Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Thomas Wisniewski

Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Thomas Wisniewski

Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung

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. Ulf Claussen

Abgegeben am 4. Juli 2013

Thomas Wisniewski

Thema der Bachelorthesis

Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung

Stichworte

ARM-Microcontroller, Messprüfstand, Ethernet, SD-Karte, LED-Display, Programmierschnittstelle, externer ADC, Relaisumschaltmatrix, NTC-Temperatursensoren, RS232, RS485, Lastrelais, HALL-Sensor

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung und den Aufbau eines Zyklierprüfstandes für Batteriezellen und -systeme. Mit dem Prüfstand können bis zu 12 Zellspannungen und -temperaturen sowie der Belastungs- bzw. Ladestrom in einstellbaren Zyklen gemessen werden. Die Speicherung der Messwerte erfolgt auf einer externen SD-Karte. Alternativ können die Messwerte über eine Ethernetschnittstelle transferiert werden. Die Steuerung erfolgt mit einem ARM Cortex M3 Mikrocontroller.

Thomas Wisniewski

Title of the paper

Cycle test machine for battery cells and -systems controlled by an ARM-controller including data management and network connection

Keywords

ARM-Microcontroller, Measurement test station, Ethernet, SD-Card, LED-Display, Programinterface, external ADC, Relay switch matrix, NTC-Temperature sensors, RS232 RS485, Loadrelays, HALL-sensor

Abstract

Inside this report the development and the construction of a cycletestmachine for batteriecells and -systems is described. Up to 12 cellvoltages and -temperatures plus the load- and the chargecurrent respectively can be measured by the test machine in adjustable cycles. Data is stored on an external SD-Card. Alternativly data can be transfered via Ethnernet interface. The Cycletestmachine is controlled by an ARM Cortex M3 microcontroller.

Inhaltsverzeichnis

1. Einführung

1.1. Allgemeines

Im Rahmen des Forschungsprojektes Drahtlose Zellensensoren für Fahrzeugbatterien (BATSEN) an der Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg) werden für das Forschungsvorhaben Schwerpunkte gesetzt, um Fahrzeugbatterien in ihrer Leistung und Lebensdauer zu steigern. Hierzu werden drahtlose Sensoren entwickelt, um in Batteriemodulen eine Überwachung jeder einzelnen Zelle zu gewährleisten. Aus den Messdaten der Sensoren sollen Kenntnisse über den Ladezustand State of Charge (SOC)und der Gesundheitszustand State of Health (SOH) der Batterie ermittelt werden. Diese Kenntnisse sind für den Einsatz von Hochleistungsbatterien in Elektrofahrzeugen für die Reichweiten- sowie Zustandsbestimmung erforderlich.

Innerhalb des Projektes BATSEN werden die entwickelten Zellsensoren kontinuierlich an konventionellen Blei-Säure-Batterien getestet und verifiziert. Für die Zukunft ist es wichtig, diese Sensoren auch an neuartigen Batteriesystemen auf Basis von Lithium-Ionen (LI-ION), wie diese auch zurzeit in der Industrie erforscht werden, zu testen.

1.2. Motivation

Anhand zweier Abschlussarbeiten von Lars Hillermann [11] und Rico Loschwitz [14] wurden Starterbatterien auf Basis von Lithium-Eisenphosphat (LiFePO4) entwickelt und aufgebaut. In ersten Kurzversuchen wurden mit diesen Batterien Startversuche an diversen Fahrzeugen erfolgreich durchgeführt. Allgemein ist jedoch nicht bekannt, wie sich so eine Starterbatterie bei der Stromauf- und Stromentnahme verhält. Diese Bachelorarbeit befasst sich mit der Entwicklung, dem Aufbau und der Erprobung eines Zyklierprüfstandes für Batterien jeglicher Technologie, die aus mehreren Zellen bestehen können. Durch zyklische *(altgriechisch: periodisch wiederkehrende gleichartige, ähnliche oder vergleichbare Ereignisse)* Aufnahmen physikalischer Messdaten soll eine darauffolgende Analyse der Batteriecharakteristik ermöglicht werden.

1.3. Vorarbeiten

Vor Beginn der Bachelorarbeit wurden Vorarbeiten durch meine Person als studentische Hilfskraft geleistet. Mit dieser Tätigkeit wurde eine erste Version des Zyklierprüfstandes realisiert. Grundlage hierzu ist folgende Abb. 1.1 eines Entwurfes.

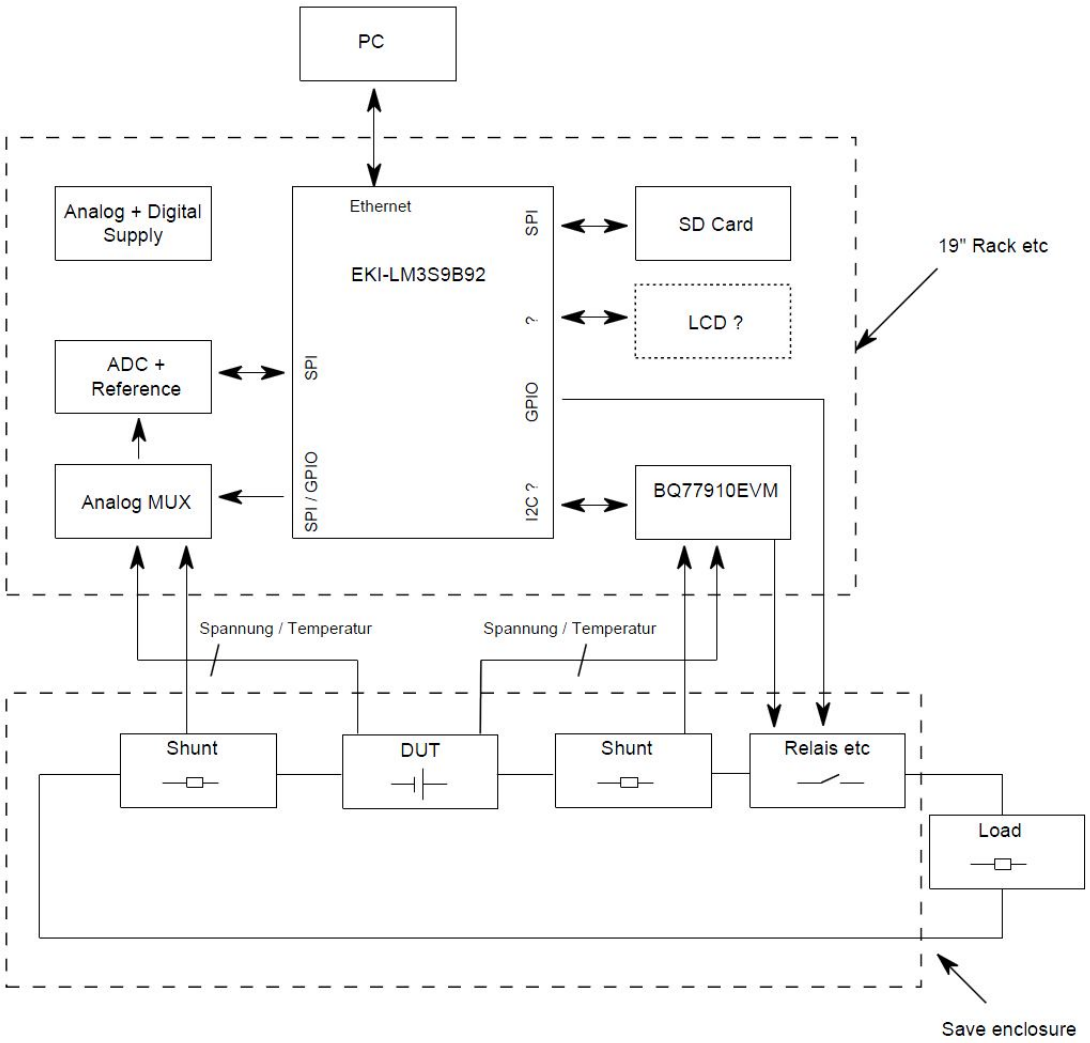


Abbildung 1.1.: Erster Entwurf Zyklierprüfplatz

Der Zyklierprüfplatz besteht aus zwei wesentlichen Komponenten, zum einen der Steuereinheit, zum anderen dem Lastkreis mit Messperipherie. Der Kern der Steuereinheit ist ein Evaluationsboard EKI-LM3S9B92 [27] des Herstellers Texas Instruments (TI). Auf diesem

Evaluationsboard befindet sich ein Mikrontroller der Familie ARM Cortex M3 selbigen Herstellers [28]. Über eine Platinenplattform, auf welche das Evaluationsboard aufgesteckt wird, werden alle Komponenten angebunden. Wesentliche Aufgaben dieses Entwurfes sind:

- Galvanisch getrennte Spannungsmessung einzelner Zellen über einen Analog Digital Converter (ADC)
- Strommessung über einen HALL-Sensor
- Temperaturmessung über NTC-Widerstände
- Speicherung der Messwerte auf einer SD-Karte
- Anbindung über Ethernet für den Datenaustausch mit einem PC
- Steuerung des Lastkreises über Lastrelais

Der Aufbau des ersten Zyklierprüfstandes ist in der Abb. 1.2 zu sehen und wird als Battery Cycling Machine V1.0 (BCMV1) bezeichnet. Einige im Entwurf 1.1 aufgezeigten Komponenten, wie z.B. die optionale Anbindung der Einheit BQ77910EVM [25] oder LC-Displays wurden innerhalb der Vorarbeit nicht realisiert bzw. untersucht. Daher sind in der Abb. 1.1 keine Schnittstellen definiert. Eine erste Inbetriebnahme der BCMV1 war möglich, konnte aber bislang nicht an Batteriezellen in Langzeitversuchen getestet werden.

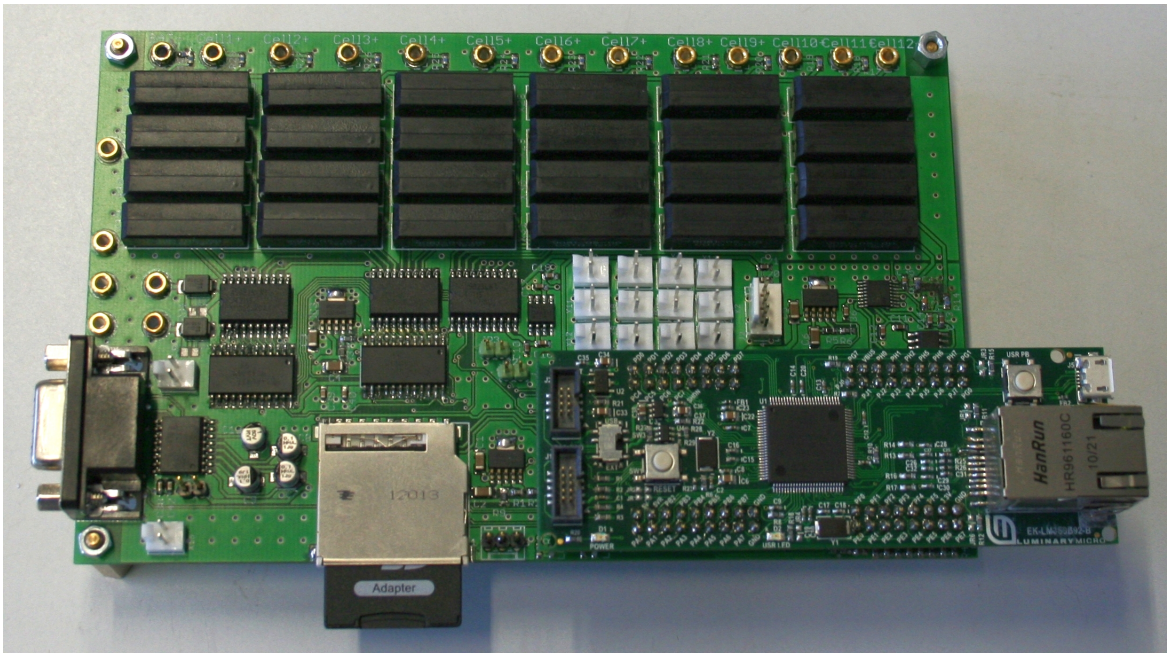


Abbildung 1.2.: Erster Prototyp des Zyklierprüfstandes

1.4. Ziele

Ziele der Bachelorarbeit sind sinngemäß der Aufgabenstellung definiert und können im Anhang A detailliert nachgelesen werden. Hauptziele sind die Neuentwicklung des Zyklierprüfstandes auf Basis des BCMV1. Das Konzept soll dabei erneuert und angepasst werden. Hierbei sollen wesentliche Komponenten übernommen, aber auch Erweiterungen und Änderungen eingefügt werden. Hauptsächliche Ziele und Merkmale der neuen Battery Cycling Machine V2.0 (BCMV2) sind:

- Entwurf einer Hauptplatine mit Mikrocontroller ohne Evaluationsboard
- Integration einer Programmiereinheit direkt im Redesign
- Entwurf einer Relaisumschaltlogik mit zusammengefasster Lösung für Multiplexer und Treiber
- Entwicklung und Anbindung einer Lasttreiberstufe für den Lastkreis
- Aufbau eines Gesamtsystems mit Gehäuse, Frontpanel, Schrank, Peripherie und Stromversorgung
- Schreiben der Software für die BCMV2 sowie MatLab-Skripten zur Datenauswertung

Weitere Teilziele der Bachelorarbeit sind:

- Realisierung einer Ethernet Anbindung für den Datenaustausch
- Ablaufsteuerung des Zyklierbetriebs durch eine Dialogeinstellung und mit Hilfe einer Konfigurationsdatei
- Einbindung von Erweiterungen und Interfaces wie ein LED-Display, RS232 und RS485
- Kanalabhängige Kalibrierung des externen ADC

Im Kapitel 2 werden diese Ziele im Konzept durch Voruntersuchungen im Detail festgesetzt. Darauf erfolgt im Kapitel 3 die Realisierung der Hardware gefolgt durch Realisierung der benötigten Software in Kapitel 4. Im Kapitel 5 wird das System erprobt und in Kapitel 6 bewertet.

2. Voruntersuchung und Konzeption

2.1. Bewährte Schaltungskonzepte

In dem bestehenden Entwurf wurden Schaltungskonzepte entwickelt, die nicht neu definiert werden müssen. Diese bewährten Lösungen werden in den folgenden Unterkapiteln aufgezeigt und gleichzeitig für das neue Konzept übernommen.

2.1.1. Stromversorgung

Der Einsatz des Spannungsreglers TPS73801 [30] hat sich für die Stromversorgung des gesamten Aufbaus bereits bei der BCMV1 bewährt. Es ist ein Low-dropout (LDO) Regler mit bis zu 1 A Ausgangsstrom bei einer einstellbaren Ausgangsspannung zwischen 1.21 V und 20 V. Der Baustein bietet eine interne Strombegrenzung, Abschaltung bei zu hoher Temperatur sowie Schutz vor Rücklaufstrom. Mit dem TPS73801 werden die zwei benötigten Potentiale 5 V und 3.3 V erzeugt.

2.1.2. Spannungsmessung

Hauptfunktionen der BCMV2 sind die Messungen physikalischer Größen. Diese wurden in der BCMV1 ausgiebig untersucht und werden übernommen.

2.1.2.1. ADC

Mit der BCMV2 sollen Zellspannungen gemessen werden können. Zum Einsatz kommt der AD7798 [3], ein ADC des Hersteller Analog Devices. Es ist ein 16-Bit Delta-Sigma-ADC (Δ-Σ-ADC) mit 3 analogen Eingangspaaren. Gemessen wird jeweils die Differenzspannung zweier Potentiale eines Eingangspaares. Bei einer externen Referenzspannungsquelle von 5 V ergibt sich eine Auflösung von:

$$Auflösung = \frac{5V}{2^{16}} = \underline{\underline{76,29\mu V \frac{1}{bit}}} \tag{2.1}$$

Gefordert wird eine Messgenauigkeit von 1 mV. Somit reicht die Auflösung des 16-Bit ADC vollkommen aus. Optional kann man auch auf eine höhere Auflösung mit dem Pin kompatiblen 24-Bit ADC 7799 [3] erweitern:

$$Auflösung = \frac{5V}{2^{24}} = \underline{\underline{0,298\mu V \frac{1}{bit}}} \tag{2.2}$$

Diese Auflösung wird jedoch nicht benötigt, da der 16-Bit ADC bereits eine ausreichende Auflösung bietet. In Kap. 4.6.1 wird untersucht, wie sich Rauschen auf den analogen Eingangskanälen des ADCs auswirkt und ob auch unter realen Bedingungen eine Messabweichung von maximal 1 mV erreicht werden kann. Der ADC kann über eine Serial Peripheral Interface (SPI) Schnittstelle an einen Mikrocontroller angebunden werden. Der externe ADC hat Vorteile gegenüber der im Mikrocontroller vorhandenen internen ADCs. So kann bei einem internen ADC keine externe Referenzspannungsquelle über 3 V angeschlossen werden, welches mit einem Spannungsvorteiler ausgeglichen werden müsste. Weiterhin bieten die internen ADCs keine Möglichkeit Differenzenspannungen zweier Potentiale zu bilden, wie es der externe ADC ermöglicht.

Der AD7798 bietet zwei Modi des Messverfahrens zwischen denen umgeschaltet werden kann. Diese sind der “Bufferd“ und der “Unbuffered“ Mode.

Im Buffered Mode arbeitet der ADC mit internem vorgeschalteten einstellbaren Verstärker. Der analoge Eingang ist hochohmig, was dazu führt, dass ein Potential gewisse Zeit anliegen muss, damit dieses genau genug gemessen werden kann. Dieser Modus ist vorteilhaft, sofern Impedanzen eine große Rolle spielen.

Im Unbuffered Mode wird der analoge Eingang direkt an den internen Δ-Σ-ADC weitergeleitet, der Eingang ist also niederohmig. Dadurch arbeitet der ADC dynamischer und erlaubt schnellere Messdatenerfassung. Zu berücksichtigen ist hierbei, dass der Eingangsstrom im Unbuffered Mode höher ist als im Buffered Mode. Für die BCMV2 wird der Unbuffered Mode gewählt, um möglichst zeitgleiche aufeinander folgende Messungen aller Zellen zu ermöglichen.

Die externe Spannungsreferenz für den ADC von 5V wird durch den Baustein ADR4550 [4] erzeugt. Der ADR4550, wie auch weitere Bauteile dieser Familie [4], ist eine Hochpräzisionsreferenzspannungsquelle mit maximal 0.02 % Abweichung. Ein weiteres Merkmal dieses Bauteils ist die kleine Stromaufnahme von unter 1 mA.

2.1.2.2. REED Relais

Die BCMV2 soll bis zu zwölf Zellen in ihrer Spannung beobachten können. Da wir aber aufgrund galvanischer Einflüsse nur einen ADC benutzen können, müssen die Zellen nacheinander an einen analogen Eingang des ADCs geschaltet werden. Hierfür wurden bisher REED-Relais verwendet. REED-Relais sind aufgrund ihrer hermetisch abgeschlossenen Bauform sehr zuverlässig. Bei geringen Schaltleistungen, wie diese bei der Spannungsmessung erfolgen, sind REED-Relais äußerst langlebig. Für jede Zellspannung werden zwei dieser Relais benötigt. Ein Relais zum Zuschalten des High-Potentials, ein weiteres zum Zuschalten des Bezugpotenials. Bei zwölf Zellen werden somit 24 dieser Reed-Relais benötigt. Die hier verwendeten REED-Relais [5] haben kompakte Bauform, sodass diese auch auf einer Platine verbaut werden können. Die Ansteuerung der Relais wird im Kap. 2.2.3 ausgeführt. Durch wählbare Sicherungen mit der Gehäusebauform 0603 sollen die Eingänge des ADC aus Kap. 2.1.2.1 vor einem Kurzschluss bei Fehlverhalten der Relaisansteuerung geschützt werden.

2.1.3. Strommessung

Eine weitere physikalische Messgröße, die es zu beobachten gilt, ist der Strom. Um den Strom zu messen, wird ein HALL-Sensor benötigt. Wie bereits in der BCMV1 wird der HALL-Sensor DHAB S/24 [13] des Herstellers LEM verwendet. Er eignet sich zur Verwendung mit einem ADC und bietet zwei Strommessbereiche mit $\pm$ 75 A über CH1 und $\pm$ 500 A über CH2. Der Sensor arbeitet radiometrisch, d.h., bei einem Strom von 0 A gibt der Sensor über CH1 und CH2 die halbe Versorgungsspannung zurück. In diesem Fall beträgt Vcc = 5 V und die Ausgänge werden an die beiden übrigen analogen Eingänge des in Kap. 2.1.2.1 vorgesehenen ADC angeschlossen. Über folgende Konversion lässt sich darauf aus dem Messwert des HALL-Sensor der Strom bestimmen:

$$I_P = (V_{OUT} - \frac{V_C}{2}) * \frac{1}{G} * \frac{5}{V_C} \tag{2.3}$$

Die Sensitivität G wird in Abhängigkeit des Strommessbereiches nach Tab. 2.1 eingesetzt.

	G in $[\frac{V}{A}]$
CH1	26.7
CH2	4

Tabelle 2.1.: Sensitivität Strommessbereich

2.1.4. Temperaturmessung

Die letzte physikalische Messgröße, die es zu erfassen gilt, ist die Temperatur einer Zelle. Dies erfolgt jeweils mittels eines Negative Temperature Coefficient Thermistor (NTC), der an die internen ADCs des zu verwendeten Microkontrollers (siehe Kap. 2.2.1) angeschlossen werden kann. Der NTC-Widerstand kann dabei direkt an die zu messende Zelle des Batterieprüflings montiert werden. Für die geforderte Anzahl von zwölf Zellen werden mit der BCMV2 zwölf Anschlüsse für die Temperatursensoren zur Verfügung gestellt. Hierfür werden somit auch zwölf ADC-Eingänge des Mikrocontrollers beansprucht.

