul_delay_count;
854
855      GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
856      GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0xf0) >> 4);//data
857
858      // ca. 1 µs delay warten Enable High
859      ul_delay_count = 4500;
860      while (ul_delay_count) ul_delay_count--;
861
862      GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
863      //10ns hold time
864      GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0);  // Reset pins
865
866      // ca. 100 µs delay warten auf nächsten Befehl
867      ul_delay_count = 1000;
868      while (ul_delay_count) ul_delay_count--;
869
870
871      GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
872      GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0x0f));//data
873
874      // ca. 1 µs delay warten Enable High
875      ul_delay_count = 4500;
876      while (ul_delay_count) ul_delay_count--;
877
878      GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
879      //10ns hold timne
880      GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0);  // Reset pins
881
882      // ca. 1 ms delay warten auf  nächsten Befehl
883      ul_delay_count = 10000;
884      while (ul_delay_count) ul_delay_count--;
885  }
886
887
888
```

```
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_ */
23
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               ethernet.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:                      2014/01/13
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:                        Initialize Ethernet with lwip. Define IP-Address,
25                                      call initialization of TCP and UDP.
26  */
27
28
29  /*****************************************************************************
30  *
31  * Includings
32  *
33  *****************************************************************************/
34  #include <utils/lwiplib.h>
35  #include <utils/uartstdio.h>
36  #include <utils/locator.h>
37  #include <inc/hw_types.h>
38  #include <inc/hw_ints.h>
39  #include <inc/hw_memmap.h>
40  #include <driverlib/rom.h>
41  #include <driverlib/sysctl.h>
42  #include <driverlib/gpio.h>
43  #include <driverlib/interrupt.h>
44  #include "driverlib/systick.h"
45  #include <lwip/ip_addr.h>
46  #include "stdio.h"
47  #include <inc/lm3s9b92.h>
48
49  /*****************************************************************************
50  *
51  * Own Includings
52  *
53  *****************************************************************************/
54  #include "header/control.h"
55  #include "header/discover.h"
56  #include "header/config.h"
57  #include "header/ethernet.h"
58
59  int ethernet_init(void);
```

```
60  void lwIPHostTimerHandler(void);
61
62  // Zur Anzeige einer Aktivität in der Konsole
63  static char g_pcTwirl[4] = { '\\', '|', '/', '-' };
64  static unsigned long g_ulTwirlPos = 0;
65
66  // TCP-Socket für die aktuelle Steuerungs-Verbindung
67  extern struct tcp_pcb* control_connection;
68
69  // Aktuelle IP Adresse
70  static unsigned long g_ulLastIPAddr = 0;
71
72  unsigned long ulUser0, ulUser1;
73
74  //MAC Adresse aus define auslesen DEBUG
75  unsigned long long macAddress = MACADDR;
76
77
78  // MAC Adresse für weitere Verwendung im Programm
79  unsigned char pucMACArray[8];
80
81
82  /*
83   * Initialize Ethernetfunction
84   */
85  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     }
225 
226 
```

```
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_ */
21
```

```
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>
27  #include "inc/hw_memmap.h"
28  #include "inc/hw_types.h"
29  #include "utils/uartstdio.h"
30  #include "utils/ustdlib.h"
31  #include "inc/hw_ints.h"
32  #include "inc/hw_types.h"
33  #include "inc/hw_uart.h"
34  #include "inc/hw_gpio.h"
35
36  #include "driverlib/uart.h"
37  #include <driverlib/gpio.h>
38
39
40  // Initialize GPIO-Ports for LED in Frontpanel
41  void led_init(void){
42      UARTprintf("Initializing Front LEDs...");
43
44      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);   // enable peripheral
45      GPIOPinTypeGPIOOutput(GPIO_PORTJ_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
        GPIO_PIN_7 ); // set PJ4 to PJ7 as digital output
46      GPIOPadConfigSet(GPIO_PORTJ_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
        GPIO_PIN_7, GPIO_STRENGTH_8MA,
47              GPIO_PIN_TYPE_STD_WPD);
48
49      UARTprintf("done\n");
50  }
51
```

```
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+PBU
                                                        F_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__ */
439
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               main.c
4
5   Auhtor:                             Thomas Wisnewski
6   Credits:                            Matthias Schneider
7                                       Tobias Steinmann
8                                       Fabian Schwartau
9                                       Johannes Roehn
10                                      Stellaris Ware
11
12  last modified:                      2014/01/08
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:                        Main File for BATSEN Battery Cycling Machine
25  */
26
27
28
29  //*****************************************************************************
30  *
31  * Includings
32  *
33  *****************************************************************************/
34  #include "inc/hw_ints.h"
35  #include "inc/hw_memmap.h"
36  #include "inc/hw_types.h"
37  #include "driverlib/debug.h"
38  #include "driverlib/gpio.h"
39  #include "driverlib/interrupt.h"
40  #include "driverlib/pin_map.h"
41  #include "driverlib/rom.h"
42  #include "driverlib/sysctl.h"
43  #include "driverlib/uart.h"
44  #include "string.h"
45  #include <stdio.h>
46  #include <stdlib.h>
47
48
49
50  //*****************************************************************************
51  *
52  * Own Includings
53  *
54  *****************************************************************************/
55  #include "utils/uartstdio.h"
56  #include "driverlib/rom_map.h"
57  #include "header/myuart.h"
58  #include "header/mysdcard.h"
59  #include "header/myssi.h"
```

```
60  #include "header/myadc.h"
61  #include "header/relais.h"
62  #include "header/config.h"
63  #include "header/mycmdline.h"
64  #include "header/clocktimer.h"
65  #include "header/temperature.h"
66  #include "header/ethernet.h"
67  #include "header/control.h"
68  #include "header/mypwm.h"
69  #include "header/display.h"
70  #include "header/rtc.h"
71  #include "header/led.h"
72  #include "header/watchdog.h"
73  #include "driverlib/timer.h"
74  #include "utils/lwiplib.h"
75  #include "third_party/fatfs/src/ff.h"
76  #include "third_party/fatfs/src/diskio.h"
77
78  /*****************************************************************************
79  *
80  * Globals
81  *
82  *****************************************************************************/
83
84
85  //Variables for the cat command
86  int g_cat_argc;
87  char g_cat_argv[20], g_cat_flag = 0;
88
89  //Flag for Display ReINIT
90  volatile int g_bDispInit = 0;
91
92  // String buffer for print operations to UART and Ethernet
93  extern char print_buffer[];
94  // TCP-Socket für die aktuelle Steuerungs-Verbindung
95  extern struct tcp_pcb* control_connection;
96
97  extern char g_cAnsBuf[];
98
99  extern volatile tBoolean g_bFeedWatchdog;
100
101 // Instanz der Konfiguration
102 config_t config;
103
104 // Define global location buffer
105 char *g_cLocalBuf;
106
107 // Define Command line pointer
108 tCmdLineEntry *g_psCmdTable;
109
110 // Define string for main location
111 const char *g_cMainLocalBuf = "Main";
112
113 // define command set for main program
114 tCmdLineEntry g_sMainCmdTable[] =
115 {
116     { "help",   Cmd_help,           "   : Display list of commands" },
117     { "h",      Cmd_help,         "      : alias for help" },
118     { "?",      Cmd_help,         "      : alias for help" },
```

```
119       { "browser",my_start_cmd_line,      ": SD-Card browser" },
120       { "config", config_cmd_line,       " : Configuration" },
121       { "alive",  Cmd_alive,            "  : Check if Cycling is active" },
122       { "start",  Cmd_start,            "  : Start measurement cycling" },
123       { "stop",   Cmd_stop,            "   : Stop measurement cycling" },
124       { 0, 0, 0 }
125   };
126
127
128   void main(void) {
129
130       // Set the system clock to run at 50MHz from the PLL.
131       //
132       ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
133                          SYSCTL_XTAL_16MHZ);
134
135
136       //
137       // Enable processor interrupts.
138       //
139       IntMasterEnable();
140
141       // set Command line pointer to the beginning of the main command structure
142       g_psCmdTable = &g_sMainCmdTable[0];
143
144       // set location buffer to main
145       g_cLocalBuf = (char*)g_cMainLocalBuf;
146
147
148       //initializing program
149       myUARTinit();
150       buttons_init();
151       ssi1_init();
152       mysdcardinit();
153       init_config();
154       myadc_init();
155       init_ethernet();
156       led_init();
157       init_relais();
158       init_clock_timer();
159       rtc_init();
160       init_temperature();
161       pwm_init();
162       display_init();
163       watchdog_init();
164
165       //DEBUG INIT TRIGGER OUTPUT
166       //GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_5);
167       //GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, 0);
168
169       //*********************************************************
170       //Set Priority of interrupts
171       //*********************************************************
172
173       //Systick 100Hz Timer for FATFs, Clock and Ethernet + Watchdog
174       ROM_IntPrioritySet(FAULT_SYSTICK,   0x00);  //1st priority
175       ROM_IntPrioritySet(INT_WATCHDOG,    0x00);  //1st priority
176
177       //UART
```

```
178       ROM_IntPrioritySet(INT_UART0,       0x20);  //2nd priority
179       //Ethernet interrupt
180       ROM_IntPrioritySet(INT_ETH,         0x20);  //2nd priority
181
182       //Cycle Control Timer
183       ROM_IntPrioritySet(INT_TIMER2A,     0x20);  //2nd priority (SAFETY Reasons)
184       //Cycle LED periodic toggle Timer
185       ROM_IntPrioritySet(INT_TIMER3A,     0x20);  //2nd priority
186
187       //Cycle Measurement Timer
188       ROM_IntPrioritySet(INT_TIMER0A,     0x40);  //3rd priority
189       //ADC Temperature Interrupts
190       ROM_IntPrioritySet(INT_ADC0SS0,     0x40);  //3rd priority
191       ROM_IntPrioritySet(INT_ADC1SS1,     0x40);  //3rd priority
192
193
194       //Display timer
195       ROM_IntPrioritySet(INT_TIMER1A,     0x60);  //4th priority
196       //Display buttons interrupts
197       ROM_IntPrioritySet(INT_GPIOC,       0x80);  //5th priority
198       ROM_IntPrioritySet(INT_GPIOD,       0x80);  //5th priority
199       ROM_IntPrioritySet(INT_GPIOF,       0x80);  //5th priority
200
201       ROM_IntPrioritySet(INT_SYSCTL,      0xE0);  //lowest priority
202
203       //*********************************************************
204
205
206       //
207       // Print a prompt to the console.
208       //
209       UARTprintf("\n\n\n");
210       UARTprintf("*********************************************\n");
211       UARTprintf("***        Battery Cycle Machine          ****\n");
212       UARTprintf("*********************************************\n");
213
214       UARTprintf("\n Enter Command (type <help> to see list of commands):");
215       UARTprintf("\n%s: > ", g_cLocalBuf);
216
217       //calibrate_adc();
218
219       //
220       // Loop forever
221       //
222       while(1){
223
224           //WATCHDOG FEED
225           g_bFeedWatchdog = true;//if not fed in the main loop watchdog will reset the
              system
226
227           //Check flag for the execution of the cat command
228           if(g_cat_flag){
229               FRESULT fresult;
230               unsigned short usBytesRead;
231
232               //
233               // Disable Timer 1 for Display INT
234               //
235               TimerDisable(TIMER1_BASE, TIMER_A);
```

```
236
237             //write message to display being disabled
238             write_to_display("     Display not    ", 0, 0);
239             write_to_display("  available during  ", 1, 0);
240             write_to_display("network transmission", 2, 0);
241             write_to_display("********************", 3 ,0);
242
243
244             //
245             // First, check to make sure that the current path (CWD), plus
246             // the file name, plus a separator and trailing null, will all
247             // fit in the temporary buffer that will be used to hold the
248             // file name.  The file name must be fully specified, with path,
249             // to FatFs.
250             //
251             if(strlen(g_cCwdBuf) + strlen(g_cat_argv) + 1 + 1 > sizeof(g_cTmpBuf))
252             {
253                 sprintf(print_buffer, "Resulting path name is too long\n");
254
255                 UARTprintf(print_buffer);
256
257                 if(control_connection){
258                     telnet_write(print_buffer);
259                 }
260                 //return(0);
261             } else {
262
263                 //
264                 // Copy the current path to the temporary buffer so it can be
                    manipulated.
265                 //
266                 strcpy(g_cTmpBuf, g_cCwdBuf);
267
268                 //
269                 // If not already at the root level, then append a separator.
270                 //
271                 if(strcmp("/", g_cCwdBuf))
272                 {
273                     strcat(g_cTmpBuf, "/");
274                 }
275
276                 //
277                 // Now finally, append the file name to result in a fully specified
                    file.
278                 //
279                 strcat(g_cTmpBuf, g_cat_argv);
280
281                 //
282                 // Open the file for reading.
283                 //
284                 fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
285
286                 //
287                 // If there was some problem opening the file, then return
288                 // an error.
289                 //
290                 if(fresult != FR_OK)
291                 {
292                     //return(fresult);
```

```
293                 } else {
294
295                     //If Ethernet connected no output via UART
296                     if(control_connection)
297                         UARTprintf("\nOutput only via connected Ethernet!\n");
298                     //
299                     // Enter a loop to repeatedly read data from the file and
                        display it,
300                     // until the end of the file is reached.
301                     //
302                     do{
303
304                         //WATCHDOG FEED
305                         g_bFeedWatchdog = true;
306
307                         //
308                         // Read a block of data from the file.  Read as much as can
                            fit
309                         // in the temporary buffer, including a space for the
                            trailing null.
310                         //
311                         fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf)
                             - 1,
312                                          &usBytesRead);
313
314                         //
315                         // If there was an error reading, then print a newline and
316                         // return the error to the user.
317                         //
318                         if(fresult != FR_OK)
319                         {
320                             sprintf(print_buffer, "\n");
321                             UARTprintf(print_buffer);
322                             if(control_connection){
323                                 telnet_write(print_buffer);
324                             }
325                             //return(fresult);
326                             break;
327                         }
328
329                         //
330                         // Null terminate the last block that was read to make it a
331                         // null terminated string that can be used with printf.
332                         //
333                         g_cTmpBuf[usBytesRead] = 0;
334
335                         //
336                         // Print the last chunk of the file that was received.
337                         //
338                         sprintf(print_buffer, "%s", g_cTmpBuf);
339
340                         //Send data over Etherenet OR UART due to performance
                            optimization
341                         //Output over both interfaces slows down the Ethernet
                            connection
342                         if(control_connection){
343                             telnet_immediate_write(print_buffer);
344                         } else
345                             UARTprintf(print_buffer);
```