Zu Verwendung kommt der NTC B57703M [7] der Firma EPCOS mit folgenden Eigenschaften (Tab. 2.2)

R_{25}	10 $k\Omega$
$B_{25/100}$	3988 K

Tabelle 2.2.: Eigenschaften NTC B57703M

$B_{25/100}$ ist hierbei die Materialkonstante des NTC. Dieser Wert wird benutzt, um trotz nicht linearer Verhältnisse, näherungsweise den Widerstand R_T in Ω bei der absoluten Temperatur T wiederzugeben:

$$R_T = R_{25} * e^{B_{25/100}(\frac{1}{T} - \frac{1}{T_{25}})} \tag{2.4}$$

$$\Rightarrow T = \frac{B_{25/100} * T_{25}}{B_{25/100} + ln(\frac{R_T}{R_{25}}) * T_{25}} \tag{2.5}$$

Setzt man die Werte der Tab. 2.2 in die Gleichung 2.5 ein erhält man mit einem 10-Bit ADC die Temperatur in Kelvin. Das Widerstandsverhältnis verhält sich nach ohmschen Gesetz äquivalent wie das am ADC anliegende Spannungsverhältnis. Zur Umrechnung in Grad Celsius werden nochmal einmal 273,15 K abgezogen.:

$$t = \frac{3988K * 298,15K}{3988K + ln(\frac{2^{10}}{ADC-Wert} - 1) * 298,15} - 273,15 \tag{2.6}$$

2.2. Neue Konzepte

2.2.1. Microcontroller

Zuvor wurde mit der BCMV1 das Evaluationsboard EKI-LM3S9B92 [27] mit Mikrocontroller verwendet. Im Redesign soll jedoch der Mikrocontroller direkt auf der Hauptplatine verbaut werden. Erste Erfahrungen über dieses Verfahren hat Tobias Steinmann in seiner Bachelorarbeit [16] verfasst. Weiterhin gibt im Forschungsprojekt BATSEN sowie an der HAW Hamburg bereits Erfahrungen, sodass die Wahl des LM3SB92 [28] als Mikrocontroller beibehalten wird. Da dieser Controller aber nicht mehr auf dem Markt verfügbar ist, muss auf den LM3SD92 [29] ausgewichen werden. Beide genannten Mikrocontroller sind untereinander kompatibel. Unterschiede sind der doppelte interne Flashspeicher des LM3SD92 sowie der zusätzliche umschaltbare 12-bit Modus des ADCs neben dem 10-bit Modus. Nachfolgend die Eigenschaften des Controllers gemäß Datenblatt sowie das Blockschaltbild in Abb. 2.1:

- ARM Cortex-M3 Processor Core
- High Performance: 80-MHz operation; 100 DMIPS performance
- 512 KB single-cycle Flash memory
- 96 KB single-cycle SRAM
- Internal ROM loaded with StellarisWare® software
- External Peripheral Interface (EPI)
- Advanced Communication Interfaces: UART, SSI, I2C, I2S, CAN, Ethernet MAC and PHY, USB
- System Integration: general-purpose timers, watchdog timers, DMA, general-purpose I/Os
- Advanced motion control using PWMs, fault inputs, and quadrature encoder inputs
- Analog support: analog and digital comparators, Analog-to-Digital Converters (ADC), on-chip voltage regulator
- JTAG and ARM Serial Wire Debug (SWD)
- 100-pin LQFP package

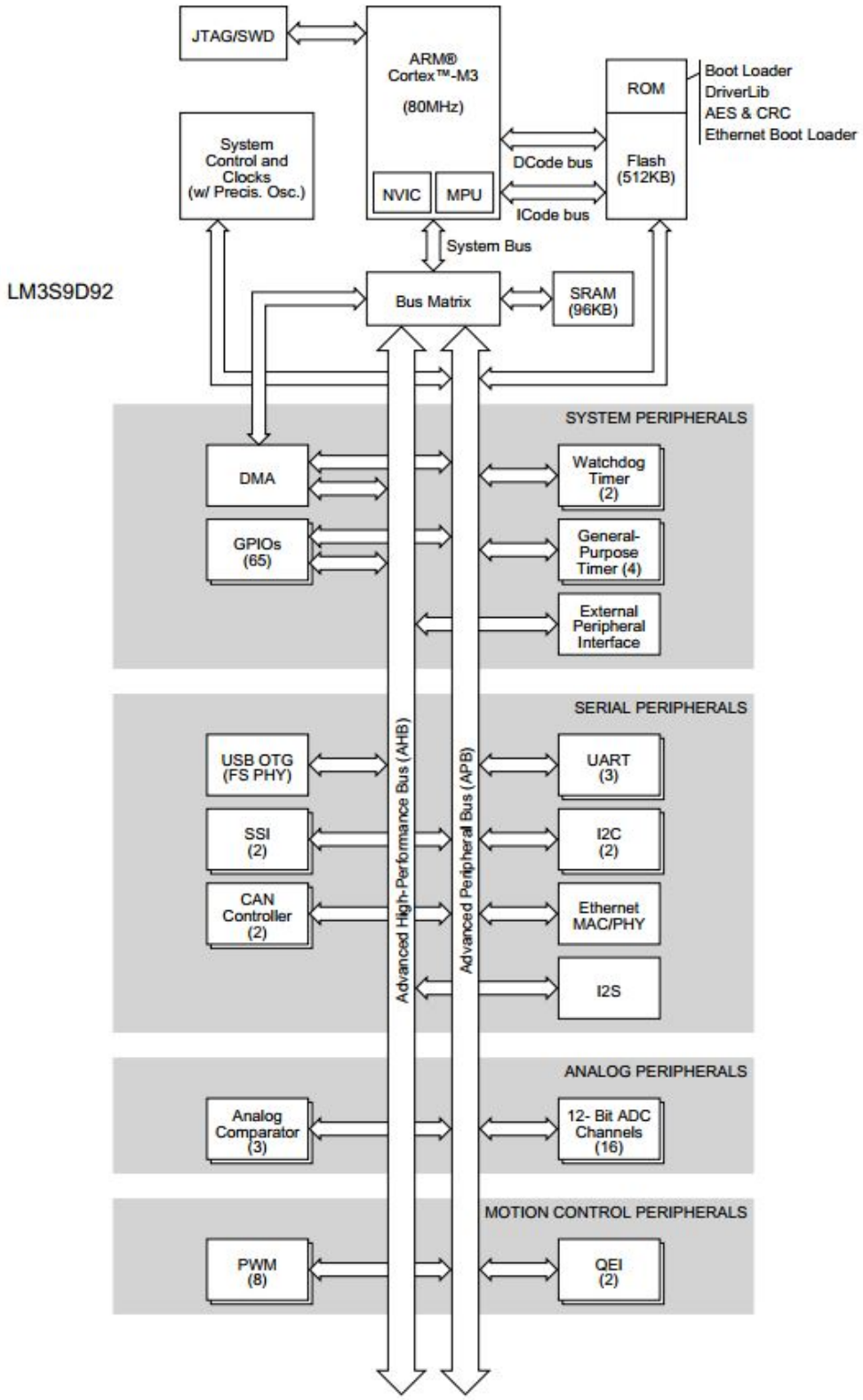


Abbildung 2.1.: High-Level Block Diagram LM3S9D92 (Quelle: [29])

Für die elektronische Inbetriebnahme des Mikrocontrolles werden neben R,L und C-Elementen noch zwei Quarze benötigt (Tab. 2.3):

Q1	25 MHz
Q2	16 MHz

Tabelle 2.3.: Dimensionierung der Quarze

Der Quarz Q1 dient als Quelle für die Taktrate des Controllers. Q2 ermöglicht die Nutzung der Ethernetschnittstelle. Eine Ethernetbuchse kann direkt and de Mikrocontroller angebunden und per Software angesteuert werden. Ethernet wird in Kap. 4.5 behandelt.

In der Tab. 2.4 wird dargestellt, welche Peripherie über welche Schnittstellen an den Mikrontroller angebunden wird:

UART (1-3)	USB-Terminal, RS-232, RS-485
SSI	SD-Karte, 16-Bit ADC
I2C	Real Time Clock
12/10-Bit ADC	12 x NTC-Sensor
PWM	LED Kontrasteinstellung
GPIOS	REED-Relais, Buttons, LED-Display, LEDs

Tabelle 2.4.: Festlegung der Schnittstellen für Peripherie

2.2.2. Programmierschnittstelle

Der in Kap. 2.2.1 vorgestellte Mikrocontroller verfügt über eine Joint Test Action Group - Programmier-/Debug-Schnittstelle (JTAG). Über ein In-Circuit Debug Interface Board (ICDI-Board), welches Teil des Stellaris Evaluationsboard [27] ist, konnte eine Schnittstelle zwischen JTAG und einem virtuellen COM-Port für PC über einen Universal Serial Bus (USB) generiert werden. Kern des ICDI-Board ist der FTDI-Chip FT2232D [9]. Der Chip ermöglicht über USB zwei seperate serielle bzw. parallele Schnittstellen zu integrieren, die frei einstellbar sind. Somit unterstützt er die Generierung einer JTAG-Schnittstelle über die programmiert und debugged werden kann und eine UART-Schnittstelle zum ansteuern eines Terminals auf dem PC.

Ziel ist es, das ICDI-Board in das Konzept zu integrieren. So soll der FTDI-Chip direkt auf dem Mainbaord des BCMV2 installiert werden, sodass die Schnittstelle On-Board verfügbar ist. Die Konfiguration des Programmierchips wird auf dem Electrically Erasable PROgrammable Memory (EEPROM) M93C86 [17] mit 16 Kbit Speicher des Herstellers STMicroelectronics gespeichert.

Die erfolgreiche Erprobung dieser Maßnahme erfolgte bereits in der Bachelorarbeit durch Jan Schlüter [15] und wird an dieser Stelle in das Konzept der BCMV2 aufgenommen. Die Programmierschnittstelle wird in Kap. 4.1 konfiguriert.

2.2.3. Relaisumschaltlogik

Wie bereits aus Kap. 2.1.2.2 hervorgeht, wird eine Ansteuerung der REED-Relais benötigt. Hierbei müssen die 24 vordefinierten REED-Relais paarweise dem ADC aus Kap. 2.1.2.1 zugeschaltet werden, sodass zwölf Ausgangssignale erforderlich sind. Eine Möglichkeit wäre die direkte paarweise Ansteuerung durch General Purpose Input/Output (GPIO)-Ports des Mikrocontrollers. Nachteilig hierbei ist zum einen die Belegung von 12 Pins am Mikrocontroller. Zum anderen gilt es die Steuerlast zu betrachten, welche die REED-Relais in schaltendem Zustand benötigen. Laut Datenblatt der Relais [5] haben die Steuerspulen für 5 V einen Widerstand von 500 Ω. Da die GPIO-Ports des Mikrocontrollers [29] jedoch maximal 3,3 V treiben, können diese die Aufgabe nicht direkt übernehmen. Eine Treiberstufe muss für folgende Stromaufnahme pro Relais Gl. 2.7 dimensioniert werden:

$$I = \frac{U}{R} = \frac{5V}{500\Omega} = \underline{\underline{10mA}} \tag{2.7}$$

Beim paarweisen Zuschalten der REED-Relais werden 20 mA pro Steueranschluss benötigt. Um auch GPIO-Ports einzusparen, wird für diese Aufgabe ein Analogmultiplexer verwendet. Der ADG726 [2] ist ein dual 16 zu 1 Analogmultiplexer mit sieben Steuereingängen. Schaltet man die Select-Eingänge $\overline{CSA}$ und $\overline{CSB}$ zusammen (siehe Abb. 2.2) und mit $\overline{WR}$ gegen GND, so können die Relais paarweise mit einer logischen Belegung von vier Steuerleitungen durch den Mikrocontroller geschaltet werden. Der Analogmultiplexer kann dabei pro Kanal 30 mA im Nennbetrieb treiben. Über ein Enable-Signal $\overline{E}$ können darauf die Ausgänge des ADG726 zum gewünschten Zeitpunkt vom Mikrocontroller gesetzt werden.

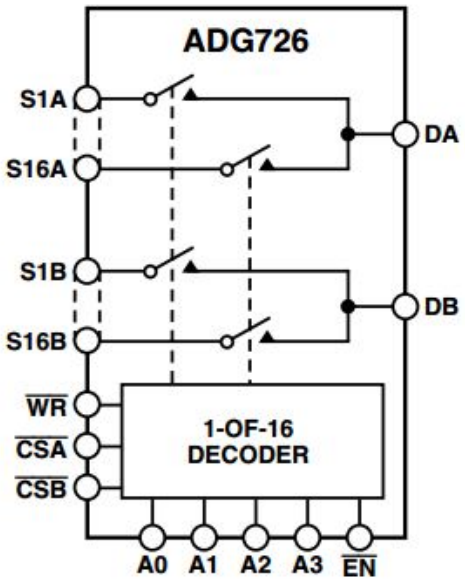


Abbildung 2.2.: Block Diagramm Dual-Analogmultiplexer ADG726 (Quelle: [2])

Von den 16 möglichen bidirektionalen Ausgängen des ADG726 werden bei der BCMV2 nur 12 für die 24 REED-Relais benutzt.

2.2.4. RS232 und RS485

Um eine mögliche Anbindung externer Geräte zu ermöglichen, werden an dieser Stelle Bussysteme konzeptioniert. Hierbei sollen UART-Schnittstellen des Mikrocontrollers verwendet werden. Zwei relevante Bussysteme sind der Recommended Standard 232 (RS232) und der Recommended Standard 485 (RS485).