```
346
347                         //
348                         // Continue reading until less than the full number of bytes
                            are
349                         // read.  That means the end of the buffer was reached.
350                         //
351
352                     } while(usBytesRead == sizeof(g_cTmpBuf) - 1);
353
354                     if(control_connection){
355                         telnet_immediate_write("\nEnd of file\n Enter Command (type
                            <help> to see list of commands):\n");
356                         sprintf(g_cAnsBuf, "%s: > \n", g_cLocalBuf);
357                         telnet_immediate_write(g_cAnsBuf);
358                     }
359
360                     //
361                     // Print a prompt to the console.
362                     //
363                     UARTprintf("\nEnd of file");
364                     UARTprintf("\n Enter Command (type <help> to see list of
                            commands):");
365                     UARTprintf("\n%s: > ", g_cLocalBuf);
366
367                 }
368             }
369              //Delete Flag
370              g_cat_flag = 0;
371
372              //
373              // Enable Timer 1 for Display INT
374              //
375             TimerEnable(TIMER1_BASE, TIMER_A);
376         }
377
378         //Re INIT Display every minute
379         if(g_bDispInit){
380             g_bDispInit = 0;
381             TimerDisable(TIMER1_BASE, TIMER_A);
382             display_reinit();
383             TimerEnable(TIMER1_BASE, TIMER_A);
384         }
385     }
386 }
387
388
389
```

```
1  /*
2   * myadc.h
3   *
4   *  Created on: 11.02.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef MAYADC_H_
9  #define MAYADC_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 int myadc_init(void);
17 unsigned char adc_status(void);
18 unsigned short adc_read_config_reg(void);
19 unsigned char adc_write_read(unsigned char);
20 void adc_clear_fifo(void);
21 void adc_set_config_reg(unsigned short);
22 unsigned short adc_read_fs_reg(void);
23 unsigned short adc_read_offset_reg(void);
24 void adc_set_config_reg(unsigned short);
25 void adc_set_mode_reg(unsigned short);
26 void adc_set_fs_reg(unsigned short);
27 void adc_set_offset_reg(unsigned short);
28 unsigned short adc_read_data(void);
29
30
31 unsigned short adc_get_single_value(void);
32 unsigned short adc_get_single_value_ch2(void);
33 unsigned short adc_get_single_value_ch3(void);
34 int adc_get_current(void);
35 int adc_get_voltage(unsigned int);
36
37 #endif /* MAYADC_H_ */
38
```

```
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                                      Stellaris Ware
11
12  last modified:                      2014/01/08
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 AD7798 usage
25
26  */
27
28
29
30
31  /*****************************************************************************
32  *
33  * Includings
34  *
35  *****************************************************************************/
36  #include "driverlib/uart.h"
37  #include "utils/uartstdio.h"
38  #include "driverlib/rom_map.h"
39  #include "inc/hw_ints.h"
40  #include "inc/hw_memmap.h"
41  #include "driverlib/rom.h"
42  #include "driverlib/gpio.h"
43  #include "driverlib/pin_map.h"
44  #include "driverlib/sysctl.h"
45  #include "driverlib/ssi.h"
46  #include "inc/hw_ssi.h"
47
48  /*****************************************************************************
49  *
50  * Own Includings
51  *
52  *****************************************************************************/
53  #include "header/myadc.h"
54  #include "header/config.h"
55  #include "header/relais.h"
56  #include "header/myssi.h"
57
58  //*****************************************************************************
59  //
```

```
60  // A global data buffer used for actual Cell voltage
61  //
62  //*****************************************************************************
63  volatile unsigned int g_iCellVoltages[12] = {0};
64  volatile unsigned int g_iCurrent = 0;
65
66  // AD7798 initialization and configuration
67  int myadc_init(void)
68  {
69
70      UARTprintf("Initialization of external ADC...");
71      unsigned long ul_delay_count;
72
73      // SSI1 Fifo löschen
74      adc_clear_fifo();
75
76      // ADC reset
77      spiChipSelect(ADC);
78      SSIDataPut(SSI1_BASE, 0xFF);
79      SSIDataPut(SSI1_BASE, 0xFF);
80      SSIDataPut(SSI1_BASE, 0xFF);
81      SSIDataPut(SSI1_BASE, 0xFF);
82      spiChipSelect(NONE);
83
84      // ca. 50 ms delay für ADC reset
85      ul_delay_count = 1000000;
86      while (ul_delay_count) ul_delay_count--;
87      //--------------------------------
88
89      // Setzen des "configuration register"
90      // unipolar mode, gain = 1, buffered, noref = 0x1010
91      //adc_set_config_reg(0x1010);
92      // bipolar mode, gain = 1, buffered, noref = 0x0010
93      //adc_set_config_reg(0x0010);
94      // bipolar mode, gain = 1, unbuffered, noref = 0x0000
95      //adc_set_config_reg(0x0000);
96      // unipolar mode, gain = 1, unbuffered, noref = 0x1000
97      adc_set_config_reg(0x1000);
98
99       // In den "power down" Modus gehen, um das "mode register" zu ändern
100     adc_set_mode_reg(0x600A);
101     adc_set_offset_reg(0);  // Initialisierung des "offset registers" mit 0
102     adc_set_fs_reg(0); // Initialisierung des "full scale registers" mit 0
103
104     // Prüfen, ob die Werte korrekt geschrieben wurden
105     if(adc_read_offset_reg() != 0)
106     {
107         UARTprintf("failed: offset calibration reset failed!\n");
108         return 0;
109     }
110     if(adc_read_fs_reg() != 0)
111     {
112         UARTprintf("failed: full scale calibration reset failed!\n");
113         return 0;
114     }
115
116     // Durchführung der internen Kalibrierung des ADCs
117     adc_set_mode_reg(0x800A); // Internal zero offset calibration
118     while((adc_status() & (1<<7)));
```

```
119       adc_set_mode_reg(0xA00A); // Internal full scale calibration
120       while((adc_status() & (1<<7)));
121
122       // Prüfen, ob die Kalibierung erfolgreich war
123       // (sich die Werte geändert haben)
124       if(adc_read_offset_reg() == 0)
125       {
126           UARTprintf("failed: offset calibration failed!\n");
127           return 0;
128       }
129       if(adc_read_fs_reg() == 0)
130       {
131           UARTprintf("failed: full scale calibration failed!\n");
132           return 0;
133       }
134
135       //Calibrate HALL-Sensor to 0
136       config.zerocurrentch1 = 0;
137       config.zerocurrentch2 = 0;
138       config.zerocurrentch1 = (32770 - adc_get_single_value_ch2());
139       config.zerocurrentch2 = (32770 - adc_get_single_value_ch3());
140
141       UARTprintf("done.\n");
142       return 1;
143
144
145   }
146
147   // Get the voltage from the ADC
148   int adc_get_voltage(unsigned int choose)
149   {
150       unsigned long ul_delay_count;
151
152       switch_relais(choose);
153
154       // ca. 1 ms delay für ADC reset
155       ul_delay_count = 1000;
156       while (ul_delay_count) ul_delay_count--;
157       //--------------------------------
158
159
160       unsigned int output = (unsigned long long)adc_get_single_value();
161       unsigned int voltage = (output*config.B[choose] - output*output/1000*config.A[
          choose])/1000 + config.C[choose];
162       switch_relais(0);
163
164       if(voltage <= config.C[choose] * 2){
165           voltage = 0;
166       }
167
168       g_iCellVoltages[choose - 1] = voltage;
169
170       return voltage;
171   }
172
173   // Get the current from the ADC
174   int adc_get_current(void){
175
176       long current = (unsigned long long)adc_get_single_value_ch2() * 76293945 /
```

```
        1000000000 * 3745  / 100 - 93625;
177
178     if(abs(current) <= 115){
179         current = 0;
180     }
181
182     if(current <= -75000 || current >= 75000){
183         current = (unsigned long long)adc_get_single_value_ch3() * 76293945 /
            1000000000 * 250 - 625000;
184
185         if(abs(current) <= 115){
186             current = 0;
187         }
188     }
189
190     g_iCurrent = (-1) * current;//positive current for charging, negative for
        discharge
191
192     return (int)g_iCurrent;
193 }
194
195
196
197 // Get the value of the ADC Ch1
198 unsigned short adc_get_single_value(void)
199 {
200     unsigned short data;
201
202     // Configure unipolar mode, gain = 1, unbuffered, noref , CH1 = 0x1000
203     adc_set_config_reg(0x1000);
204
205     // set ADC to single conversion mode with f_ADC = 50 Hz =  0x2005
206     // set ADC to single conversion mode with f_ADC = 62 Hz =  0x2004
207     // set ADC to single conversion mode with f_ADC = 123 Hz = 0x2003
208     // set ADC to single conversion mode with f_ADC = 470 Hz = 0x2001
209     adc_set_mode_reg(0x2008);
210
211     //DEBUG TRIGGER switch off for measure
212     //GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, 0);
213
214     // wait for conversion to be done
215     while((adc_status() & (1<<7)));
216
217     data = (adc_read_data());
218
219     return data;
220 }
221
222 // Get the value of the ADC Ch2
223 unsigned short adc_get_single_value_ch2(void)
224 {
225     unsigned short data;
226
227     // Configure unipolar mode, gain = 1, unbuffered, noref , CH2 = 0x1001
228     adc_set_config_reg(0x1001);
229
230     // set ADC to single conversion mode with f_ADC = 123 Hz
231     adc_set_mode_reg(0x2003);
232
```

```
233       // wait for conversion to be done
234       while((adc_status() & (1<<7)));
235
236       data = (adc_read_data()) + config.zerocurrentch1;
237
238       return data;
239   }
240
241   // Get the value of the ADC Ch3
242   unsigned short adc_get_single_value_ch3(void)
243   {
244       unsigned short data;
245
246       // Configure unipolar mode, gain = 1, unbuffered, noref , CH3 = 0x1002
247       adc_set_config_reg(0x1002);
248
249       // set ADC to single conversion mode with f_ADC = 123 Hz
250       adc_set_mode_reg(0x2003);
251
252       // wait for conversion to be done
253       while((adc_status() & (1<<7)));
254
255       data = (adc_read_data())  + config.zerocurrentch2;
256
257       return data;
258   }
259
260   /*
261    * Auslesen des Konfigurations Registers des ADCs.
262    * Es werden hierfür zwei Bytes gelesen und anschließend
263    * zusammengesetzt.
264    */
265   unsigned short adc_read_config_reg(void)
266   {
267       unsigned short data;
268       spiChipSelect(ADC);
269       adc_write_read(0x50);
270       data = (adc_write_read(0x00))<<8;
271       data |= adc_write_read(0x00);
272       spiChipSelect(NONE);
273       return data;
274   }
275
276   /*
277    * Lesen des Status Registers vom ADC.
278    * Return: 1 Byte Status Register
279    */
280   unsigned char adc_status(void)
281   {
282       spiChipSelect(ADC);
283       adc_clear_fifo();
284       adc_write_read(0x40);
285       unsigned char status_reg = adc_write_read(0x00);
286
287       spiChipSelect(NONE);
288       return status_reg;
289   }
290
291   /*
```

```
292  * Schreib-, Leseoperation für den ADC über SSI
293  */
294 unsigned char adc_write_read(unsigned char data)
295 {
296
297     adc_clear_fifo();
298
299     unsigned long ans;
300     SSIDataPut(SSI1_BASE, data);
301     SSIDataGet(SSI1_BASE, &ans);
302
303     return (unsigned char)(ans&0xff);
304
305 }
306
307 /*
308  * Löschen des Hardware FIFOs vom SPI-Interface
309  */
310 void adc_clear_fifo(void)
311 {
312
313     unsigned long dummy;
314     while(SSIDataGetNonBlocking(SSI1_BASE, &dummy));  // clear the fifo
315
316 }
317
318 /*
319  * Auslesen des Fullscale Registers des ADC.
320  * Dieses ist zwei Bytes groß, die hier automatisch
321  * beide gelsen und zusammen gesetzt werden.
322  */
323 unsigned short adc_read_fs_reg(void)
324 {
325     unsigned short data;
326     spiChipSelect(ADC);
327     adc_write_read(0x78);
328     data = (adc_write_read(0x00))<<8;
329     data |= adc_write_read(0x00);
330     spiChipSelect(NONE);
331     return data;
332 }
333
334 /*
335  * Auslesen des Offsetregisters des ADC.
336  * Dieses ist zwei Bytes groß, die hier automatisch
337  * beide gelsen und zusammen gesetzt werden.
338  */
339 unsigned short adc_read_offset_reg(void)
340 {
341     unsigned short data;
342     spiChipSelect(ADC);
343     adc_write_read(0x70);
344     data = (adc_write_read(0x00))<<8;
345     data |= adc_write_read(0x00);
346     spiChipSelect(NONE);
347     return data;
348 }
349
350 /*
```

```
351     * Schreiben des Konfigurationsregsters
352     * Das Register ist 2 Bytes groß, es muss daher ein
353     * zwei Byte short Wert übergeben werden. Es erfolgt
354     * keine Prüfung.
355     */
356    void adc_set_config_reg(unsigned short value)
357    {
358        spiChipSelect(ADC);
359        adc_write_read(0x10);
360        adc_write_read((unsigned char)(value>>8));
361        adc_write_read((unsigned char)(value&0xff));
362        spiChipSelect(NONE);
363    }
364
365    /*
366    * Schreiben des Mode Registers.
367    * Das Register ist 2 Bytes groß, es muss daher ein
368    * zwei Byte short Wert übergeben werden. Es erfolgt
369    * keine Prüfung.
370     */
371    void adc_set_mode_reg(unsigned short value)
372    {
373        spiChipSelect(ADC);
374        adc_write_read(0x08);
375        adc_write_read((unsigned char)(value>>8));
376        adc_write_read((unsigned char)(value&0xff));
377        spiChipSelect(NONE);
378    }
379
380    /*
381     * Setzen des Fullscale Registers des ADC.
382     * Es erfolgt keine Prüfung!
383     */
384    void adc_set_fs_reg(unsigned short value)
385    {
386        spiChipSelect(ADC);
387        adc_write_read(0x38);
388        adc_write_read((unsigned char)(value>>8));
389        adc_write_read((unsigned char)(value));
390        spiChipSelect(NONE);
391    }
392
393    /*
394     * Setzen des Offsetregisters des ADC.
395     * Es erfolgt keine Prüfung!
396     */
397    void adc_set_offset_reg(unsigned short value)
398    {
399        spiChipSelect(ADC);
400        adc_write_read(0x30);
401        adc_write_read((unsigned char)(value>>8));
402        adc_write_read((unsigned char)(value));
403        spiChipSelect(NONE);
404    }
405
406    /*
407     * Auslesen der Daten vom ADC.
408     * Es werden hierfür zwei Bytes gelesen und anschließend
409     * zusammengesetzt. Diese Funktion wird nur intern
```

```
410     * verwendet!
411     */
412    unsigned short adc_read_data(void)
413    {
414        unsigned short data;
415        spiChipSelect(ADC);
416        adc_write_read(0x58);
417        data = (adc_write_read(0x00))<<8;
418        data |= adc_write_read(0x00);
419        spiChipSelect(NONE);
420        return data;
421    }
422
```

```
/*
 * 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.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 8555 of the Stellaris Firmware Development Package.
//
//*****************************************************************************

#ifndef __CMDLINE_H__
#define __CMDLINE_H__

//*****************************************************************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
//*****************************************************************************
#ifdef __cplusplus
extern "C"
{
#endif

//*****************************************************************************
//
//! \addtogroup cmdline_api
//! @{
//
//*****************************************************************************

//*****************************************************************************
//
//! Defines the value that is returned if the command is not found.
//
```

```
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__
134
```

```
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("\n%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   }
325
326
327
```

```
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_ */
19
```

```
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 }
106
107
108
109
110
```

```
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
56  //*****************************************************************************
57  //
58  // This buffer holds the full path to the current working directory.
59  // Initially it is root ("/").
```

```
60  //
61  //*****************************************************************************
62  static char g_cCwdBuf[PATH_BUF_SIZE] = "/";
63
64  //*****************************************************************************
65  //
66  // A temporary data buffer used when manipulating file paths, or reading data
67  // from the SD card.
68  //
69  //*****************************************************************************
70  static char g_cTmpBuf[PATH_BUF_SIZE];
71
72
73  //*****************************************************************************
74  //
75  // The following are data structures used by FatFs.
76  //
77  //*****************************************************************************
78  static FATFS g_sFatFs;
79  static DIR g_sDirObject;
80  static FILINFO g_sFileInfo;
81  static FIL g_sFileObject;
82
83
84
85  #endif /* MYSDCARD_H_ */
86
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               mysdcard.c
4
5   Auhtor:                             Thomas Wisniewski
6   Credits:                            Matthias Schneider
7                                       Johannes Roehn
8                                       Stellaris Ware
9
10  last modified:                      2013/12/11
11
12  Project Status                      Under Construction
13  Status:                             running
14
15  CCS:                                5.5.1.00031
16  Stellarisware:                      8555
17
18  Hardware:                           Stellaris EKS-LM3S9B92 on Extension Board with
19                                      16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
20                                      NTC-Connectors, MAX3232 and Suplly Circuits
21
22  Description:                        Source Code for SDCard at SPI used with FatFs
23
24                                      Mainly copied from ti stellarisware example:
25                                      stellarisware/boards/ek-lm329b96/sdcard
26  */
27
28  //*****************************************************************************
29  *
30  * Original Stellaris Ware Includings
31  *
32  *****************************************************************************/
33
34  #include <string.h>
35  #include "inc/hw_memmap.h"
36  #include "inc/hw_types.h"
37  #include "inc/hw_ssi.h"
38  #include <inc/lm3s9b92.h>
39  #include "driverlib/gpio.h"
40  #include "driverlib/interrupt.h"
41  #include "driverlib/ssi.h"
42  #include "driverlib/sysctl.h"
43  #include "driverlib/systick.h"
44  #include <driverlib/gpio.h>
45  #include "driverlib/rom.h"
46  #include "driverlib/timer.h"
47  #include "header/mycmdline.h"
48  #include "utils/uartstdio.h"
49  #include "utils/ustdlib.h"
50  #include "third_party/fatfs/src/ff.h"
51  #include "third_party/fatfs/src/diskio.h"
52
53  //*****************************************************************************
54  *
55  * Own Includings
56  *
57  *****************************************************************************/
58
59  #include "header/mysdcard.h"
```

```
60  #include "header/myadc.h"
61  #include "header/temperature.h"
62  #include "header/clocktimer.h"
63  #include "header/config.h"
64  #include "header/control.h"
65  #include "header/display.h"
66  #include "header/relais.h"
67  #include "stdio.h"
68  #include "utils/lwiplib.h"
69  #include "header/myssi.h"
70
71
72  //*****************************************************************************
73  //Extern variables for Clock
74  //*****************************************************************************
75  extern volatile unsigned long clock_msec;
76  extern volatile unsigned long clock_sec;
77  extern volatile unsigned long clock_min;
78  extern volatile unsigned long clock_hour;
79  extern volatile unsigned long clock_day;
80  extern volatile unsigned long clock_month;
81  extern volatile unsigned long clock_year;
82
83  //*****************************************************************************
84  //Extern variable for current active power relay
85  //*****************************************************************************
86  extern volatile unsigned int power_relay_active;
87  extern volatile unsigned long timer2minute;
88  extern volatile unsigned int cycle_active;
89  extern volatile unsigned int measurement_active;
90  extern volatile unsigned int current_cycle_step;
91  extern volatile int gBat_charge;
92  //*****************************************************************************
93  //Extern variable for current, cell voltages and temperatures
94  //*****************************************************************************
95  extern volatile unsigned int g_iCurrent;
96  extern volatile unsigned int g_iCellVoltages[12];
97  extern volatile int g_iCellTemperature[12];
98
99  //Variables for cat command execution in main
100 extern int g_cat_argc;
101 extern char g_cat_argv[], g_cat_flag;
102
103 //Minute Counter for maximum Time
104 extern volatile int max_time_counter;
105
106 extern volatile unsigned int wait_start_time;
107
108 //*****************************************************************************
109 //Variable for filename for measurment saving
110 //*****************************************************************************
111 char MEAS_FILE[64];
112
113 ////*************************************************************************
114 ////
115 //// Defines the size of the buffers that hold the path, or temporary
116 //// data from the SD card.  There are two buffers allocated of this size.
117 //// The buffer size must be large enough to hold the longest expected
118 //// full path name, including the file name, and a trailing null character.
```

```
119 ////
120 ////*****************************************************************************
121 //#define PATH_BUF_SIZE   80
122
123
124 ////*****************************************************************************
125 ////
126 //// This buffer holds the full path to the current working directory.
127 //// Initially it is root ("/").
128 ////
129 ////*****************************************************************************
130 //static char g_cCwdBuf[PATH_BUF_SIZE] = "/";
131 //
132 ////*****************************************************************************
133 ////
134 //// A temporary data buffer used when manipulating file paths, or reading data
135 //// from the SD card.
136 ////
137 ////*****************************************************************************
138 //static char g_cTmpBuf[PATH_BUF_SIZE];
139 //
140 //
141 ////*****************************************************************************
142 ////
143 //// The following are data structures used by FatFs.
144 ////
145 ////*****************************************************************************
146 //static FATFS g_sFatFs;
147 //static DIR g_sDirObject;
148 //static FILINFO g_sFileInfo;
149 //static FIL g_sFileObject;
150
151
152 //*****************************************************************************
153 //
154 // A structure that holds a mapping between an FRESULT numerical code,
155 // and a string represenation.  FRESULT codes are returned from the FatFs
156 // FAT file system driver.
157 //
158 //*****************************************************************************
159 typedef struct
160 {
161     FRESULT fresult;
162     char *pcResultStr;
163 }
164 tFresultString;
165
166 //*****************************************************************************
167 //
168 // A macro to make it easy to add result codes to the table.
169 //
170 //*****************************************************************************
171 #define FRESULT_ENTRY(f)        { (f), (#f) }
172
173 //*****************************************************************************
174 //
175 // A table that holds a mapping between the numerical FRESULT code and
176 // it's name as a string.  This is used for looking up error codes for
177 // printing to the console.
```

```
178 //
179 //*****************************************************************************
180 tFresultString g_sFresultStrings[] =
181 {
182     FRESULT_ENTRY(FR_OK),
183     FRESULT_ENTRY(FR_NOT_READY),
184     FRESULT_ENTRY(FR_NO_FILE),
185     FRESULT_ENTRY(FR_NO_PATH),
186     FRESULT_ENTRY(FR_INVALID_NAME),
187     FRESULT_ENTRY(FR_INVALID_DRIVE),
188     FRESULT_ENTRY(FR_DENIED),
189     FRESULT_ENTRY(FR_EXIST),
190     FRESULT_ENTRY(FR_RW_ERROR),
191     FRESULT_ENTRY(FR_WRITE_PROTECTED),
192     FRESULT_ENTRY(FR_NOT_ENABLED),
193     FRESULT_ENTRY(FR_NO_FILESYSTEM),
194     FRESULT_ENTRY(FR_INVALID_OBJECT),
195     FRESULT_ENTRY(FR_MKFS_ABORTED)
196 };
197
198 //*****************************************************************************
199 //
200 // A macro that holds the number of result codes.
201 //
202 //*****************************************************************************
203 #define NUM_FRESULT_CODES (sizeof(g_sFresultStrings) / sizeof(tFresultString))
204
205 //*****************************************************************************
206 //
207 // This function returns a string representation of an error code
208 // that was returned from a function call to FatFs.  It can be used
209 // for printing human readable error messages.
210 //
211 //*****************************************************************************
212 const char *
213 StringFromFresult(FRESULT fresult)
214 {
215     unsigned int uIdx;
216
217     //
218     // Enter a loop to search the error code table for a matching
219     // error code.
220     //
221     for(uIdx = 0; uIdx < NUM_FRESULT_CODES; uIdx++)
222     {
223         //
224         // If a match is found, then return the string name of the
225         // error code.
226         //
227         if(g_sFresultStrings[uIdx].fresult == fresult)
228         {
229             return(g_sFresultStrings[uIdx].pcResultStr);
230         }
231     }
232
233     //
234     // At this point no matching code was found, so return a
235     // string indicating unknown error.
236     //
```

```
237         return("UNKNOWN ERROR CODE");
238     }
239
240
241     //*****************************************************************************
242     //
243     // This is the table that holds the command names, implementing functions,
244     // and brief description.
245     //
246     //*****************************************************************************
247     tCmdLineEntry g_sBrowserCmdTable[] =
248     {
249         { "help",   Cmd_help,      " : Display list of commands" },
250         { "h",      Cmd_help,   "    : alias for help" },
251         { "?",      Cmd_help,   "    : alias for help" },
252         { "reinit", Cmd_reinit,      ": Re initialize the SD-Card filesystem" },
253         { "ls",     Cmd_ls,      "   : Display list of files" },
254         { "chdir",  Cmd_cd,         ": Change directory" },
255         { "cd",     Cmd_cd,      "   : alias for chdir" },
256         { "pwd",    Cmd_pwd,      "  : Show current working directory" },
257         { "cat",    Cmd_cat,      "  : Show contents of a text file (use only in Idle
            Mode)" },
258         { "cre",    Cmd_cre,      "  : Create a file" },
259         { "del",    Cmd_del,      "  : Delete a file" },
260         { "exit",   Cmd_sdcard_exit,      " : Exit SD-Card browser" },
261         { 0, 0, 0 }
262     };
263
264     extern tCmdLineEntry g_sMainCmdTable;
265     extern tCmdLineEntry *g_psCmdTable;
266
267     // string for main location
268     extern const char *g_cMainLocalBuf;
269
270     // Global location buffer
271     extern char *g_cLocalBuf;
272
273     // TCP-Socket für die aktuelle Steuerungs-Verbindung
274     extern struct tcp_pcb* control_connection;
275
276     // String buffer for print operations to UART and Ethernet
277     char print_buffer[80];
278
279     //define string for SD-Card location
280     char *g_cSdLocalBuf  = "SD-Card";
281
282
283     //*****************************************************************************
284     //
285     // This is the handler for this SysTick interrupt.  FatFs requires a
286     // timer tick every 10 ms for internal timing purposes.
287     //
288     //*****************************************************************************
289     void
290     SysTickHandler(void)
291     {
292         //
293         // Call the FatFs tick timer.
294         //
```