Mit RS232 können seriell Geräte angebunden werden, um eine bidirektionale Kommunikation maximal zweier Teilnehmer zu ermöglichen. Hierfür wird ein Baustein benötigt, der eine Pegelwandlung von Transistor-Transistor-Logik (TTL) zum Pegel des RS232 vornimmt. Da zwei von drei UART-Schnittstellen des Mikrocontrollers bisher unbelegt sind, werden zwei Schnittstellen für RS232 vorbereitet. Eine der beiden UART-Schnittstellen unterstützt hierbei das Full-handshaking-Verfahren, womit der Sender mittels Ready to Send (RTS) eine Übertragung ankündigt und der Empfänger über Clear to Send (CTS) die Empfangsbereitschaft von Daten quittiert. Für die Pegelwandlung wird der MAX3232E [23] von TI eingesetzt.

Die UART-Schnittstelle ohne Fullhandshakeverfahren soll später durch mechanisches Umsetzten von Jumpern von RS232 auf RS485 umschaltbar sein. RS485 bietet die Möglichkeit, eine serielle Kommunikation mehrerer Teilnehmer am Bus zu realisieren. Hierfür wird ein Transceiver benötigt, der die benötigte Bustopologie von RS485 generiert. Ausgewählt hierfür wurde der SN75176b [21] von TI. Hierbei gilt, je nach interner Kontaktbeschaltung, folgende Pinbelegung am jeweiligen D-SUB9 Anschluss (Tab. 2.5):

	UART1	UART2	
Pin-Nr.	RS232	RS232	RS485
1	-	-	-
2	Rx	Rx	B
3	Tx	Tx	A
4	-	-	-
5	GND	GND	GND
6	-	-	-
7	RTS	-	-
8	CTS	-	-
9	-	-	-

Tabelle 2.5.: Pinbelegung der RS232 und RS485 Anschlüsse (D-SUB9)

2.2.5. SD-Karte

Die Messdaten der Spannungs-, Strom- und Temperaturerfassung müssen kontinuierlich abgespeichert werden. Der Mikrocontroller unterstützt mittels vorkonfigurierter Bibliotheken die Anbindung eines Massenspeichers unter Verwendung eines FatFs-Moduls. Mittels SPI lässt sich somit in einfacher Form eine SD-Karte anbinden, auf der ein File Allocation Table (FAT)-Filesystem initialisiert werden kann. Dabei werden die Formate FAT12, FAT16 und FAT32 unterstützt, je nachdem, wie die SD-Karte formatiert ist.

Die Nutzung einer SD-Karte wurde bereits mit der BCMV1 erfolgreich erprobt, das Konzept soll hier jedoch noch einmal erläutert und leicht verändert werden. Um einen SPI-Anschluss am Mikrocontroller einzusparen, wird die SD-Karte an denselben SPI-Anschluss angeschlossen wie der in Kap. 2.1.2.1 erwähnte externe ADC.

Mittels manuellem Umschalten des Chip select (CS) in der Software kann der Mikrocontroller als Master bestimmen, welcher der beiden Slaves, also der ADC oder die SD-Karte, den SPI-Bus beanspruchen kann. Beim Umschalten wird die Taktrate des SPI-Buses umkonfiguriert und umpolarisiert. Standardmäßig läuft die SD-Karte mit 12,5 MHz, was eine Schreibgeschwindigkeit von 12,5 Mb/s ermöglicht. Der Anschluss an den ADC erfordert eine Taktrate von 4 MHz.

Je nach SD-Karte, Speichergröße und Formatierung gibt es maximale Größen von Dateien. Angenommen es wird eine Speicherkarte mit 2 GB Speicherplatz verwendet, so wird mit dem Format FAT16 gearbeitet. Dabei kann eine Datei maximal 2 GB - 1 Byte groß werden. Unter Verwendung einer Speicherkarte mit 4 GB oder mehr wird das FAT32 Format genutzt. Dabei wächst die maximale Größe einer Datei auf 4 GB - 1 Byte.

Die Nutzung einer SDHC-Karte, die auch Speicherplatz von 4 GB bis 32 GB unterstützt, ist möglich. Dafür muss der SPI-Anschluss auf 25 MHz getaktet werden, was aufgrund der Tatsache, dass die SPI-Schnittstelle direkt mit der Taktrate des vorhanden Mikrocontroller unter Verwendung eines Vorteilers versorgt werden kann, möglich ist. Die BCMV2 wird auf eine Taktrate von 50 MHz dimensioniert. Dadurch wird eine Datenrate von 25 Mbit/s oder mehr ermöglicht. Hierbei beschränkt sich jedoch die maximale Größe einer Datei weiterhin auf 4 GB - 1 Byte.

An dieser Stelle soll noch untersucht werden, wie viel Zeit benötigt wird, im Zyklierbetrieb eine 4 GB - 1 Byte große Datei zu generieren. Angenommen, jede Sekunde erfolgt eine Messung aller 12 Zellen. Für jede Zelle wird die Spannung, Temperatur und der aktuelle

Strangstrom gespeichert. Zu jeder Messung gehört auch die Zeit als zu speichernder Wert. Weiterhin wird festgelegt, dass pro Messung einer Zelle ein String der Log-File auf der SD-Karte hinzugefügt wird, der in etwa so aussieht:

$$CE, YYYY, MM, DD, HH, MM, SS, VOL(uV), TEM, CUR(mA), R$$

Ein solcher String enthält demnach 46 Zeichen oder 46 Bytes. Erfolgt das hinzufügen einer solchen Zeile Pro Zelle pro Messung, so ergeben sich pro Sekunde:

$$12 * 46Bytes * 1s = \underline{\underline{552Byte/s}} \tag{2.8}$$

Um nun eine Dateigröße von 4GB - 1 Byte zu erreichen, errechnet sich eine benötigte Zeit von ca. 90 Tagen (Gl. 2.9):

$$t = \frac{4GB - 1Byte}{552Byte/s} = 7780738s = 2161h = \underline{\underline{90d}} \tag{2.9}$$

Da die Messungen in der Regel wenige Tage dauern, würde die Nutzung einer 2 GB Speicherkarte vollkommen ausreichen, die mit der halben Dauer der in Gl. 2.9 errechneten Zeit vollgeschrieben wäre, was immerhin auch 45 Tage beanspruchen würde. Der angenommene Fall ist dabei ein sogenannter “Worst Case“ Fall, da die maximale Anzahl von Zellen bei höchster Zyklusrate (1 sec) auf die SD-Karte geschrieben wird. Die Nutzung einer SDHC-Karte ist, sofern auch möglich, nicht zwingend notwendig. Daher wird die Taktrate für die SD-Karte auf 12,5 MHz festgesetzt. Die Nutzung einer SDHC-Karte wurde im Rahmen dieser Bachelorarbeit nicht erprobt.

2.2.6. LED-Display

Um im laufendem Zyklierbetrieb auf aktuelle Informationen zugreifen zu können, soll ein Display integriert werden. Hierzu wurde ein DOTMATRIX LED-Display der Reihe Blueline vom Hersteller Electronic Assembly ausgewählt, das EA W204B-NLW [6]. Durch die Nutzung von LEDs benötigt das Display wenig Strom. Die Displayansteuerung erfolgt onboard mit zwei HD44780 [12] Controllern, welche vom ARM-Controller angesprochen werden können. Hierbei lässt sich das Display im 8 oder 4-Bit Modus betreiben. Im Fall der BCMV2 wird der 4-Bit Modus gewählt, um Pins einzusparen, die durch andere notwendige Peripherie benötigt werden. Weiterhin gilt es die Hintergrundbeleuchtung mit einer Kontrasteinstellung einzustellen. Dies erfolgt mit einer Spannung zwischen 0 und 0,5 V. Um diese Einstellung softwareseitig regelbar zu gestalten, wird hierzu ein PWM-Signal vom Mikrocontroller erzeugt. Weiterhin gilt es den Strom für die LED-Beleuchtung einen Vorwiderstand zu dimensionieren. Bei einer Flussspannung von 3,3V und einem maximalen Strom von 45 mA dimensioniert sich der Vorwiderstand wie folgt (Gl. 2.10):

$$R_{LED} = \frac{U_{LED}}{I_{max}} = \frac{3.3V}{45mA} \approx \underline{\underline{75\Omega}} \tag{2.10}$$

Da in der E-Reihe 24 75 Ω nicht vorhanden sind, wird der nächstgrößere Wert ausgewählt. Somit ergibt sich ein Vorwiderstand von 82 Ω.

Um mit der LED-Anzeige zu arbeiten, werden vier Anschlüsse für Taster zur Verfügung gestellt, welche über GPIO-Ports angeschlossen werden können. Hierbei ist es wichtig, diese Buttons zu entprellen. Hierzu wurde von den vorhandenen Tastern die Entprellzeit mit einem Oszilloskop aufgezeichnet (Abb. 2.3):

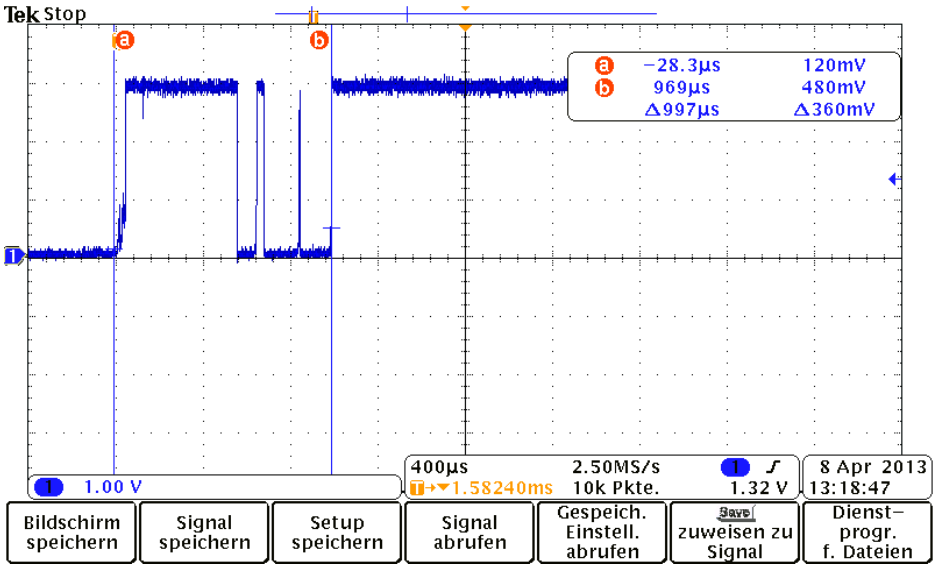


Abbildung 2.3.: Bestimmung Entprellzeit Taster für Displaysteuerung

Aus dem Osziloogramm (Abb. 2.3) lässt sich eine Entprellzeit von ca. t = 1 ms ablesen. Für saubere Flanken wird nun jeweils eine Entprellschaltung zwischen Taster und Mikrocontroller eingebaut. Dabei wird ein invertierender Hex-Schmitt-Trigger von TI verwendet, der SN74AHCT14 [20]. Vor dem Schmitt-Trigger wird ein RC-Tiefpass angeschlossen, um das Prellen wirkungsvoll zu unterdrücken (Abb. 2.4):