```
295       TimerClock();
296       disk_timerproc();
297       lwIPTimer(10);
298
299   }
300
301   //*****************************************************************************
302   //
303   // This function implements the "re-init" command.
304   // The SD-Card filesystem will be reinitialized after plugging in the SD-Card
305   // while the system is running
306   //
307   //*****************************************************************************
308   int
309   Cmd_reinit(int argc, char *argv[])
310   {
311      mysdcardinit();
312      return 0;
313   }
314
315   //*****************************************************************************
316   //
317   // This function implements the "ls" command.  It opens the current
318   // directory and enumerates through the contents, and prints a line for
319   // each item it finds.  It shows details such as file attributes, time and
320   // date, and the file size, along with the name.  It shows a summary of
321   // file sizes at the end along with free space.
322   //
323   //*****************************************************************************
324   int
325   Cmd_ls(int argc, char *argv[])
326   {
327
328       unsigned long ulTotalSize;
329       unsigned long ulFileCount;
330       unsigned long ulDirCount;
331       FRESULT fresult;
332       FATFS *pFatFs;
333
334       //
335       // Open the current directory for access.
336       //
337       fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
338
339       //
340       // Check for error and return if there is a problem.
341       //
342       if(fresult != FR_OK)
343       {
344           return(fresult);
345       }
346
347       ulTotalSize = 0;
348       ulFileCount = 0;
349       ulDirCount = 0;
350
351       //
352       // Give an extra blank line before the listing.
353       //
```

```
354       UARTprintf("\n");
355
356       //
357       // Enter loop to enumerate through all directory entries.
358       //
359       for(;;)
360       {
361           //
362           // Read an entry from the directory.
363           //
364           fresult = f_readdir(&g_sDirObject, &g_sFileInfo);
365
366           //
367           // Check for error and return if there is a problem.
368           //
369           if(fresult != FR_OK)
370           {
371               sprintf(print_buffer, "Error at reading File System Entry ");
372
373               UARTprintf(print_buffer);               // debug information
374               if(control_connection){
375                   telnet_write(print_buffer);
376               }
377               return(fresult);
378           }
379
380           //
381           // If the file name is blank, then this is the end of the
382           // listing.
383           //
384           if(!g_sFileInfo.fname[0])
385           {
386               break;
387           }
388
389           //
390           // If the attribue is directory, then increment the directory count.
391           //
392           if(g_sFileInfo.fattrib & AM_DIR)
393           {
394               ulDirCount++;
395           }
396
397           //
398           // Otherwise, it is a file.  Increment the file count, and
399           // add in the file size to the total.
400           //
401           else
402           {
403               ulFileCount++;
404               ulTotalSize += g_sFileInfo.fsize;
405           }
406
407           //
408           // Print the entry information on a single line with formatting
409           // to show the attributes, date, time, size, and name.
410           //
411           sprintf(print_buffer, "%c%c%c%c%c %u/%02u/%02u %02u:%02u %9u  %s\n",
412                               (g_sFileInfo.fattrib & AM_DIR) ? 'D' : '-',
```

```
413                             (g_sFileInfo.fattrib & AM_RDO) ? 'R' : '-',
414                             (g_sFileInfo.fattrib & AM_HID) ? 'H' : '-',
415                             (g_sFileInfo.fattrib & AM_SYS) ? 'S' : '-',
416                             (g_sFileInfo.fattrib & AM_ARC) ? 'A' : '-',
417                             (g_sFileInfo.fdate >> 9) + 1980,
418                             (g_sFileInfo.fdate >> 5) & 15,
419                              g_sFileInfo.fdate & 31,
420                             (g_sFileInfo.ftime >> 11),
421                             (g_sFileInfo.ftime >> 5) & 63,
422                              g_sFileInfo.fsize,
423                              g_sFileInfo.fname);
424
425           UARTprintf(print_buffer);
426           if(control_connection){
427               telnet_write(print_buffer);
428           }
429       }   // endfor
430
431       //
432       // Print summary lines showing the file, dir, and size totals.
433       //
434       sprintf(print_buffer, "\n%4u File(s),%10u bytes total\n%4u Dir(s)",
435               ulFileCount, ulTotalSize, ulDirCount);
436       UARTprintf(print_buffer);
437       if(control_connection){
438           telnet_write(print_buffer);
439       }
440       //
441       // Get the free space.
442       //
443       fresult = f_getfree("/", &ulTotalSize, &pFatFs);
444
445       //
446       // Check for error and return if there is a problem.
447       //
448       if(fresult != FR_OK)
449       {
450           //UARTprintf(" Error at the end of the line ");     // debug information
451           return(fresult);
452       }
453
454       //
455       // Display the amount of free space that was calculated.
456       //
457       sprintf(print_buffer, ", %10uK bytes free\n", ulTotalSize * pFatFs->sects_clust /
           2);
458
459       UARTprintf(print_buffer);
460       if(control_connection){
461           telnet_write(print_buffer);
462       }
463
464
465       //
466       // Made it to here, return with no errors.
467       //
468       return(0);
469   }
470
```

```
471 //*****************************************************************************
472 //
473 // This function implements the "cd" command.  It takes an argument
474 // that specifes the directory to make the current working directory.
475 // Path separators must use a forward slash "/".  The argument to cd
476 // can be one of the following:
477 // * root ("/")
478 // * a fully specified path ("/my/path/to/mydir")
479 // * a single directory name that is in the current directory ("mydir")
480 // * parent directory ("..")
481 //
482 // It does not understand relative paths, so dont try something like this:
483 // ("../my/new/path")
484 //
485 // Once the new directory is specified, it attempts to open the directory
486 // to make sure it exists.  If the new path is opened successfully, then
487 // the current working directory (cwd) is changed to the new path.
488 //
489 //*****************************************************************************
490 int
491 Cmd_cd(int argc, char *argv[])
492 {
493     unsigned int uIdx;
494     FRESULT fresult;
495 
496     //
497     // Copy the current working path into a temporary buffer so
498     // it can be manipulated.
499     //
500     strcpy(g_cTmpBuf, g_cCwdBuf);
501 
502     //
503     // If the first character is /, then this is a fully specified
504     // path, and it should just be used as-is.
505     //
506     if(argv[1][0] == '/')
507     {
508         //
509         // Make sure the new path is not bigger than the cwd buffer.
510         //
511         if(strlen(argv[1]) + 1 > sizeof(g_cCwdBuf))
512         {
513             sprintf(print_buffer, "Resulting path name is too long\n");
514 
515             UARTprintf(print_buffer);
516             if(control_connection){
517                 telnet_write(print_buffer);
518             }
519             return(0);
520         }
521 
522         //
523         // If the new path name (in argv[1])  is not too long, then
524         // copy it into the temporary buffer so it can be checked.
525         //
526         else
527         {
528             strncpy(g_cTmpBuf, argv[1], sizeof(g_cTmpBuf));
529         }
```

```
530       }
531
532       //
533       // If the argument is .. then attempt to remove the lowest level
534       // on the CWD.
535       //
536       else if(!strcmp(argv[1], ".."))
537       {
538           //
539           // Get the index to the last character in the current path.
540           //
541           uIdx = strlen(g_cTmpBuf) - 1;
542
543           //
544           // Back up from the end of the path name until a separator (/)
545           // is found, or until we bump up to the start of the path.
546           //
547           while((g_cTmpBuf[uIdx] != '/') && (uIdx > 1))
548           {
549               //
550               // Back up one character.
551               //
552               uIdx--;
553           }
554
555           //
556           // Now we are either at the lowest level separator in the
557           // current path, or at the beginning of the string (root).
558           // So set the new end of string here, effectively removing
559           // that last part of the path.
560           //
561           g_cTmpBuf[uIdx] = 0;
562       }
563
564       //
565       // Otherwise this is just a normal path name from the current
566       // directory, and it needs to be appended to the current path.
567       //
568       else
569       {
570           //
571           // Test to make sure that when the new additional path is
572           // added on to the current path, there is room in the buffer
573           // for the full new path.  It needs to include a new separator,
574           // and a trailing null character.
575           //
576           if(strlen(g_cTmpBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cCwdBuf))
577           {
578               sprintf(print_buffer, "Resulting path name is too long\n");
579
580               UARTprintf(print_buffer);
581
582               if(control_connection){
583                   telnet_write(print_buffer);
584               }
585               return(0);
586           }
587
588           //
```

```
589             // The new path is okay, so add the separator and then append
590             // the new directory to the path.
591             //
592             else
593             {
594                 //
595                 // If not already at the root level, then append a /
596                 //
597                 if(strcmp(g_cTmpBuf, "/"))
598                 {
599                     strcat(g_cTmpBuf, "/");
600                 }
601
602                 //
603                 // Append the new directory to the path.
604                 //
605                 strcat(g_cTmpBuf, argv[1]);
606             }
607         }
608
609         //
610         // At this point, a candidate new directory path is in chTmpBuf.
611         // Try to open it to make sure it is valid.
612         //
613         fresult = f_opendir(&g_sDirObject, g_cTmpBuf);
614
615         //
616         // If it cant be opened, then it is a bad path.  Inform
617         // user and return.
618         //
619         if(fresult != FR_OK)
620         {
621             sprintf(print_buffer, "cd: %s\n", g_cTmpBuf);
622
623             UARTprintf(print_buffer);
624
625             if(control_connection){
626                 telnet_write(print_buffer);
627             }
628             return(fresult);
629         }
630
631         //
632         // Otherwise, it is a valid new path, so copy it into the CWD.
633         //
634         else
635         {
636             strncpy(g_cCwdBuf, g_cTmpBuf, sizeof(g_cCwdBuf));
637         }
638
639
640         //
641         // Return success.
642         //
643         return(0);
644     }
645
646     //*****************************************************************************
647     //
```

```
648 // This function implements the "pwd" command.  It simply prints the
649 // current working directory.
650 //
651 //*****************************************************************************
652 int
653 Cmd_pwd(int argc, char *argv[])
654 {
655
656     //
657     // Print the CWD to the console.
658     //
659     sprintf(print_buffer, "%s\n", g_cCwdBuf);
660
661     UARTprintf(print_buffer);
662
663     if(control_connection){
664         telnet_write(print_buffer);
665     }
666
667
668     //
669     // Return success.
670     //
671     return(0);
672 }
673
674 //*****************************************************************************
675 //
676 // This function implements the "cat" command.  It reads the contents of
677 // a file and prints it to the console.  This should only be used on
678 // text files.  If it is used on a binary file, then a bunch of garbage
679 // is likely to printed on the console.
680 //
681 //*****************************************************************************
682 int
683 Cmd_cat(int argc, char *argv[])
684 {
685     //set flag for execution in main loop
686     g_cat_flag = 1;
687
688     //hand over argc and argv
689     g_cat_argc = argc;
690     strcpy(g_cat_argv, argv[1]);
691
692     //
693     // Return success.
694     //
695     return(0);
696 }
697
698 //*****************************************************************************
699 //
700 // This function implements the "del" command.  It deletes a file
701 //
702 //*****************************************************************************
703 int
704 Cmd_del(int argc, char *argv[])
705 {
706
```

```
707       FRESULT fresult;
708
709       //
710       // First, check to make sure that the current path (CWD), plus
711       // the file name, plus a separator and trailing null, will all
712       // fit in the temporary buffer that will be used to hold the
713       // file name.  The file name must be fully specified, with path,
714       // to FatFs.
715       //
716       if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
717       {
718           sprintf(print_buffer, "Resulting path name is too long\n");
719
720           UARTprintf(print_buffer);
721
722           if(control_connection){
723               telnet_write(print_buffer);
724           }
725           return(0);
726       }
727
728       //
729       // Copy the current path to the temporary buffer so it can be manipulated.
730       //
731       strcpy(g_cTmpBuf, g_cCwdBuf);
732
733       //
734       // If not already at the root level, then append a separator.
735       //
736       if(strcmp("/", g_cCwdBuf))
737       {
738           strcat(g_cTmpBuf, "/");
739       }
740
741       //
742       // Now finally, append the file name to result in a fully specified file.
743       //
744       strcat(g_cTmpBuf, argv[1]);
745
746       //
747           // Delete File
748           //
749             fresult = f_unlink(g_cTmpBuf);
750
751             //
752             // Check if operation succeeded
753             //
754
755             if(fresult != FR_OK)
756             {
757                 return(fresult);
758             }
759
760
761
762       //
763       // Return success.
764       //
765       return(0);
```

```
766
767 }
768
769 //*****************************************************************************
770 //
771 // This function implements the "cre" command.  It creates a file
772 //
773 //*****************************************************************************
774 int
775 Cmd_cre(int argc, char *argv[])
776 {
777
778     FRESULT fresult;
779     FIL fnew;      /* new file object */
780
781     //
782     // First, check to make sure that the current path (CWD), plus
783     // the file name, plus a separator and trailing null, will all
784     // fit in the temporary buffer that will be used to hold the
785     // file name.  The file name must be fully specified, with path,
786     // to FatFs.
787     //
788     if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
789     {
790         sprintf(print_buffer, "Resulting path name is too long\n");
791
792         UARTprintf(print_buffer);
793
794         if(control_connection){
795             telnet_write(print_buffer);
796         }
797         return(0);
798     }
799
800     //
801     // Copy the current path to the temporary buffer so it can be manipulated.
802     //
803     strcpy(g_cTmpBuf, g_cCwdBuf);
804
805     //
806     // If not already at the root level, then append a separator.
807     //
808     if(strcmp("/", g_cCwdBuf))
809     {
810         strcat(g_cTmpBuf, "/");
811     }
812
813     //
814     // Now finally, append the file name to result in a fully specified file.
815     //
816     strcat(g_cTmpBuf, argv[1]);
817
818     //
819     // Create the File
820     //
821     fresult = f_open(&fnew, argv[1], FA_CREATE_ALWAYS | FA_WRITE );
822
823     //
824     // Check if creation succeeded
```