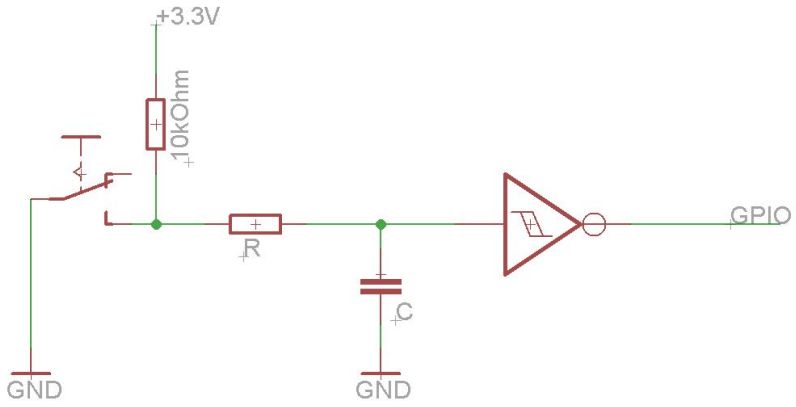


Abbildung 2.4.: Entprellschaltung mit invertierendem Schmitt-Trigger

Für die Dimensionierung sind weiterhin die Schwellspannungen des Schmitt-Triggers zu beachten. Diese liegen laut Datenblatt [20] bei U_H = 1,8 V und U_L = 1,4 V. Rechnerisch ergibt sich z.B. für U_L eine Zeitkonstante τ von 1,8 ms (Gl. 2.12):

$$U_L(t) = U * (1 - e^{\frac{t}{\tau}}) \tag{2.11}$$

$$\Rightarrow \tau = \frac{-t}{ln(1 - \frac{U_L}{U})} = \frac{-1ms}{ln(1 - \frac{1.4V}{3.3V})} = \underline{\underline{1.8ms}} \tag{2.12}$$

Mit dieser rechnerischen Zeitkonstante kann man nun den RC-Tiefpass dimensionieren. Da es vorkommen kann, dass unterschiedliche Schalter auch längere Entprellzeiten haben können, wird die Zeitkonstante pauschal erhöht und die Bauteile an die E24-Reihe angepasst. Somit erhalten wir z.B. bei $\tau_{absolut}$ = 2,7 ms den Zusammenhang (Gl. 2.13) mit den dimensionierten Werten aus der Tab. 2.6:

$$\tau_{absolut} = \tau + \tau_{toleranz} = 1,8ms + 0,9ms = \underline{\underline{2.7ms = R * C}} \tag{2.13}$$

R	27 kΩ
C	0,1 μF

Tabelle 2.6.: Dimensionierung Entprellschaltung

2.2.7. Real-Time-Clock

Bei der Messdatenerfassung ist der Messzeitpunkt eine weitere relevante Zusatzinformation. Der Mikrocontroller bietet die Möglichkeit einer internen Real Time Clock (RTC) mit eigenem Oszillator, welches für eine genaue Uhrzeit ausreichend ist. Jedoch erfordert die interne RTC eine Übergabe von Anfangswerten nach Einschalten der Spannungsversorgung, da die Uhr nicht weiterläuft wenn das Gerät ausgeschaltet ist. Der LM3S9D92 [29] bietet keine Möglichkeit des Anschlusses einer Backup-Batterie welche die RTC auch arbeiten lässt, sofern das Gerät ausgeschaltet ist.

Um beim Einschalten des Gerätes eine manuelle Einstellung der aktuellen Uhrzeit zu vermeiden, wird eine externe RTC von TI ins Konzept aufgenommen. Die BQ32000 [24] ermöglicht es, beim Einschalten des Gerätes eine aktuelle Uhrzeit automatisch während des Initialisierung des Mikrocontrollers über eine Inter-Integrated Circuit (I2C) Schnittstelle zu übergeben.

Für die externe RTC muss weiterhin ein Batteriehalter für eine Knopfzelle sowie ein Uhrenquarz mit 32,768 kHz vorgesehen werden. Wird die RTC von der Backup-Batterie betrieben, so liegt der Stromverbrauch der RTC bei maximal 1,5 μA. Nutzt man eine Knopfzelle des Typs CR 2032 mit 220 mAh, so reicht diese im Idealfall ohne äußere Einflüsse laut Gl. 2.14 für über 145.000 Stunden bzw. 16,5 Jahre.

$$t = \frac{220mAh}{1,5\mu A} = 146667h = 61110d \approx \underline{\underline{16,5a}} \tag{2.14}$$

2.2.8. Lasttreibermodul

Neben der BCMV2 wird eine weitere Platine benötigt, welche die Lastrelais treiben kann sowie die Grundversorgungsspannung für die BCMV2 liefert. Über Datenleitungen soll eine Steuerlogik ausgehend vom BCMV2 verarbeitet werden. Erste erfolgreiche Tests konnten hierfür mit der Load Driver Module V1 (LDMV1) erfolgen. Die LDMV1 wurde im Rahmen einer studentischen Hilfskraft von Herrn Alexander Hansen entwickelt und aufgebaut. Im Laufe des Projekts muss die LDMV1 jedoch neu konzeptioniert und um Funktionen erweitert werden. Dazu wird auch ein Design-Fehler behoben, der undefinierte Zustände der Relaisumschaltlogik ermöglichte, da die Latch-Enable-Datenleitung ($\overline{LE}$) für den in Kap. 2.2.8.2 erwähnten Logikbaustein vom Mikrocontroller auf Grund fehlender Anschlussmöglichkeit nicht ansprechbar war. Das ($\overline{LE}$) wird zum zeitlichem Freigeben der Ausgänge des Logikbausteins benötigt. Die neuen Konzepte werden in der Load Driver Module V2 (LDMV2) aufgegriffen, wodurch ein direkter Nachfolger, basierend auf der Arbeit von Alexander Hansen, entwickelt werden soll.

2.2.8.1. Lastrelais

Die zu messenden Batteriezellen sollen über Lastrelais in verschiedene Zustände gebracht werden. So sollen die Batterieprüflinge während des Messvorganges z.B. über ein Ladegerät geladen oder über eine elektronische Last entladen werden. Mit vier Lastrelais können dabei vier verschiedene Geräte dem Lastkreis des Batterieprüflings zugeschaltet werden. Der Aufbau könnte dabei beispielhaft wie in Abb. 2.5 erfolgen.

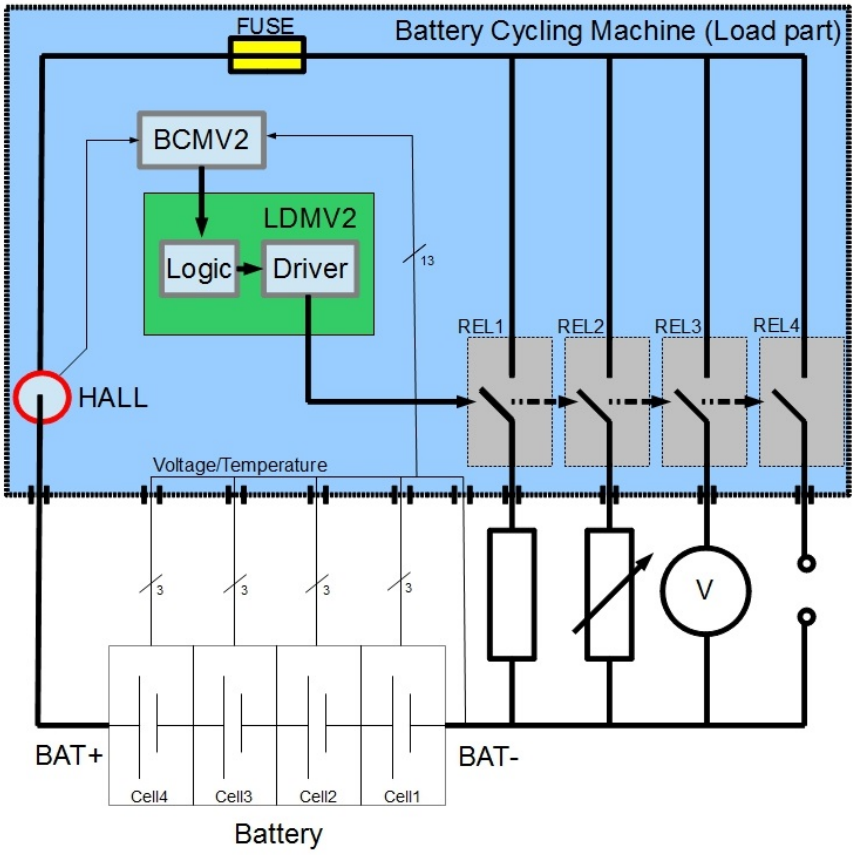


Abbildung 2.5.: Interne Beschaltung der Lastrelais am Bsp. mit 4 Zellen

Im Rahmen dieser Arbeit werden Ströme bis zu 10 A erwartet. Um auch mögliche höhere Ströme treiben zu können, werden Relais mit einer Lastspitze von 20 A bei 400 V ausgewählt. Die Steuerspulen der Relais benötigen eine Ansprechspannung von 12 V. Dabei nehmen diese einen Strom von ca. 0,14 A auf. Der Spulenwiderstand beträgt 86 Ω. Über eine 20 A Sicherung sollen die Relais und vor allem die Batterie selbst vor Überstrom geschützt werden.

2.2.8.2. Steuerlogik

Die Lastrelais werden von N-MOSFETs getrieben welche auf der LDMV2 untergebracht werden sollen. Der CSD18504Q5A [31] eignet sich um die Steuerlast der Lastrelais treiben zu können. Da dieser N-MOSFET auch Spannungen von 40V bei maximalem Dauerstrom von 15A treiben kann, können auch optional größere Lastrelais zugeschaltet werden.

Bei der ganzen Steuerlogik, welche vom Mikrocontroller der BCMV2 ausgeht, ist es wichtig dass zeitgleich immer nur ein Relais aktiv sein darf. Dafür ist ein Bauteil nötig, welches aus einer bestimmten Eingangslogik nur ein Ausgangskanal und somit ein N-MOSFET gleichzeitig beschalten kann. Diese Aufgabe hat bereits in der LDMV1 der Demultiplexerbaustein CD74HC4514 [19] übernommen. Von den vier Dateneingängen werden dabei nur drei benutzt. Entsprechend der aufgeführten Wahrheitstabelle (Tab. 2.7) können somit die vier N-MOSFETs der Lastrelais angesprochen werden. Darüber hinaus können noch drei weitere adaptive Ausgänge aktiviert werden, welche über Stecker auf der Platine des LDMV2 frei belegbar sein sollen.

A3	A2	A1	OUTPUT	RELAY
0	0	0	Y0	-
0	0	1	Y1	REL1
0	1	0	Y2	REL2
0	1	1	Y3	REL3
1	0	0	Y4	REL4
1	0	1	Y5	FREE
1	1	0	Y6	FREE
1	1	1	Y7	FREE

Tabelle 2.7.: Wahrheitstabelle Relais-Logik-Encoder

2.2.8.3. Notlaufprogramm bei Stromausfall

Eine weitere neue Funktion der LDMV2 beinhaltet ein Notlaufprogramm bei Ausfall des Stromnetzes. Die Platine soll über eine einfache direkte Diodenschaltung (D1 und D2) bei Stromausfall auf eine Backup-Batterie zugreifen und das gesamte Gerät weiterhin versorgen. Siehe hierzu zu Abb. 2.6. Für den Zyklierbetrieb ist es dabei wichtig, diese Stromausfälle zu erkennen. Daher wird zur Detektion der Stromquelle ein Signal ($POWER\_FAIL$) an den Mikrocontroller der BCMV2 geleitet, das im Normalbetrieb einen High-Pegel aufweist. Bei Stromausfall wird der Pegel gegen GND gezogen, worauf der Mikrocontroller ein Notlaufprogramm einleiten kann, welches in Kap. 4.9 erläutert wird. Der Spannungsteiler mit R1 und R2 ermöglicht die Anpassung des Signalpegels auf 3,3 V.