```
825         //
826
827         if(fresult != FR_OK)
828         {
829             return(fresult);
830         }
831
832         /* Close opened files */
833         f_close(&fnew);
834
835
836         //
837         // Return success.
838         //
839         return(0);
840
841     }
842
843     //*****************************************************************************
844     //
845     // This function implements the "help" command.  It prints a simple list
846     // of the available commands with a brief description.
847     //
848     //*****************************************************************************
849     int
850     Cmd_help(int argc, char *argv[])
851     {
852
853         tCmdLineEntry *pEntry;
854
855         //
856         // Print some header text.
857         //
858
859         UARTprintf("\nAvailable commands\n");
860         UARTprintf("------------------\n");
861
862         if(control_connection){
863             telnet_write("Available commands\n");
864             telnet_write("------------------\n");
865         }
866
867         //
868         // Point at the beginning of the command table.
869         //
870         pEntry = g_psCmdTable;
871
872         //
873         // Enter a loop to read each entry from the command table.  The
874         // end of the table has been reached when the command name is NULL.
875         //
876         while(pEntry->pcCmd)
877         {
878             //
879             // Print the command name and the brief description.
880             //
881             sprintf(print_buffer, "%s%s\n", pEntry->pcCmd, pEntry->pcHelp);
882
883             UARTprintf(print_buffer);
```

```
884             if(control_connection){
885                 telnet_write(print_buffer);
886             }
887 
888             //
889             // Advance to the next entry in the table.
890             //
891             pEntry++;
892         }
893 
894 
895         //
896         // Return success.
897         //
898         return(0);
899     }
900 
901     int Cmd_sdcard_exit(int argc, char *argv[])
902     {
903 
904         // set Command line pointer to the beginning of the main command structure
905         g_psCmdTable = &g_sMainCmdTable;
906 
907         // set location buffer to main
908         g_cLocalBuf = (char*)g_cMainLocalBuf;
909 
910         return(0);
911     }
912 
913     //*****************************************************************************
914     //
915     // This is the table that holds the command names, implementing functions,
916     // and brief description.
917     //
918     //*****************************************************************************
919     //extern tCmdLineEntry g_sCmdTable[];
920 
921 
922     //*****************************************************************************
923     //
924     // Own Initialization of SSI0, SysTick, FatFs and SDcard
925     //
926     //*****************************************************************************
927 
928     void mysdcardinit(void)
929     {
930 
931         // variable for FatFs results
932         FRESULT fresult = FR_NOT_READY;
933 
934         UARTprintf("SD card initialization...");
935 
936         // Configure SysTick for a 100Hz interrupt.  The FatFs driver
937         // wants a 10 ms tick.
938 
939         SysTickPeriodSet(SysCtlClockGet() / 100);
940         SysTickEnable();
941         SysTickIntEnable();
942 
```

```
943         fresult = f_mount(0, &g_sFatFs);
944 
945         if(fresult != FR_OK)
946         {
947             UARTprintf("f_mount error: %s\n", StringFromFresult(fresult));
948         }
949         // debugging stuff
950         //else
951         //{
952         //    UARTprintf(" f_mount successful\n");
953         //}
954 
955         // reset status flag
956         fresult = FR_NOT_READY;
957 
958         // Open the current directory for access.
959         fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
960 
961         // Check for error and return if there is a problem.
962         if(fresult != FR_OK)
963         {
964             UARTprintf(" f_opendir error: %s\n", StringFromFresult(fresult));
965         }
966         else
967         {
968             UARTprintf("done.\n");
969         }
970         // debugging stuff
971         //else
972         //{
973         //    UARTprintf(" f_opendir successful\n");
974         //}
975     }
976 
977 
978 
979 
980 
981     //*****************************************************************************
982     //
983     // This function implements the "read_File" command.  It reads the contents of
984     // a file and prints it to the console.  This should only be used on
985     // text files.  If it is used on a binary file, then a bunch of garbage
986     // is likely to printed on the console.
987     //
988     //*****************************************************************************
989     int read_file(const char *filename)
990     {
991 
992         FRESULT fresult;
993         unsigned short usBytesRead;
994 
995         //
996         // First, check to make sure that the current path (CWD), plus
997         // the file name, plus a separator and trailing null, will all
998         // fit in the temporary buffer that will be used to hold the
999         // file name.  The file name must be fully specified, with path,
1000        // to FatFs.
1001        //
```

```
1002        if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1003        {
1004            UARTprintf("Resulting path name is too long\n");
1005            return(0);
1006        }
1007
1008        //
1009        // Copy the current path to the temporary buffer so it can be manipulated.
1010        //
1011        strcpy(g_cTmpBuf, g_cCwdBuf);
1012
1013        //
1014        // If not already at the root level, then append a separator.
1015        //
1016        if(strcmp("/", g_cCwdBuf))
1017        {
1018            strcat(g_cTmpBuf, "/");
1019        }
1020
1021        //
1022        // Now finally, append the file name to result in a fully specified file.
1023        //
1024        strcat(g_cTmpBuf, filename);
1025
1026        //
1027        // Open the file for reading.
1028        //
1029        fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1030
1031        //
1032        // If there was some problem opening the file, then return
1033        // an error.
1034        //
1035        if(fresult != FR_OK)
1036        {
1037            return(fresult);
1038        }
1039
1040        //
1041        // Enter a loop to repeatedly read data from the file and display it,
1042        // until the end of the file is reached.
1043        //
1044        do
1045        {
1046            //
1047            // Read a block of data from the file.  Read as much as can fit
1048            // in the temporary buffer, including a space for the trailing null.
1049            //
1050            fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf) - 1,
1051                             &usBytesRead);
1052
1053            //
1054            // If there was an error reading, then print a newline and
1055            // return the error to the user.
1056            //
1057            if(fresult != FR_OK)
1058            {
1059                UARTprintf("\n");
1060                return(fresult);
```

```
1061            }
1062
1063            //
1064            // Null terminate the last block that was read to make it a
1065            // null terminated string that can be used with printf.
1066            //
1067            g_cTmpBuf[usBytesRead] = 0;
1068
1069            //
1070            // Print the last chunk of the file that was received.
1071            //
1072
1073            UARTprintf("%s", g_cTmpBuf);
1074
1075
1076            //
1077            // Wait for the UART transmit buffer to empty.
1078            //
1079
1080            #if defined(UART_BUFFERED)
1081                UARTFlushTx(false);
1082            #endif
1083
1084        //
1085        // Continue reading until less than the full number of bytes are
1086        // read.  That means the end of the buffer was reached.
1087        //
1088        }
1089        while(usBytesRead == sizeof(g_cTmpBuf) - 1);
1090
1091        /* Close opened files */
1092        f_close(&g_sFileObject);
1093
1094
1095        //
1096        // Return success.
1097        //
1098        return(0);
1099    }
1100
1101    //*****************************************************************************
1102    //
1103    // This function implements the "read_config" command.  It reads the contents of
1104    // a file and prints it to the console.  This should only be used on
1105    // text files.  If it is used on a binary file, then a bunch of garbage
1106    // is likely to printed on the console.
1107    //
1108    //*****************************************************************************
1109    int read_into_buffer(const char *filename, char *buffer)
1110    {
1111
1112        FRESULT fresult;
1113        unsigned short usBytesRead;
1114
1115        //
1116        // First, check to make sure that the current path (CWD), plus
1117        // the file name, plus a separator and trailing null, will all
1118        // fit in the temporary buffer that will be used to hold the
1119        // file name.  The file name must be fully specified, with path,
```

```
1120        // to FatFs.
1121        //
1122        if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1123        {
1124            UARTprintf("Resulting path name is too long\n");
1125            return(0);
1126        }
1127
1128        //
1129        // Copy the current path to the temporary buffer so it can be manipulated.
1130        //
1131        strcpy(g_cTmpBuf, g_cCwdBuf);
1132
1133        //
1134        // If not already at the root level, then append a separator.
1135        //
1136        if(strcmp("/", g_cCwdBuf))
1137        {
1138            strcat(g_cTmpBuf, "/");
1139        }
1140
1141        //
1142        // Now finally, append the file name to result in a fully specified file.
1143        //
1144        strcat(g_cTmpBuf, filename);
1145
1146        //
1147        // Open the file for reading.
1148        //
1149        fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1150
1151        //
1152        // If there was some problem opening the file, then return
1153        // an error.
1154        //
1155        if(fresult != FR_OK)
1156        {
1157            return(fresult);
1158        }
1159
1160        //
1161        // Enter a loop to repeatedly read data from the file and display it,
1162        // until the end of the file is reached.
1163        //
1164        char *p = buffer;
1165        do
1166        {
1167            //
1168            // Read a block of data from the file.  Read as much as can fit
1169            // in the temporary buffer, including a space for the trailing null.
1170            //
1171            fresult = f_read(&g_sFileObject, p, sizeof(p) - 1,
1172                             &usBytesRead);
1173
1174            //
1175            // If there was an error reading, then print a newline and
1176            // return the error to the user.
1177            //
1178            if(fresult != FR_OK)
```

```
1179            {
1180                UARTprintf("\n");
1181                return(fresult);
1182            }
1183
1184            //
1185            // Null terminate the last block that was read to make it a
1186            // null terminated string that can be used with printf.
1187            //
1188            p[usBytesRead] = 0;
1189
1190            //
1191            // Print the last chunk of the file that was received.
1192            //
1193            //UARTprintf("%s", p);
1194            p += usBytesRead;
1195
1196            //
1197            // Wait for the UART transmit buffer to empty.
1198            //
1199            #if defined(UART_BUFFERED)
1200                UARTFlushTx(false);
1201            #endif
1202
1203        //
1204        // Continue reading until less than the full number of bytes are
1205        // read.  That means the end of the buffer was reached.
1206        //
1207        }
1208        while(usBytesRead == sizeof(p) - 1);
1209
1210        /* Close opened files */
1211        fresult = f_close(&g_sFileObject);
1212
1213        if(fresult != FR_OK)
1214        {
1215            return(fresult);
1216        }
1217
1218
1219        //
1220        // Return success.
1221        //
1222        return (0);
1223    }
1224
1225
1226
1227    //
1228    // This function implements the "delete_File" command.
1229    // It simply delete a file on the SD Card, selected with its filename
1230    //
1231    int delete_file(const char *filename){
1232
1233
1234            FRESULT fresult;
1235
1236            //
1237            // First, check to make sure that the current path (CWD), plus
```

```
1238            // the file name, plus a separator and trailing null, will all
1239            // fit in the temporary buffer that will be used to hold the
1240            // file name.  The file name must be fully specified, with path,
1241            // to FatFs.
1242            //
1243            if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1244            {
1245                UARTprintf("Resulting path name is too long\n");
1246                return(0);
1247            }
1248
1249            //
1250            // Copy the current path to the temporary buffer so it can be manipulated.
1251            //
1252            strcpy(g_cTmpBuf, g_cCwdBuf);
1253
1254            //
1255            // If not already at the root level, then append a separator.
1256            //
1257            if(strcmp("/", g_cCwdBuf))
1258            {
1259                strcat(g_cTmpBuf, "/");
1260            }
1261
1262            //
1263            // Now finally, append the file name to result in a fully specified file.
1264            //
1265            strcat(g_cTmpBuf, filename);
1266
1267                //
1268                // Wait for the UART transmit buffer to empty.
1269                //
1270            #if defined(UART_BUFFERED)
1271                UARTFlushTx(false);
1272            #endif
1273
1274
1275
1276        //
1277        // Delete File
1278        //
1279          fresult = f_unlink(g_cTmpBuf);
1280
1281          //
1282          // Check if operation succeeded
1283          //
1284
1285          if(fresult != FR_OK)
1286          {
1287              return(fresult);
1288          }
1289
1290
1291          //
1292          // Return success.
1293          //
1294          return(0);
1295
1296    }
```

```
1297
1298
1299  //*****************************************************************************
1300  //
1301  // This function implements the "write_to_file" command.  It overrides the content of
1302  // a file with new input.
1303  //
1304  //*****************************************************************************
1305
1306  int write_to_file(const char *filename, const char *write_Buffer){
1307
1308
1309          FIL fnew;      /* new file object */
1310          FRESULT fresult;
1311          unsigned short bw = 0;
1312
1313
1314          //
1315          // First, check to make sure that the current path (CWD), plus
1316          // the file name, plus a separator and trailing null, will all
1317          // fit in the temporary buffer that will be used to hold the
1318          // file name.  The file name must be fully specified, with path,
1319          // to FatFs.
1320          //
1321          if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1322          {
1323              UARTprintf("Resulting path name is too long\n");
1324              return(0);
1325          }
1326
1327          //
1328          // Copy the current path to the temporary buffer so it can be manipulated.
1329          //
1330          strcpy(g_cTmpBuf, g_cCwdBuf);
1331
1332          //
1333          // If not already at the root level, then append a separator.
1334          //
1335          if(strcmp("/", g_cCwdBuf))
1336          {
1337              strcat(g_cTmpBuf, "/");
1338          }
1339
1340          //
1341          // Now finally, append the file name to result in a fully specified file.
1342          //
1343          strcat(g_cTmpBuf, filename);
1344
1345              //
1346              // Wait for the UART transmit buffer to empty.
1347              //
1348              #if defined(UART_BUFFERED)
1349                  UARTFlushTx(false);
1350              #endif
1351
1352
1353      //
1354      // Create File
1355      //
```

```
1356
1357        fresult = f_open(&fnew, g_cTmpBuf, FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
1358
1359
1360
1361        if(fresult != FR_OK)
1362        {
1363            return(fresult);
1364        }
1365
1366
1367
1368        fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1369
1370        if(fresult != FR_OK)
1371        {
1372            return(fresult);
1373        }
1374
1375
1376        /* Close opened files */
1377        f_close(&fnew);
1378
1379
1380        //
1381        // Return success.
1382        //
1383        return(0);
1384
1385    }
1386
1387
1388    //*****************************************************************************
1389    //
1390    // This function implements the "add_to_file" command.  It appends new input to
1391    // an existing file, selected by its filename. If file not exists, file is being
    created.
1392    //
1393    //*****************************************************************************
1394
1395    int add_to_file(const char *filename, const char *write_Buffer){
1396
1397
1398        FIL fnew;      /* new file object */
1399        FRESULT fresult;
1400        unsigned short bw = 0;
1401
1402
1403        //
1404        // First, check to make sure that the current path (CWD), plus
1405        // the file name, plus a separator and trailing null, will all
1406        // fit in the temporary buffer that will be used to hold the
1407        // file name.  The file name must be fully specified, with path,
1408        // to FatFs.
1409        //
1410        if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1411        {
1412            UARTprintf("Resulting path name is too long\n");
1413            return(0);
```