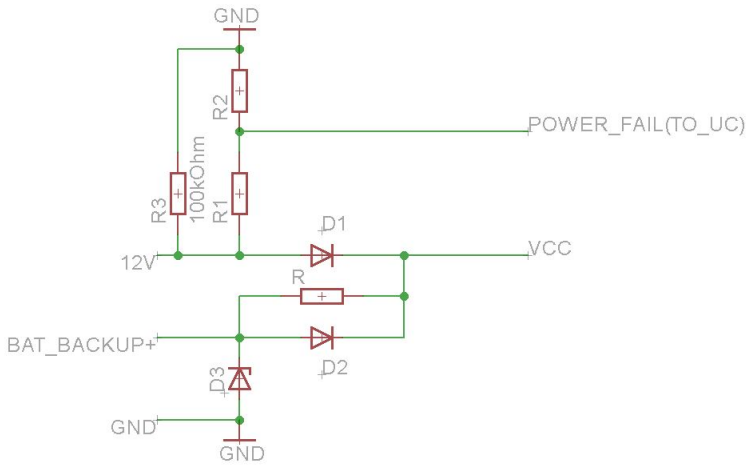


Abbildung 2.6.: Detektion Stromausfall

Optional kann auch die Backup-Batterie über den Widerstand R und einer Zenerdiode Z geladen werden. Die Bauteile sollen vorgesehen werden, im Rahmen dieser Bachelorarbeit werden diese jedoch nicht dimensioniert.

2.2.9. Gesamtsystem

Für das Gesamtsystem ist es vorgesehen, sämtliche Bauteile und Platinen in einem Gehäuse unterzubringen. Als Stromversorgung soll ein Netzteil für 12 V integriert werden. Hier zu wird ein 19 Zoll Sub Rack eingeplant an dem ein Frontpanel zum Bedienen des Systems angebracht werden kann. Im Frontpanel kann sämtliche Peripherie wie z.B. das LED-Display, ein SD-Karten Slot, Eine Ethernet Buchse sowie weitere Anschlüsse für Temperatursensoren oder Zellkontakte angebracht werden. Weiterhin ist eine USB-Buchse eingeplant, mit der zum einen Software aufgespielt werden zugleich aber auch über UART eine Konsole zum Konfigurieren der Zyklenablaufsteuerung aufgerufen werden kann. In Abb. 2.7 ist das Gesamtsystem am Beispiel einer angeschlossenen vier-zelligen Batterie abgebildet.

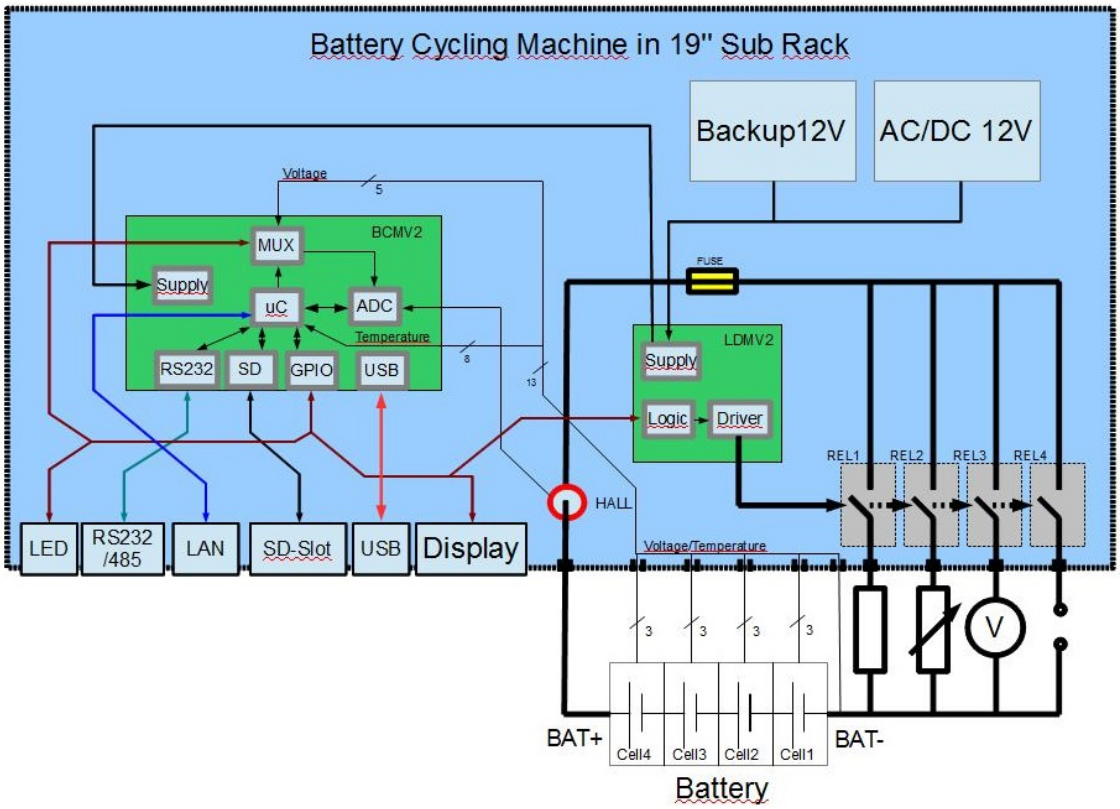


Abbildung 2.7.: Konzept Gesamtsystem am Bsp. mit 4 Zellen

3. Schaltungsentwicklung, Aufbau und Inbetriebnahme

3.1. Hauptplatine

Die Schaltpläne und die Platinenlayouts sämtlicher Platinen wurden mit der Software Eagle 5.11.0 erzeugt. Dabei wurden die in Kap. 2 erwähnten Konzepte umgesetzt. Neben der Hauptplatine für die BCMV2 wurde eine weitere Platine für das LDMV2 in Kap. 3.2 entwickelt.

3.1.1. Schaltplan

Der Schaltplan für die Hauptplatine der BCMV2 ist im Anhang B.1 zu finden. Der Übersicht wegen ist der gesamte Schaltplan auf 12 Unterpläne aufgeteilt. Jeder Unterplan zeigt dabei ein Teilsystem der BCMV2:

1. MikroController - LM3S9D92
2. Spannungsversorgung
3. Programmierschnittstelle und Reset
4. SD-Karte, Buttons, Display
5. RS232 & RS232/RS485
6. ADC
7. Ethernet
8. Relais MUX
9. Temperatursensoren
10. Relais für ADC Unteres Potential
11. Relais für ADC Oberes Potential
12. RTC

Für einige benutzte Bauelemente sind keine Bibliotheken direkt in Eagle vorhanden. Für die Bauelemente des Herstellers TI konnten entsprechende Bibliotheken im Format 'bxl' direkt von der Homepage des Herstellers geladen werden. Mittels der von TI zur Verfügung gestellten Software Ultra Librarian können die Bibliotheken darauf ins Eagle Format 'lbr' konvertiert werden. Die gleiche Prozedur war auch für Bauteilbibliotheken des Herstellers Analog Devices unter Verwendung von Ultra Librarian möglich.

Bibliotheken für diverse Bauteile wie REED-Relais, Quarze, Dioden oder der Batteriehalter waren zum Teil nicht verfügbar und mussten mit dem Bibliotheken Erzeugungs-Tool von Eagle manuell erstellt werden.

3.1.2. Platinenlayout

Das Platinenlayout der BCMV2 aus dem Anhang C.1 wurde ebenfalls mit Eagle erzeugt. Die Ausmaße der Platine entsprechen einer Eurokarte mit 160 mm x 100 mm. Es ist eine doppelseitige Platine, wobei alle Pads der Bauteile, bis auf den Batteriehalter für die RTC, auf dem Top-Layer positioniert wurden. Auf Grund der Komplexität der Schaltung wurden die Leiterbahnen manuell verlegt. Auf den in Eagle integrierten Autorouter wurde dabei verzichtet. Besonders in den Schaltungsteilen um den Mikrocontroller sowie der Programmierschnittstelle ist die Leiterbahnanzahl und -dichte besonders hoch. Dies folgt aus dem Konzeptentwurf, der eine Belegung aller 100 Pins am Mikrocontroller erfordert. Die Leiterbahnstärke der Signalleitungen wurde durchgehend auf 0.3 mm oder mehr eingestellt. Leiterbahnstärken der Stromversorgung wurden zwischen 0.4 mm und 0.8 mm gewählt.

Die Komponenten, die mit dem im Kap. 2.1.2.1 konzeptionierten ADC verbunden sind, haben eine eigene Massefläche, welche von der Massefläche der restlichen Platine getrennt ist und über drei Null-Ohm-Brücken **(R17, R19 und R100)** angebunden werden kann. Dadurch soll ein Einwirken der digitalen Komponenten der kompletten Schaltung auf die analoge ADC-Messung möglichst unterdrückt werden.

Der Widerstand **R72** darf nicht bestückt werden. Dies geht aus der Bachelorarbeit von Tobias Steinmann [16] hervor. Durch Verwendung von **R72** wäre die Masse der Platine mit der Masse der Ethernetleitung verbunden, sodass die galvanische Trennung beider Ethernetteilnehmer nicht gewährleistet wäre.

Externe Peripherie die am Frontpanel (Kap. 3.3) angebaut wird, kann über vorhandene Anschlüsse auf der Platine angebunden werden. Hierzu zählen das LED-Display, die Buttons, die LEDs, die Anschlüsse für die Temperatursensoren sowie für die Batteriezellen, der SD-Kartenhalter und die Schnittstellen für Ethernet, USB, RS232 und RS485. Weiterhin sind Anschlüsse zum Anbinden der LDMV2 und des HALL-Sensors vorhanden.

3.2. Lasttreibermodul

3.2.1. Schaltplan

Im Anhang B.2 befindet sich der Schaltplan für die LDMV2 auf der ersten Seite und wurde dem Konzept aus Kap. 2.2.8 nach mit Eagle entwickelt. Bei dem Schaltplan wurden ebenso wie bei der BCMV2 (Kap. 3.1) eigene Bibliotheken für Bauteile erzeugt. Eine wichtige eigens erzeugte Bibliothek ist die Bibliothek für den N-MOSFET Baustein. Je nach Schaltungslogik werden die Lastrelais von den N-MOSFETS Low-Side geschaltet, d.h. im Ruhezustand sind die Ansteuerspulen der Lastrelais von der Masse getrennt und werden von den N-MOSFETS mit der Masse verbunden.

3.2.2. SD-Kartenhalter

Auf der zweiten Seite des Schaltplanes im Anhang B.2 befindet sich der Schaltplan für den SD-Kartenhalter welcher am Frontpanel (Kap. 3.3) montiert werden soll. Dieser Schaltplan soll im Layout des LDMV2 in einer eigenen Sektion berücksichtigt werden. Somit können zwei Platinen mit einem Platinenlayout erzeugt werden, worauf nach Trennung durch Fräsen wieder beide Platinen erzeugt werden können.

3.2.3. Platinenlayout

Die Platinenlayout der LDMV2 sowie des SD-Kartenhalters befindet sich im Anhang C.2. Die Maße der gesamten Platine sind 150 mm * 50 mm. Deutlich zu erkennen sind dabei die gestrichelten Umrisslinien für die Fräsung. Nach Fräsung belaufen sich die Maße der LDMV2 auf 100 mm * 50 mm und die die Maße des SD-Kartenhalters auf 40 mm * 40 mm. Über Schraubklemmen können Versorgungsspannung und die Backup-Batterie sowie die vier Lastrelais an die LDMV2 angeschlossen werden. Über einen Anschluss kann die Steuerlogik des BCMV2 zugeführt werden. Weiterhin erzeugt der LMS1587 [22] aus 12 V die benötigten 7.5 V Spannungsversorgung für die Hauptplatine mit maximalem Ausgangsstrom von 3 A.