```
1414        }
1415
1416        //
1417        // Copy the current path to the temporary buffer so it can be manipulated.
1418        //
1419        strcpy(g_cTmpBuf, g_cCwdBuf);
1420
1421        //
1422        // If not already at the root level, then append a separator.
1423        //
1424        if(strcmp("/", g_cCwdBuf))
1425        {
1426            strcat(g_cTmpBuf, "/");
1427        }
1428
1429        //
1430        // Now finally, append the file name to result in a fully specified file.
1431        //
1432        strcat(g_cTmpBuf, filename);
1433
1434        //
1435        // Wait for the UART transmit buffer to empty.
1436        //
1437        #if defined(UART_BUFFERED)
1438            UARTFlushTx(false);
1439        #endif
1440
1441
1442        //
1443        // Open/Create File
1444        //
1445
1446    fresult = f_open(&fnew, g_cTmpBuf, FA_OPEN_ALWAYS | FA_WRITE |FA_READ);
1447
1448    if(fresult != FR_OK)
1449    {
1450        return(fresult);
1451    }
1452
1453    // Set Pointer to end of File
1454    f_lseek(&fnew,fnew.fsize);
1455
1456    // Write String to end of File
1457    fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1458    if(fresult != FR_OK)
1459    {
1460        return(fresult);
1461    }
1462
1463    /* Close opened files */
1464    f_close(&fnew);
1465
1466
1467    //
1468    // Return success.
1469    //
1470    return(0);
1471
1472
```

```
1473  }
1474
1475  //*****************************************************************************
1476  //
1477  // This function implements the "create_file" command.  It creates a file on
1478  // the current working directory with specified filename.
1479  //
1480  //*****************************************************************************
1481
1482  int create_file(const char *filename)
1483  {
1484
1485
1486      FIL fnew;      /* new file object */
1487      FRESULT fresult;
1488
1489      //
1490      // First, check to make sure that the current path (CWD), plus
1491      // the file name, plus a separator and trailing null, will all
1492      // fit in the temporary buffer that will be used to hold the
1493      // file name.  The file name must be fully specified, with path,
1494      // to FatFs.
1495      //
1496      if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1497      {
1498          UARTprintf("Resulting path name is too long\n");
1499          return(0);
1500      }
1501
1502      //
1503      // Copy the current path to the temporary buffer so it can be manipulated.
1504      //
1505      strcpy(g_cTmpBuf, g_cCwdBuf);
1506
1507      //
1508      // If not already at the root level, then append a separator.
1509      //
1510      if(strcmp("/", g_cCwdBuf))
1511      {
1512          strcat(g_cTmpBuf, "/");
1513      }
1514
1515      //
1516      // Now finally, append the file name to result in a fully specified file.
1517      //
1518      strcat(g_cTmpBuf, filename);
1519
1520          //
1521          // Wait for the UART transmit buffer to empty.
1522          //
1523          #if defined(UART_BUFFERED)
1524              UARTFlushTx(false);
1525          #endif
1526
1527
1528  //
1529  // Create the File
1530  //
1531    fresult = f_open(&fnew, filename, FA_CREATE_ALWAYS | FA_WRITE );
```

```
1532
1533      //
1534      // Check if creation succeeded
1535      //
1536
1537      if(fresult != FR_OK)
1538      {
1539          return(fresult);
1540      }
1541
1542      /* Close opened files */
1543      f_close(&fnew);
1544
1545
1546         //
1547         // Return success.
1548         //
1549         return(0);
1550   }
1551
1552
1553
1554
1555   //*****************************************************************************
1556   //
1557   // Own variation of sdcard.c main function from stellarisware to implement an
1558   // command line based file explorer for browsing the SD Card
1559   //
1560   //*****************************************************************************
1561
1562   int my_start_cmd_line(int argc, char *argv[])
1563   {
1564       // set Command line pointer to the beginning of the SD-Card/Browser command
          structure
1565       g_psCmdTable = &g_sBrowserCmdTable[0];
1566
1567       sprintf(g_cSdLocalBuf, "SD-Card: %s>", g_cCwdBuf);
1568
1569       // set location buffer to Browser
1570       g_cLocalBuf = (char*)g_cSdLocalBuf;
1571
1572       return(0);
1573   }
1574
1575   //*****************************************************************************
1576   //
1577   // Main function for measurement of voltages, current and temperature. Multiplex
1578   // all relevant cells (depended on configuration) and write data to LogFile on
1579   // SD-Card.
1580   //
1581   //*****************************************************************************
1582   void do_measure(){
1583
1584   //*****************************************************************************
1585   // Use this commented version if current is needed measured with every cellvoltage
1586   //*****************************************************************************
1587   //  unsigned int i = 0;
1588   //  char buf[64];
1589   //  //UARTprintf("\n");
```

```
1590  //  for (i=1; i <= config.quantity_cells; i++){
1591  //      sprintf(buf,
      "%02d,%04d,%02d,%02d,%02d,%02d,%02d,uV,%07d,C,%03d,mA,%07d,%01d\n"
1592  //                          , i, clock_year, clock_month, clock_day, clock_hour,
      clock_min, clock_sec
1593  //                          ,adc_get_voltage(i), get_temperature(i),
      adc_get_current(), power_relay_active);
1594  //
1595  //      //UARTprintf(buf);
1596  //      add_to_file(MEAS_FILE, buf);
1597  //
1598  ////        if(control_connection){
1599  ////            telnet_write(buf);
1600  ////        }
1601  //
1602  //  }
1603  //*****************************************************************************
1604
1605      int i = 0;
1606      char buf[64];
1607
1608      UARTprintf("\n");
1609      //Turn on Measure-LED
1610      GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  GPIO_PIN_5);
1611
1612      //reset serial interface of the ADC
1613      spiChipSelect(ADC);
1614      SSIDataPut(SSI1_BASE, 0xFF);
1615      SSIDataPut(SSI1_BASE, 0xFF);
1616      SSIDataPut(SSI1_BASE, 0xFF);
1617      SSIDataPut(SSI1_BASE, 0xFF);
1618      spiChipSelect(NONE);
1619
1620      adc_get_current();//measure current and load into global variable
1621
1622      if(cycle_active){//check current limt if cycling is active
1623          if(abs(g_iCurrent) > config.max_current){//Check if current doesnt exceed
          limit
1624              //Stop cycling immediately but continue measurement
1625              stop_cycling();
1626
1627              handlerState = ERRCURR;//Display error message
1628              UARTprintf("\nMAXIMUM CURRENT EXCEEDED!\n");
1629              if(control_connection){
1630                  telnet_write("\nMAXIMUM CURRENT EXCEEDED!\n");
1631              }
1632          } else if(abs(gBat_charge += (g_iCurrent*config.cyclestep)) > config.
          max_charge){//check if charge limit exceeded (mAsec)
1633              //Stop cycling immediately but continue measurement
1634              stop_cycling();
1635
1636              handlerState = ERRCHARGE;//Display error message
1637              UARTprintf("\nMAXIMUM CHARGE EXCEEDED!\n");
1638              if(control_connection){
1639                  telnet_write("\nMAXIMUM CHARGE EXCEEDED!\n");
1640              }
1641          }
1642      }
1643
```

```
1644        //Get all the values
1645        for (i = 1; i <= config.quantity_cells; i++){
1646            if(measurement_active){
1647                //save measurements in global variable
1648                adc_get_voltage(i);
1649                get_temperature(i);
1650
1651                sprintf(buf, "%02d,%04d,%02d,%02d,%02d,%02d,%02d,%07d,%03d,%07d,%01d\n"
1652                                            , i, clock_year, clock_month, clock_day,
                                                clock_hour, clock_min, clock_sec
1653                                            ,g_iCellVoltages[i-1], g_iCellTemperature[i-1
                                                ], g_iCurrent, power_relay_active);
1654
1655                //save the values
1656                add_to_file(MEAS_FILE, buf);
1657                UARTprintf(buf);
1658                if(control_connection){
1659                    telnet_write(buf);
1660                }
1661
1662                if(cycle_active){//check voltage and temperature limits and stop cycling
1663                    if(g_iCellVoltages[i-1] < config.min_voltage || g_iCellVoltages[i-1]
                        > config.max_voltage){//Check for Voltage and Temperature limits
1664                        //Stop cycling immediately continue measurement
1665                        stop_cycling();
1666
1667                        handlerState = ERRVOLT;//Display message
1668                        UARTprintf("\nVOLTAGE LIMITS EXCEEDED!\n");
1669                        if(control_connection){
1670                            telnet_write("\nVOLTAGE LIMITS EXCEEDED!\n");
1671                        }
1672                    }else if(g_iCellTemperature[i-1] > config.max_temp){
1673                        //Stop cycling immediately continue measurement
1674                        stop_cycling();
1675
1676                        handlerState = ERRTEMP;//Display message
1677                        UARTprintf("\nTEMPERATURE LIMIT EXCEEDED!\n");
1678                        if(control_connection){
1679                            telnet_write("\nTEMPERATURE LIMIT EXCEEDED!\n");
1680                        }
1681                    }
1682                }
1683            }
1684        }
1685        //Turn off Measure-LED
1686        GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  0x00);
1687
1688        if(cycle_active)
1689            UARTprintf("Cycling active, step %d of %d running\n", current_cycle_step,
            config.step_quantity);
1690        else
1691            UARTprintf("Cycling not active\n");
1692    }
1693
1694
1695
```

```
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      {
16          ADC,
17          SD,
18          NONE
19      }
20  spi_cs_t;
21
22  /*****************************************************************************
23  *
24  * Function Declarations
25  *
26  *****************************************************************************/
27  void ssi1_init(void);
28  void spiChipSelect(spi_cs_t chip);
29
30  #endif /* MYSSI_H_ */
31
```

```
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                                       Stellaris Ware
10
11  last modified:                      2013/05/26
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 ssi1 initalization for use with
    SD Card and 16-bit ADC
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 "header/myssi.h"
50  #include "third_party/fatfs/src/diskio.h"
51
52
53
54
55  //*****************************************************************************
56  //
57  // Initialize SSI1-Interface
58  //
```

```
59  //*****************************************************************************
60  void ssi1_init(void)
61  {
62      // Output
63      UARTprintf("Initializing SS1...");
64      // enable SSI1
65      SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);
66      // enable GPIO Port F
67      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
68      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
69      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
70      // Den Takt auf 1MHz stellen (hier ist noch Luft nach oben)
71      SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3,
        SSI_MODE_MASTER, 2000000, 8);  // ADC=SSI_FRF_MOTO_MODE_3
72      // Pin Configuration
73      GPIOPinConfigure(GPIO_PE0_SSI1CLK);
74      GPIOPinConfigure(GPIO_PF4_SSI1RX);
75      GPIOPinConfigure(GPIO_PF5_SSI1TX);
76      GPIOPinTypeSSI(GPIO_PORTF_BASE, GPIO_PIN_4 | GPIO_PIN_5);
77      GPIOPinTypeSSI(GPIO_PORTE_BASE, GPIO_PIN_0);
78      // Einschalten des SPI Interfaces
79      SSIEnable(SSI1_BASE);
80
81      //Configure Pins for Chipselect
82      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3); // set PA3 as digital output
        "SS1_CS_SD"
83      GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1); // set PE1 as digital output
        "SS1_CS_ADC"
84
85      ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3);  //deselect SS1_CS_SD
86      ROM_GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);  //deselect SS1_CS_ADC
87
88
89      // Output
90      UARTprintf("done.\n");
91  }
92
93  /**
94   * Steuerung der SPI Chip Selects fuer ADC und SD
95  */
96  void spiChipSelect(spi_cs_t chip){
97
98
99      switch (chip) {
100         case ADC:
101
102             GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1); // set PE1 as
                digital output "SS1_CS_ADC"
103
104             SSIDisable(SSI1_BASE);
105
106             // Den Takt auf 4MHz stellen (hier ist noch Luft nach oben)
107             SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_3,
                SSI_MODE_MASTER, 4000000, 8);  // ADC=SSI_FRF_MOTO_MODE_3
108
109             SSIEnable(SSI1_BASE);
110
111
112             ROM_GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0x00);    //select
```

```
                SS1_CS_ADC
113
114
115             break;
116
117
118         case SD:
119
120             //SD-Card automaticly sets CS in own sourcefiles!
121
122             break;
123
124         case NONE:
125
126             GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);  //deselect
                SS1_CS_ADC
127
128             SSIDisable(SSI1_BASE);
129
130             // Den Takt auf 12,5MHz stellen (max)
131             SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0,
                SSI_MODE_MASTER, 12500000, 8);  // ADC=SSI_FRF_MOTO_MODE_3
132
133             SSIEnable(SSI1_BASE);
134
135             break;
136     }
137 }
138
139
140
141
```

```
1  /*
2   * myuart.h
3   *
4   *  Created on: 17.01.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef MYUART_H_
9  #define MYUART_H_
10
11
12 /*****************************************************************************
13 *
14 * Function Declarations
15 *
16 *****************************************************************************/
17 void myUARTinit(void);
18 void UART0IntHandler(void);
19 int myUARTgets(char *pcBuf, unsigned long ulLen);
20
21
22 #endif /* MYUART_H_ */
23
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               myuart.c
4
5   Auhtor:                             Thomas Wisnewski
6   Credits:                            Matthias Schneider
7                                       Tobias Steinmann
8                                       Fabian Schwartau
9                                       Stellaris Ware
10
11  last modified:                      2013/05/26
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 UART0 Initialization for use with
24                                      UARTStdio.h and UARTprintf()
25
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
51
52  //*****************************************************************************
53  //
54  // Defines the size of the buffer that holds the command line.
55  //
56  //*****************************************************************************
57  #define UART_BUF_SIZE    64
58
59  //*****************************************************************************
```

```
60  // Boolean for fully received uart commands
61  //*****************************************************************************
62  tBoolean received = false;
63
64  //*****************************************************************************
65  //
66  // The buffer that holds the command line.
67  //
68  //*****************************************************************************
69  char g_cUartBuf[UART_BUF_SIZE];
70
71  //*****************************************************************************
72  // index for g_cUartBuf[] loop
73  //*****************************************************************************
74  int i = 0;
75
76  //*****************************************************************************
77  //
78  // Initialize UART0
79  //
80  //*****************************************************************************
81  void myUARTinit(void)
82  {
83      //Enable Peripheral
84      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
85      GPIOPinConfigure(GPIO_PA0_U0RX);
86      GPIOPinConfigure(GPIO_PA1_U0TX);
87      ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
88      UARTStdioInit(0);
89
90
91      //Enable Interrupts
92      ROM_UARTEnable(UART0_BASE);
93      ROM_IntEnable(INT_UART0);
94      ROM_UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
95
96
97      UARTprintf("\n\nInitializing UART0...done.\n");
98  }
99
100
101 //*****************************************************************************
102 //
103 // The UART interrupt handler.
104 //
105 //*****************************************************************************
106
107 void UART0IntHandler(void)
108 {
109     unsigned long ulStatus;
110     long c;
111
112     //
113     // Get the interrupt status.
114     //
115     ulStatus = UARTIntStatus(UART0_BASE, true);
116
117     //
118     // Clear the asserted interrupts.
```

```
119         //
120         UARTIntClear(UART0_BASE, ulStatus);
121
122         //
123         // Loop while there are characters in the receive FIFO.
124         //
125         while(UARTCharsAvail(UART0_BASE)){
126             c = UARTCharGetNonBlocking(UART0_BASE);
127
128             if(c != -1){
129                 if(c == 10){
130                     if(g_cUartBuf[i-1] == 13){
131                         g_cUartBuf[i - 1] = 0;
132                     }
133                     received = true;
134                     i = 0;
135                     break;
136                 }
137                 else{
138                 g_cUartBuf[i] = (char) c;
139                 i++;
140                 }
141             }
142         }
143
144         //
145         // If command fully received, interpret it and execute. Reset for next command.
146         //
147         if(received){
148             Cmd_interprete(g_cUartBuf);
149             received = false;
150             memset(g_cUartBuf,0,UART_BUF_SIZE);
151         }
152
153     }
154
155
```

```
1   /*
2    * relais.h
3    *
4    *  Created on: 15.07.2012
5    *      Author: Thomas W.
6    */
7
8   #ifndef RELAIS_H_
9   #define RELAIS_H_
10
11  /*****************************************************************************
12  *
13  * Function Declarations
14  *
15  *****************************************************************************/
16  void init_relais(void);
17  void switch_relais(int choose);
18  int switch_power_relais(int choose);
19  int Cmd_relais(int argc, char *argv[]);
20
21
22
23  #endif /* RELAIS_H_ */
24
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               relais.c
4
5   Auhtor:                             Thomas Wisnewski
6   Credits:                            Stellaris Ware
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:                        Source Code for controlling installed relais
21
22  */
23
24  /*****************************************************************************
25  *
26  *   Includings
27  *
28  *****************************************************************************/
29  #include <string.h>
30  #include <stdarg.h>
31  #include <stdio.h>
32  #include <stdlib.h>
33  #include "inc/hw_memmap.h"
34  #include "inc/hw_types.h"
35  #include "utils/uartstdio.h"
36  #include "utils/ustdlib.h"
37  #include "inc/hw_ints.h"
38  #include "inc/hw_types.h"
39  #include "inc/hw_uart.h"
40  #include <inc/hw_ssi.h>
41  #include "inc/hw_gpio.h"
42  #include "driverlib/debug.h"
43  #include "driverlib/interrupt.h"
44  #include "driverlib/rom.h"
45  #include "driverlib/rom_map.h"
46  #include "driverlib/sysctl.h"
47  #include "driverlib/uart.h"
48  #include <driverlib/ssi.h>
49  #include <driverlib/gpio.h>
50  #include <driverlib/sysctl.h>
51
52  /*****************************************************************************
53  *
54  *   Own Includings
55  *
56  *****************************************************************************/
57  #include "header/relais.h"
58  #include "header/control.h"
59  #include "header/config.h"
```

```
60
61  extern volatile unsigned int power_relay_active;
62
63  // TCP-Socket für die aktuelle Steuerungs-Verbindung
64  extern struct tcp_pcb* control_connection;
65
66  //*****************************************************************************
67  //
68  // Defines the size of the buffer that holds the input data.
69  //
70  //*****************************************************************************
71  #define RELAIS_INPUT_DATA_SIZE    64
72
73  //*****************************************************************************
74  //
75  // A temporary data buffer used for input data
76  //
77  //*****************************************************************************
78  static char g_cTmpInpData[RELAIS_INPUT_DATA_SIZE];
79
80
81  //*****************************************************************************
82  //
83  // Initialize GPIO ports for relay-mux and powerrelays
84  //
85  //*****************************************************************************
86  void init_relais(void)
87  {
88
89      UARTprintf("Initializing Relais...");
90
91      //Enable Peripheral for Relais-Mux
92      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);   // enable peripheral
93      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);   // enable peripheral
94
95      //Enable Peripheral for Relais on power section
96      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);   // enable peripheral
97      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);   // enable peripheral
98
99  //  //Unlock PB7 for changing from NMI-Mode
100 //
101 //  //
102 //  // Unlock access to the commit register
103 //  //
104 //  HWREG(GPIO_PORTB_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY_DD;
105 //
106 //  //
107 //  // Set the commit register for PB7 to allow changing the function
108 //  //
109 //  HWREG(GPIO_PORTB_BASE + GPIO_O_CR) = 0x80;
110 //
111 //  //
112 //  // Enable the alternate function for PB7 (NMI)
113 //  //
114 //  HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) |= 0x80;
115 //
116 //  //
117 //  // Turn on the digital enable for PB7
118 //  //
```

```
119 //  HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= 0x80;
120
121
122     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 ); // set PB2 and
        PB3 as digital output
123     GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7); //
        set PH3, PH4 and PH7 as digital output
124     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7); //
        set PC5, PC6 and PC7 as digital output
125     GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_7); // set PG7 as digital output
126
127     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3, 0x00); // set PB2 and PB3
        to logic "0"
128     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7, 0x00); //
        set PH3, PH4 and PH7 to logic "0"
129
130     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0x00); //
        set PC5, PC6 and PC7 to logic "0"
131     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7,  GPIO_PIN_7); // set PG7 to logic "1"
        "Strobe"
132
133     //Switch off all relays
134     switch_relais(0);
135     switch_power_relais(0);
136
137     UARTprintf("done\n");
138 }
139
140 //*****************************************************************************
141 //
142 // Function for switching relays-mux
143 //
144 //*****************************************************************************
145 void switch_relais(int choose)
146 {
147
148         GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PB7 (Enable)
            to logic "1"
149
150
151         switch(choose){
152
153         case 0:
154
155             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
156             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
157             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
158             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
159
160         break;
161
162
163
164         case 1:
165
166             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
167             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
168             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
169             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
```

```
170             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
171
172         break;
173
174         case 2:
175
176             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
177             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
178             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
179             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
180             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
181
182         break;
183
184         case 3:
185
186             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
187             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
188             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
189             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
190             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
191
192         break;
193
194         case 4:
195
196             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
197             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
198             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
199             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
200             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
201
202         break;
203
204         case 5:
205
206             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
207             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
208             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
209             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
210             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
211
212         break;
213
214         case 6:
215
216             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
217             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
218             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
219             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
220             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
221
222         break;
```

```
223
224             case 7:
225
226                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
227                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
228                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
229                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
230                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                    logic "0"
231
232             break;
233
234             case 8:
235
236                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
237                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
238                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
239                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
240                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                    logic "0"
241
242             break;
243
244             case 9:
245
246                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
247                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
248                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
249                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
250                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                    logic "0"
251
252             break;
253
254             case 10:
255
256                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
257                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
258                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
259                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
260                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                    logic "0"
261
262             break;
263
264             case 11:
265
266                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
267                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
268                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
269                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
270                 GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                    logic "0"
271
272             break;
273
274             case 12:
275
276                 GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
```

```
277             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
278             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
279             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
280             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to
                logic "0"
281
282         break;
283
284         default:
285
286             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
287             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
288             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
289             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
290
291
292         break;
293     }
294
295
296
297
298 }
299
300
301 //*****************************************************************************
302 //
303 // Function for switching powerrelay
304 //
305 //*****************************************************************************
306 int switch_power_relais(int choose)
307 {
308     int active;
309     unsigned long ul_delay_count;
310
311         switch(choose){
312
313         case 0:
314
315             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
316             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
317             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
318             active = 0;
319
320         break;
321
322         case 1:
323
324             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
325             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
326             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
327             active = 1;
328         break;
329
330         case 2:
331
332             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
333             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
334             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
```

```
335             active = 2;
336         break;
337
338         case 3:
339
340             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
341             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
342             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
343             active = 3;
344         break;
345
346         case 4:
347
348             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, GPIO_PIN_5);
349             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
350             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
351             active = 4;
352         break;
353
354         default:
355
356             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
357             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
358             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
359             active = 0;
360         break;
361
362         }
363
364         GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, 0x00); // set PG7 (Strobe) to
         logic "0"
365         // ca. 10 ms delay
366         ul_delay_count = 100000;
367         while (ul_delay_count) ul_delay_count--;
368         //--------------------------------
369         GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PG7 (Strobe)
         to logic "1"
370         // ca. 10 ms delay
371         ul_delay_count = 100000;
372         while (ul_delay_count) ul_delay_count--;
373         //--------------------------------
374
375         if(active)//switch the current active LED
376             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  GPIO_PIN_7);
377         else
378             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  0);
379
380         return active;
381 }
382
383 //*****************************************************************************
384 //
385 // This function implements the "Relais" command. It activates ONE chosen power relais
386 //
387 //*****************************************************************************
388 int
389 Cmd_relais(int argc, char *argv[])
390 {
391     char buf[32];
```

```
392
393       if(argc == 2){
394           int relais;
395
396           //
397           // Copy the first input data into buffer.
398           //
399           strcpy(g_cTmpInpData, argv[1]);
400
401           relais = (int) strtol(g_cTmpInpData, (char **)NULL, 10);
402
403           if(relais >= 0 && relais <= 4){
404
405               power_relay_active = switch_power_relais(relais);
406
407               UARTprintf("\nRelais %02d activated", power_relay_active);
408               if(control_connection){
409                   sprintf(buf, "Relais %02d activated\n", power_relay_active);
410                   telnet_write(buf);
411               }
412
413               //
414               // Return success.
415               //
416               return 0;
417           }
418       }
419       return -1;//Bad command
420   }
421
```

```
1  /*
2   * rtc.h
3   *
4   *  Created on: 03.06.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef RTC_H_
9  #define RTC_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void rtc_init(void);
17 void set_clock_values(void);
18 void set_rtc_values(unsigned int year, unsigned int month, unsigned int day, unsigned
    int hour, unsigned int min, unsigned int sec );
19
20 #endif /* RTC_H_ */
21
```

```
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     }
275
276
277
```

```
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
97  extern void GPIOD01Handler(void);
98  extern void GPIOF00Handler(void);
99  extern void GPIOC04Handler(void);
100 extern void GPIOH05Handler(void);
101 extern void watchdog_int_handler(void);
102
103 //*****************************************************************************
104 //
105 // The vector table.  Note that the proper constructs must be placed on this to
106 // ensure that it ends up at physical address 0x0000.0000 or at the start of
107 // the program if located at a start address other than 0.
108 //
109 //*****************************************************************************
110 #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
111 void (* const g_pfnVectors[])(void) =
112 {
113     (void (*)(void))((unsigned long)&__STACK_TOP),
114                                             // The initial stack pointer
115     ResetISR,                               // The reset handler
116     NmiSR,                                  // The NMI handler
117     FaultISR,                               // The hard fault handler
```

```
118       IntDefaultHandler,                      // The MPU fault handler
119       IntDefaultHandler,                      // The bus fault handler
120       IntDefaultHandler,                      // The usage fault handler
121       0,                                      // Reserved
122       0,                                      // Reserved
123       0,                                      // Reserved
124       0,                                      // Reserved
125       IntDefaultHandler,                      // SVCall handler
126       IntDefaultHandler,                      // Debug monitor handler
127       0,                                      // Reserved
128       IntDefaultHandler,                      // The PendSV handler
129       SysTickHandler,                         // The SysTick handler
130       IntDefaultHandler,                      // GPIO Port A
131       IntDefaultHandler,                      // GPIO Port B
132       GPIOC04Handler,                         // GPIO Port C
133       GPIOD01Handler,                         // GPIO Port D
134       IntDefaultHandler,                      // GPIO Port E
135       UART0IntHandler,                        // UART0 Rx and Tx
136       //UARTStdioIntHandler,                  // UART0 Rx and Tx
137       IntDefaultHandler,                      // UART1 Rx and Tx
138       IntDefaultHandler,                      // SSI0 Rx and Tx
139       IntDefaultHandler,                      // I2C0 Master and Slave
140       IntDefaultHandler,                      // PWM Fault
141       IntDefaultHandler,                      // PWM Generator 0
142       IntDefaultHandler,                      // PWM Generator 1
143       IntDefaultHandler,                      // PWM Generator 2
144       IntDefaultHandler,                      // Quadrature Encoder 0
145       ADC0Handler,                            // ADC Sequence 0
146       IntDefaultHandler,                      // ADC Sequence 1
147       IntDefaultHandler,                      // ADC Sequence 2
148       IntDefaultHandler,                      // ADC Sequence 3
149       //IntDefaultHandler,                      // Watchdog timer
150       watchdog_int_handler,                   // Watchdog timer
151       Timer0IntHandler,                       // Timer 0 subtimer A
152       IntDefaultHandler,                      // Timer 0 subtimer B
153       Timer1IntHandler,                       // Timer 1 subtimer A
154       IntDefaultHandler,                      // Timer 1 subtimer B
155       Timer2IntHandler,                       // Timer 2 subtimer A
156       IntDefaultHandler,                      // Timer 2 subtimer B
157       IntDefaultHandler,                      // Analog Comparator 0
158       IntDefaultHandler,                      // Analog Comparator 1
159       IntDefaultHandler,                      // Analog Comparator 2
160       IntDefaultHandler,                      // System Control (PLL, OSC, BO)
161       IntDefaultHandler,                      // FLASH Control
162       GPIOF00Handler,                         // GPIO Port F
163       IntDefaultHandler,                      // GPIO Port G
164       GPIOH05Handler,                         // GPIO Port H
165       IntDefaultHandler,                      // UART2 Rx and Tx
166       IntDefaultHandler,                      // SSI1 Rx and Tx
167       Timer3IntHandler,                       // Timer 3 subtimer A
168       IntDefaultHandler,                      // Timer 3 subtimer B
169       IntDefaultHandler,                      // I2C1 Master and Slave
170       IntDefaultHandler,                      // Quadrature Encoder 1
171       IntDefaultHandler,                      // CAN0
172       IntDefaultHandler,                      // CAN1
173       IntDefaultHandler,                      // CAN2
174       lwIPEthernetIntHandler,                 // Ethernet
175       IntDefaultHandler,                      // Hibernate
176       IntDefaultHandler,                      // USB0
```