3.3. Gehäuse und Frontpanel

Die Platinen BCMV2 und LDMV2 wurden nach Anfertigung und Bestückung in einem 19“ Rack zusammen mit sämtlichen weiteren Komponenten montiert und miteinander verbunden. Den Aufbau zeigt die folgende Abb. 3.1 (von oben):

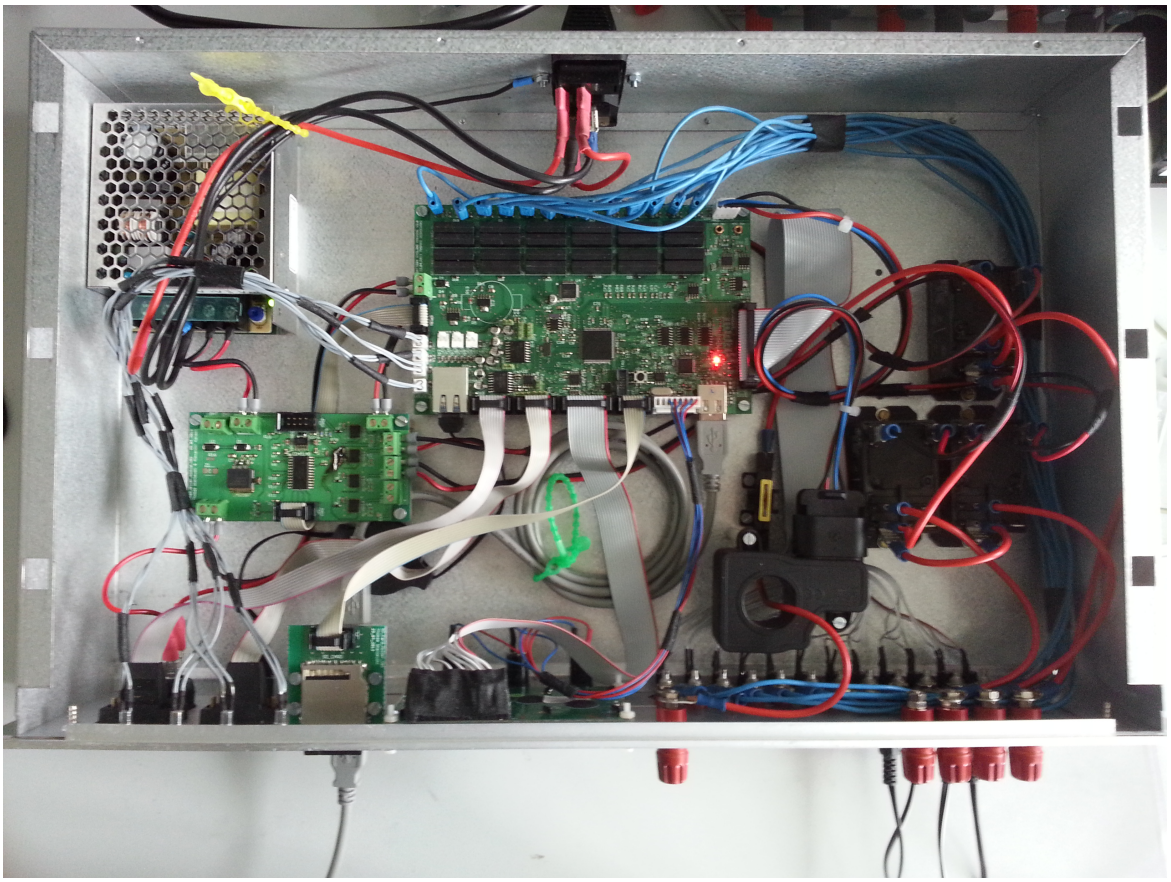


Abbildung 3.1.: Gesamtansicht innerer Aufbau des 19“ Rack

Die Hauptplatine BCMV2 befindet sich mittig im Gehäuse (siehe Abb. 3.1). Im linken Drittel befindet sich die Hauptstromversorgung wie auch das LDMV2. Im rechten Drittel des Aufbaus befindet sich der Lastkreis, welcher an den Batterieprüfling angeschlossen werden kann. Über Kabelverbindungen ist das Frontpanel (in der Abb. 3.1 unten) angeschlossen.

3.3.1. Bedienschnittstellen

Das Frontpanel bietet mehrere Bedienschnittstellen und Anschlussmöglichkeiten. Das fertige Frontpanel ist in der Abb. 3.2 abgebildet:

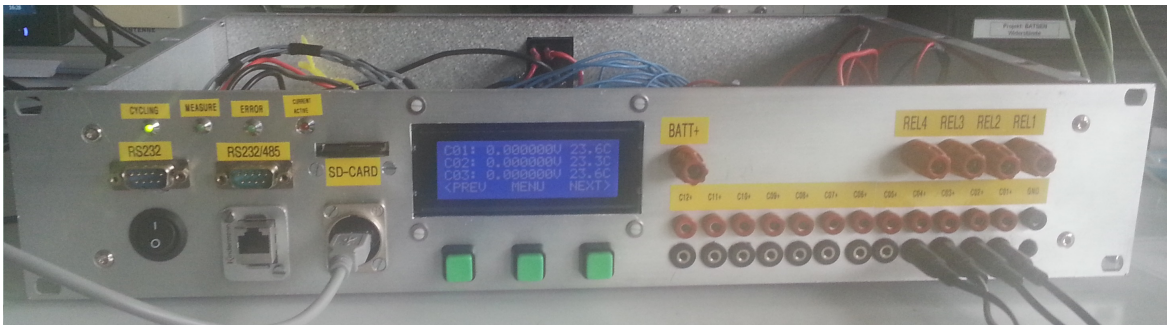


Abbildung 3.2.: Ansicht Frontpanel

Das Frontpanel wurde größten Teils von Herrn Andreé Schlüter angefertigt und nur geringfügig nachbearbeitet. Das Frontpanel ist in drei Bereiche unterteilt. Im linken Bereich (Abb. 3.3) befinden sich die Anschlüsse für Ethernet und USB, der SD-Karten Slot wie auch Busanbindungsmöglichkeiten für RS232 und RS485 über zwei D-SUB9 Anschlüsse. Mit dem Ein/Aus-Schalter kann das Gerät eingeschaltet werden und der aktuelle Status des Zyklierprüfplatzes wird über die vier LEDs angezeigt.

Abbildung 3.3.: Linker Bereich des Frontpanels

Im mittleren Bereich des Frontpanels (Abb. 3.4) befindet sich das LED-Display welches durch drei Taster gesteuert werden kann.

Abbildung 3.4.: Mittlerer Bereich des Frontpanels

Die Funktion der Taster wird dabei in der untersten Zeile des Display dargestellt. Wie die Software die Funktion der Taster steuert, wird in Kap. 4.4 dargestellt.

Im rechten Bereich des Frontpanels (Abb. 3.5) befinden sich die Anschlüsse für den Batterieprüfling und die Temperatursensoren. Die Temperatursensoren können über einen 3,5 mm Klinkenanschluss angeschlossen werden. Die einzelnen Zellen werden über Bananenstecker verbunden.

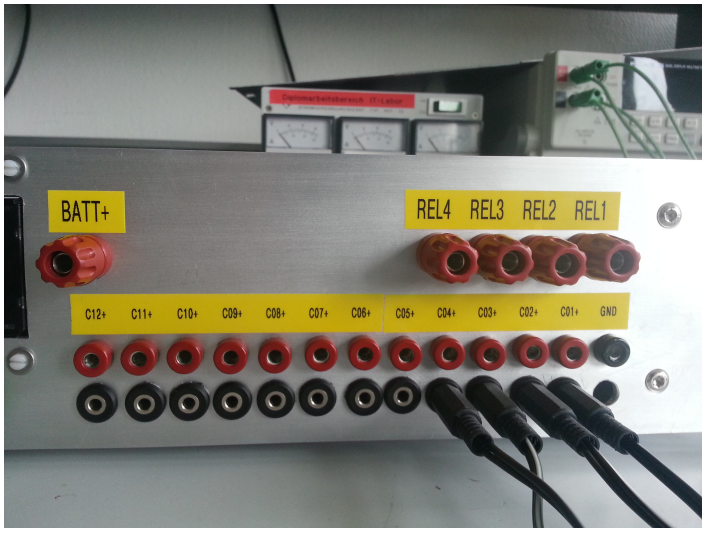


Abbildung 3.5.: Rechter Bereich des Frontpanels

Der Pluspol des Batterieprüflings kann über den Anschluss “BATT+“ an den internen Lastkreis des Gehäuses angeschlossen werden. Die Kontakte der Lastrelais werden an die Anschlüssen “REL1“ bis “REL4“ am Frontpanel geführt.

3.3.2. Netzteil und Notversorgung

Das gesamte Gerät wird über ein 12V-Netzteil mit 36 Watt Ausgangsleistung versorgt. Dieses befindet sich links oben in der Ecke (siehe Abb. 3.1). Der dazugehörige 230V-Anschluss an der Geräterückseite ist über eine Sicherung am Hauptschalter abgesichert. Der Anschluss einer Notlaufbatterie ist in diesem Aufbau zwar vorbereitet, jedoch wird im Rahmen dieser Abschlussarbeit keine Batterie installiert.

3.3.3. Lastkreis, Überstrom- und Verpolungsschutz

Gemäß Konzept in Kap. 2.2.8.1 wird der Lastkreis mit Lastrelais aufgebaut. Es wurden drei von den vier konzeptionierten Relais installiert. Der Einbau eines vierten Relais ist durch Bohrungen vorbereitet. Die Relais, Kabelverbindungen und Anschlusskontakte sind auf max. 20 A ausgelegt. Als Überstromschutz dient eine austauschbare 20 A Sicherung. Da es vorgesehen ist, dass nur ein Potential des Batterieprüflings an den Lastkreis der Zykliermaschine angeschlossen wird und der Strom bei Lade- bzw. Entladevorgängen bidirektional verlaufen kann, ist ein Verpolungsschutz nicht vorgesehen. Der HALL-Sensor für die Strommessung wurde in den Lastkreis integriert. Hierbei ist der Sensor für eine positive Stromrichtung ausgehend vom “BATT+“-Anschluss ausgelegt. Der Lastkreis ist in Abb. 3.6 dargestellt:

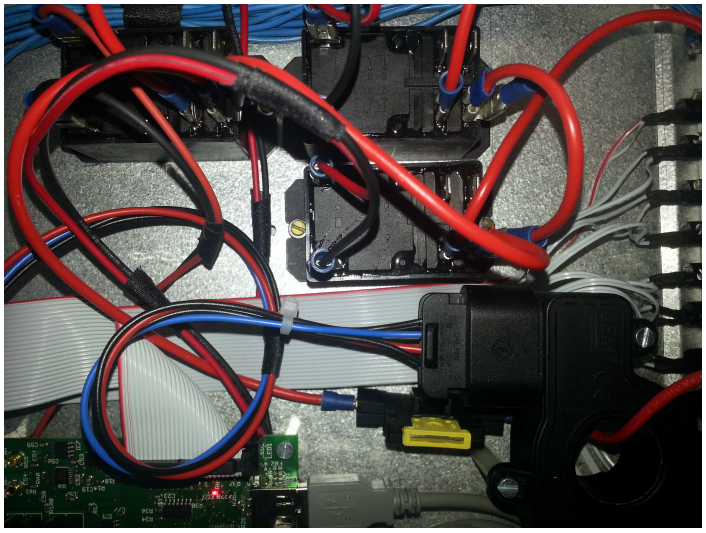


Abbildung 3.6.: Lastkreis im 19“ Rack

3.3.4. Aufbau des Gesamtsystems

Das 19“ Rack wurde in einen Metallschrank für 19“ Einschübe montiert. Zusätzlich wurden zwei Schubladenelemente eingebaut, in denen externe Peripherie, wie z.B. die Temperatursensoren, untergebracht werden können. Der Batterieprüfling kann auf dem Schrankboden (unter Verwendung einer Schutzwanne mit Deckel) oder außerhalb des Schrankes platziert werden. Lade- bzw. Entladegeräte können auf montierbaren Regalen platziert werden. Das Gesamtsystem zeigt Abb. 3.7:

Abbildung 3.7.: Aufbau Gesamtsystem

3.4. Inbetriebnahme

Eine erste Inbetriebnahme ist in sofern möglich, dass über die vorhandene JTAG-Schnittstelle der Controller direkt mit einem ICDI-Board programmiert wird. Damit kann mittels eines fertigen einfachen Programms der Controller mit angelegter Spannungsversorgung im Betrieb überprüft werden. Dieser Test verlief erfolgreich. Die Inbetriebnahme weiterer Peripherie erfordert die Implementation von Software und wird in Kap. 4 erläutert.