```
177       IntDefaultHandler,                      // PWM Generator 3
178       IntDefaultHandler,                      // uDMA Software Transfer
179       IntDefaultHandler,                      // uDMA Error
180       IntDefaultHandler,                      // ADC1 Sequence 0
181       ADC1Handler,                            // ADC1 Sequence 1
182       IntDefaultHandler,                      // ADC1 Sequence 2
183       IntDefaultHandler,                      // ADC1 Sequence 3
184       IntDefaultHandler,                      // I2S0
185       IntDefaultHandler,                      // External Bus Interface 0
186       IntDefaultHandler                       // GPIO Port J
187   };
188
189   //*****************************************************************************
190   //
191   // This is the code that gets called when the processor first starts execution
192   // following a reset event.  Only the absolutely necessary set is performed,
193   // after which the application supplied entry() routine is called.  Any fancy
194   // actions (such as making decisions based on the reset cause register, and
195   // resetting the bits in that register) are left solely in the hands of the
196   // application.
197   //
198   //*****************************************************************************
199   void
200   ResetISR(void)
201   {
202       //
203       // Jump to the CCS C initialization routine.
204       //
205       __asm("    .global _c_int00\n"
206             "    b.w     _c_int00");
207   }
208
209   //*****************************************************************************
210   //
211   // This is the code that gets called when the processor receives a NMI.  This
212   // simply enters an infinite loop, preserving the system state for examination
213   // by a debugger.
214   //
215   //*****************************************************************************
216   static void
217   NmiSR(void)
218   {
219       //
220       // Enter an infinite loop.
221       //
222       while(1)
223       {
224       }
225   }
226
227   //*****************************************************************************
228   //
229   // This is the code that gets called when the processor receives a fault
230   // interrupt.  This simply enters an infinite loop, preserving the system state
231   // for examination by a debugger.
232   //
233   //*****************************************************************************
234   static void
235   FaultISR(void)
```

```
236 {
237     unsigned long ul_delay_count;
238     GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);
239 
240     Cmd_stop(0,0);
241     //
242     // Enter an infinite loop.
243     //
244     while(1)
245     {
246 
247         GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_6, ~GPIOPinRead(GPIO_PORTJ_BASE,
        GPIO_PIN_6));
248 
249         // ca. 0,5 s delay
250         ul_delay_count = 5000000;
251         while (ul_delay_count) ul_delay_count--;
252         //--------------------------------
253 
254     }
255 }
256 
257 //*****************************************************************************
258 //
259 // This is the code that gets called when the processor receives an unexpected
260 // interrupt.  This simply enters an infinite loop, preserving the system state
261 // for examination by a debugger.
262 //
263 //*****************************************************************************
264 static void
265 IntDefaultHandler(void)
266 {
267     //
268     // Go into an infinite loop.
269     //
270     while(1)
271     {
272     }
273 }
274 
```

```
1  /*
2   * temperature.h
3   *
4   *  Created on: 06.03.2013
5   *      Author: Thomas W.
6   */
7
8  #ifndef TEMPERATURE_H_
9  #define TEMPERATURE_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void init_temperature(void);
17 int get_temperature(int chose);
18
19
20 #endif /* TEMPERATURE_H_ */
21
```

```
1   /*
2   Project:                            battery_cycling_sw_v4
3   File:                               main.c
4
5   Auhtor:                             Thomas Wisniewski
6   Credits:                            Matthias Schneider
7                                       Tobias Steinmann
8                                       Fabian Schwartau
9                                       Stellaris Ware
10
11  last modified:                      2013/05/28
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:                        Code for initializing intern ADCs for
    temperature measurement
24  */
25
26  /*****************************************************************************
27  *
28  *   Includings
29  *
30  *****************************************************************************/
31  #include "driverlib/rom.h"
32  #include "third_party/fatfs/src/ff.h"
33  #include "third_party/fatfs/src/diskio.h"
34  #include "inc/hw_ints.h"
35  #include "inc/hw_memmap.h"
36  #include "inc/hw_types.h"
37  #include "driverlib/adc.h"
38  #include "driverlib/debug.h"
39  #include "driverlib/gpio.h"
40  #include "driverlib/interrupt.h"
41  #include "driverlib/pin_map.h"
42  #include "driverlib/rom.h"
43  #include "driverlib/sysctl.h"
44  #include "driverlib/timer.h"
45  #include <stdio.h>
46  #include "utils/uartstdio.h"
47  #include "math.h"
48
49
50
51
52  //*****************************************************************************
53  //
54  // A global data buffer used for actual Cell temperature
55  //
56  //*****************************************************************************
57  volatile int g_iCellTemperature[12] = {0};
58
```

```
59  //Buffer for ADC-Samples
60  unsigned long g_pulData0[8];
61  unsigned long g_pulData1[4];
62
63  /*****************************************************************************
64  *
65  *   Initialize internal ADCs for Temperature sensors
66  *
67  *****************************************************************************/
68  void init_temperature(){
69
70      UARTprintf("Initializing internal ADC...");
71
72      //enable Peripheral
73      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
74      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC1);
75
76      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
77      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
78      ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
79
80      ROM_GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
        GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);
81      ROM_GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
        GPIO_PIN_7);
82      ROM_GPIOPinTypeADC(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_5);
83
84      // Set Speed to 500.000 Samples per Second (up to 1MSPS)
85      ROM_SysCtlADCSpeedSet(SYSCTL_ADCSPEED_500KSPS);
86
87      //
88      // Configure the ADCs.
89      //
90      ADCSequenceConfigure(ADC0_BASE, 0, ADC_TRIGGER_TIMER, 0);
91      ADCSequenceStepConfigure(ADC0_BASE, 0, 0, ADC_CTL_CH0);
92      ADCSequenceStepConfigure(ADC0_BASE, 0, 1, ADC_CTL_CH1);
93      ADCSequenceStepConfigure(ADC0_BASE, 0, 2, ADC_CTL_CH2);
94      ADCSequenceStepConfigure(ADC0_BASE, 0, 3, ADC_CTL_CH3);
95      ADCSequenceStepConfigure(ADC0_BASE, 0, 4, ADC_CTL_CH8);
96      ADCSequenceStepConfigure(ADC0_BASE, 0, 5, ADC_CTL_CH9);
97      ADCSequenceStepConfigure(ADC0_BASE, 0, 6, ADC_CTL_CH11);
98      ADCSequenceStepConfigure(ADC0_BASE, 0, 7, ADC_CTL_CH10 | ADC_CTL_IE | ADC_CTL_END
        );
99
100     ADCSequenceConfigure(ADC1_BASE, 1, ADC_TRIGGER_TIMER, 1);
101     ADCSequenceStepConfigure(ADC1_BASE, 1, 0, ADC_CTL_CH4);
102     ADCSequenceStepConfigure(ADC1_BASE, 1, 1, ADC_CTL_CH5);
103     ADCSequenceStepConfigure(ADC1_BASE, 1, 2, ADC_CTL_CH6);
104     ADCSequenceStepConfigure(ADC1_BASE, 1, 3, ADC_CTL_CH7 | ADC_CTL_IE | ADC_CTL_END);
105
106     ADCSequenceEnable(ADC0_BASE, 0);
107     ADCSequenceEnable(ADC1_BASE, 1);
108
109     //
110     // Clear out the FIFO (not really important for this exercise)
111     //
112     ROM_ADCSequenceDataGet(ADC0_BASE, 0, g_pulData0);
113     ROM_ADCSequenceDataGet(ADC1_BASE, 1, g_pulData1);
114
```

```
115       //
116       // Clear the interrupt status (again, not too important)
117       //
118       ROM_ADCIntClear(ADC0_BASE, 0);
119       ROM_ADCIntClear(ADC1_BASE, 1);
120
121       //
122       // Allow the ADC sequencer in both ADCs to generate interrupts
123       // for SS0
124       //
125       ROM_ADCIntEnable(ADC0_BASE, 0);
126       ROM_ADCIntEnable(ADC1_BASE, 1);
127
128       //
129       // Get the NVIC to generate the appropriate interrupts for the
130       // interrupt handlers as spec'd in the vector table (see startup_*.c)
131       //
132       ROM_IntEnable(INT_ADC0SS0);
133       ROM_IntEnable(INT_ADC1SS1);
134
135       UARTprintf("done\n");
136
137   }
138
139
140   /*****************************************************************************
141   *
142   *   Get the temperature from global buffer
143   *
144   *****************************************************************************/
145   int get_temperature(int choose)
146   {
147           int temperature;
148
149           if((choose >= 0) && (choose <= 8)){
150               temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData0[
                  choose - 1] / 10000 - 1 ) * 298) - 273) * 10;
151           }
152
153           if((choose >= 9) && (choose <= 12)){
154               temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData1[
                  choose - 9] / 10000 - 1 ) * 298) - 273) * 10;
155           }
156
157           if(temperature == 2147483647){
158               g_iCellTemperature[choose-1] = 0;
159               temperature = 0;
160           }else
161               g_iCellTemperature[choose-1] = temperature;
162
163           return temperature;
164   }
165
166
167   //
168   // The ADC unit 0 interrupt hander
169   //
170   void
171   ADC0Handler(void)
```

```
172 {
173     ROM_ADCIntClear(ADC0_BASE, 0);
174     //write adc-values to global buffer
175     ROM_ADCSequenceDataGet(ADC0_BASE, 0, g_pulData0);
176 }
177
178
179 //
180 // The ADC unit 1 interrupt handler
181 //
182 void
183 ADC1Handler(void)
184 {
185     ROM_ADCIntClear(ADC1_BASE, 1);
186     //write adc-values to global buffer
187     ROM_ADCSequenceDataGet(ADC1_BASE, 1, g_pulData1);
188 }
189
190
191
```

```
1  /*
2   * watchdog.h
3   *
4   *  Created on: 17.12.2013
5   *      Author: Johannes
6   */
7
8  #ifndef WATCHDOG_H_
9  #define WATCHDOG_H_
10
11 void watchdog_init(void);
12 void watchdog_int_handler(void);
13
14 #endif /* WATCHDOG_H_ */
15
```

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

Auhtor:                         Johannes Roehn


last modified:                  2013/12/17

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 watchdog and interrupt
handler
*/

/*******************************************************************************
*
* Own Includings
*
*******************************************************************************/
#include "header/watchdog.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "header/relais.h"
#include "utils/uartstdio.h"
#include "header/mysdcard.h"

//*******************************************************************************
//Extern variable for name of file for measurment saving
//*******************************************************************************
extern char MEAS_FILE[128];

extern volatile unsigned int measurement_active;

//*****************************************************************************
//
// Flag to tell the watchdog interrupt handler whether or not to clear the
// interrupt (feed the watchdog).
//
//*****************************************************************************
volatile tBoolean g_bFeedWatchdog = true;



void watchdog_init(void){

    /* Enable the peripherals used by this example. */
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_WDOG0);

```

```
59      //
60      // Enable the watchdog interrupt.
61      //
62      ROM_IntEnable(INT_WATCHDOG);
63
64      /* Set the period of the watchdog timer. */
65      ROM_WatchdogReloadSet(WATCHDOG0_BASE, SysCtlClockGet() * 10);//called every 5
        seconds
66
67      /* Enable reset generation from the watchdog timer. */
68      ROM_WatchdogResetEnable(WATCHDOG0_BASE);
69
70      /* Enable the watchdog timer. */
71      ROM_WatchdogEnable(WATCHDOG0_BASE);
72
73      ROM_WatchdogLock(WATCHDOG0_BASE);//Lock the watchdog registers
74
75      ROM_WatchdogStallEnable(WATCHDOG0_BASE);
76  }
77
78
79  void watchdog_int_handler(void){
80
81      if(g_bFeedWatchdog){//feed the watchdog if variable is set in main loop
82          g_bFeedWatchdog = false;
83          ROM_WatchdogIntClear(WATCHDOG0_BASE);
84      }else{
85          //system will  be restarted, switch off all relays
86          switch_power_relais(0);
87          switch_relais(0);
88          UARTprintf("WATCHDOG FAIL!\nSYSTEM RESET!\n");
89          if(measurement_active)
90              add_to_file(MEAS_FILE, "\nWATCHDOG FAIL!\nSYSTEM RESET!\n");
91      }
92  }
93
```

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, 10. September 2014	
Ort, Datum	Unterschrift