4. Konzeption, Entwurf und Implementation der Controllersoftware

4.1. Konfiguration der Programmierschnittstelle

Bevor die Controllersoftware nach und nach eingebunden werden kann, muss zuvor die integrierte Programmierschnittstelle in Betrieb genommen werden. Hierfür muss der entsprechende EEPROM, der an den FTDI-Chip angebunden ist, mit einer Basiskonfiguration geladen werden. Für diese Aufgabe wird das Programm FT Prog 2.6.8 benötigt, welches zum freien Download auf der Homepage von FTDI zur Verfügung gestellt wird. Es bietet die Möglichkeit, über eine grafische Oberfläche Einstellungen für die Konfiguration vorzunehmen. Weiterhin können diese Einstellungen in XML-Templates gespeichert und aus XML-Templates geladen werden.

Neben der Vergabe eines Namens und einer eindeutigen Seriennummer können die zwei vorhandenen Kanäle, oder auch Ports, konfiguriert werden, wie es bereits in Kap. 2.2.2 erwähnt wurde. Port A wird dabei als 245 FIFO mit dem D2XX Direct Treiber konfiguriert. Durch diese Einstellung wird der Port A zur Umsetzung von einer JTAG-Schnittstelle zu USB konfiguriert.

Port B wird so konfiguriert, dass die Umsetzung von UART zu USB stattfindet. So kann am PC ein virtueller COM-Port zu Verfügung gestellt werden, mit dem über ein Terminal-Programm eine Ein- und Ausgabe für den Mikrocontroller ermöglicht wird.

Damit der PC diese Funktionen unterstützen kann, müssen Treiber für die USB-Verbindungen auf dem Rechner installiert sein. Hierbei handelt es sich um die FTDI Treiber in der Version 2.06.00, welche direkt von der FTDI Homepage geladen werden können.

Nach vollständiger Durchführung aller Einstellungen kann die Konfiguration auf den EEPROM geladen werden. Eine detaillierte Beschreibung bietet die Bachelorarbeit von Jan Schlüter [15]. Ebenfalls kann die Bedienungsanleitung [10] von FTDI direkt zur Hilfe genommen werden.

4.2. Initialisierung der Peripherie

In diesem Kapitel soll die schrittweise Implementation der Software für die BCMV2 dargestellt werden. Die Controllersoftware wird dabei mit Hilfe der Entwicklungsumgebung Code Composer Studio v5 (CCSv5) von TI entwickelt. Teile der Controllersoftware greifen auf die herstellereigene Software-Bibliothek StellarisWare [32] von TI zu. Durch vordefinierte Makros lässt sich dadurch die Peripherie des Mikrocontrollers vereinfacht einbinden. Softwareimplementationen für Peripherien, wie z.B. die SD-Karte oder die Ethernetschnittstelle, basieren auf Beispielprogramm aus der umfangreichen Sammlung der Stellarisware. Die neu zu erzeugende Software in der Version 3.0 basiert in großen Teilen auf der Version 2.0 der BCMV1. Es gilt die Version 2.0 an die neue Hardware anzupassen und diese um Funktionen zu erweitern sowie im Gesamten zu optimieren.

Während des Initialisierungsvorganges (siehe Abb. 4.1) nach einem Neustart des Gerätes wird zunächst die Taktfrequenz eingestellt. Hierbei wird die Taktquelle auf die Phase-Locked Loop (PLL) festgesetzt, welche mit dem externen 16 MHz Quarz bis zu 80 MHz als Taktrate erzeugen kann. Die Taktrate für die BCMV2 wird auf 50 MHz eingestellt.

Die Wahl der Taktfrequenz ist durch die Nutzung von Timer-Funktionen bestimmt. Da die BCMV2 Messwerte in zeitlicher Abhängigkeit aufnehmen soll, spielen Timer eine dominante Rolle zur Steuerung der Messdatenerfassung. Die Zählregister der Timer sind dabei 32-bit breit. Einer der Timer ist so konzeptioniert, dass dieser jede Minute einen Interrupt ausführen soll. Da die Zählregister in Abhängigkeit der Taktrate des Mikrocontrollers gefüllt werden, muss das sechzigfache der Taktrate im Zählregister Platz finden. In der Tab. 4.1 sei durch die Gl. 4.1 dargestellt, welche maximalen Zählwerte (in Sekunden) der Timer in Abhängigkeit der Taktrate erreicht werden können:

Takt	Maximale Zeit (sec)
50 MHz	85,89
66 MHz	64,42
80 MHz	53,68

Tabelle 4.1.: Auswahl Taktfrequenz in Abhängigkeit des langsamsten Timers

$$t_{max} = \frac{2^{32}}{Takt} \tag{4.1}$$

Eine Umstellung der Taktrate auf 80 MHz ist demnach ohne weiteres nicht möglich, da der langsamste Timer nie eine Minute erreichen könnte und das Zählregister vorzeitig überlaufen würde. Die Wahl von 50 MHz ist für die Anwendung ausreichend. Bei Bedarf kann die Taktrate auch auf 66 MHz gesetzt werden.

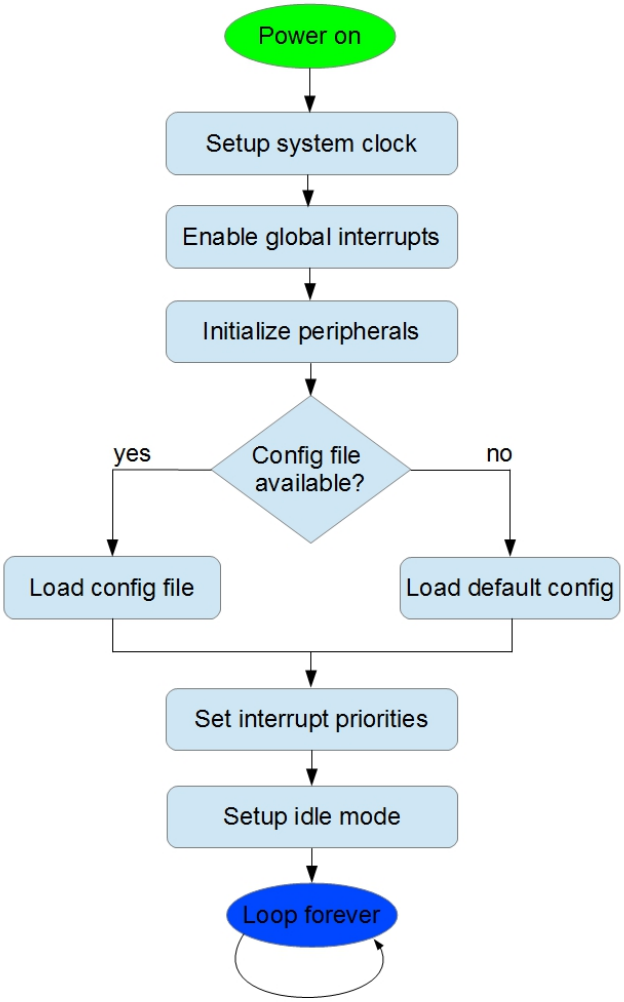


Abbildung 4.1.: Initialisierungsablauf nach Einschalten des Gerätes

Nach Einstellung der Taktrate werden alle internen und externen Komponenten, welche an den Mikrocontroller angebunden sind, initialisiert. Zu Beginn wird die UART-Schnittstelle initialisiert, damit bereits während der Initialisierung mit einem Terminal-Programm auf einem PC (z.B. hTerm) eine Ausgabe des Initialisierungsvorganges stattfinden kann. Da einige Komponenten bereits während der Initialisierung Interrupts benötigen, wird bereits vor der Initialisierung einzelner Komponenten die globale Interruptsteuerung aktiviert. Die chronologische Anordnung der Initialisierungsschritte vermeidet dabei, dass Interrupts zu ungünstigen Zeitpunkten die gesamte Initialisierung stören könnten. Während der Initialisierung wird auch überprüft, ob sich ein Konfigurationsdatei für die Zyklenablaufsteuerung auf der SD-Karte befindet.In diesem Fall werden die Daten ins Programm geladen. Ist keine Konfigurationsdatei vorhanden, so werden Standardwerte, welche für weitere Initialisierungsschritte erforderlich sind, geladen. Die Auswertung und genauere Erläuterung der Konfigurationsdatei erfolgt in Kap. 4.3.1.

Zuletzt werden noch die Prioritäten der Interrupts festgesetzt. Die höchste Priorität hat der Systick-Timer. Dieser tritt alle 10 ms auf. Dieser wird benötigt, um das FAT-Filesystem der SD-Karte sowie eine vorhandene Ethernetverbindung aufrecht zu erhalten. Eine weitere Funktion dieses Timers ist die permanente Erhöhung der Systemuhrzeit.

An zweiter Stelle der Prioritätenliste sind die Interruptquellen untergebracht, welche für die zyklische Messdatenerfassung notwendig sind. An letzter Stelle sind die nebensächlichen Interrupts untergebracht. Im Wesentlichen sind dies die Interrupts, die das LED-Display samt Tastern steuern, wie auch die interruptgestützte Ausgabe über Ethernet.

Dieses Priorisieren der Interruptquellen ist wichtig, da sich das BCMV2 nach dem Initialisierungsprozess im “Idle Modus“ befindet. Ab hier läuft das gesamte Programm Interrupt gesteuert.

4.3. Zyklensteuerung

In diesem Kapitel soll dargestellt werden, wie die Zyklenablaufsteuerung konfiguriert werden kann, wie im laufenden Betrieb Einstellungen geändert werden und Messvorgänge gestartet werden können. Weiterhin soll erläutert werden, wie die Zyklenablaufsteuerung auf der Mikrocontrollerebene abläuft.

4.3.1. Konfigurationsdatei

Eine Funktion die Zyklusablaufsteuerung zu konfigurieren ist die Einbindung einer Konfigurationsdatei, in welcher Konfigurationsparameter im Textformat hinterlegt sind. Diese Konfigurationsdatei wird während des Initialisierungsvorganges (Kap. 4.2) von der SD-Karte, falls vorhanden, gelesen. Durch Parsen des Textflusses werden die Konfigurationsparameter im Programm beschrieben.

Die Konfigurationsdatei kann komfortabel an einem PC mit einem Text-Editor (empfohlen wird das Tool Notepad++) bearbeitet werden. Gemäß der Abb. 4.2 können innerhalb der Konfigurationsdatei folgende Einstellungen eingegeben werden:

1. Die Anzahl der zu messenden Zellen (Quantity of Cells, dezimal zweistellig)
2. Zeitabstand zwischen den Messvorgängen in Sekunden (Cyclestep, vierstellig)
3. ADC Verstärkungsfaktor je Kanal (ADC Gain in $\frac{pV}{bit}$, achtstellig)
4. ADC Offsetwert je Kanal (ADC Offset in μV, sechsstellig)
5. Die IP-Adresse für die Ethernet Schnittstelle (IP Address, jedes Oktet dreistellig)
6. Die gesamte Dauer der Messung in Minuten (Measuretime, fünfstellig)
7. Anzahl der Zeitumschaltpunkte der Lastrelais (Relay Quantity, zweistellig)
8. Lastrelaisumschaltzeitpunkt in Minuten nach Programmstart (Relaytime, fünfstellig)
9. Zu schaltendes Lastrelais zum Umschaltzeitpunkt (Relayselect, zweistellig)

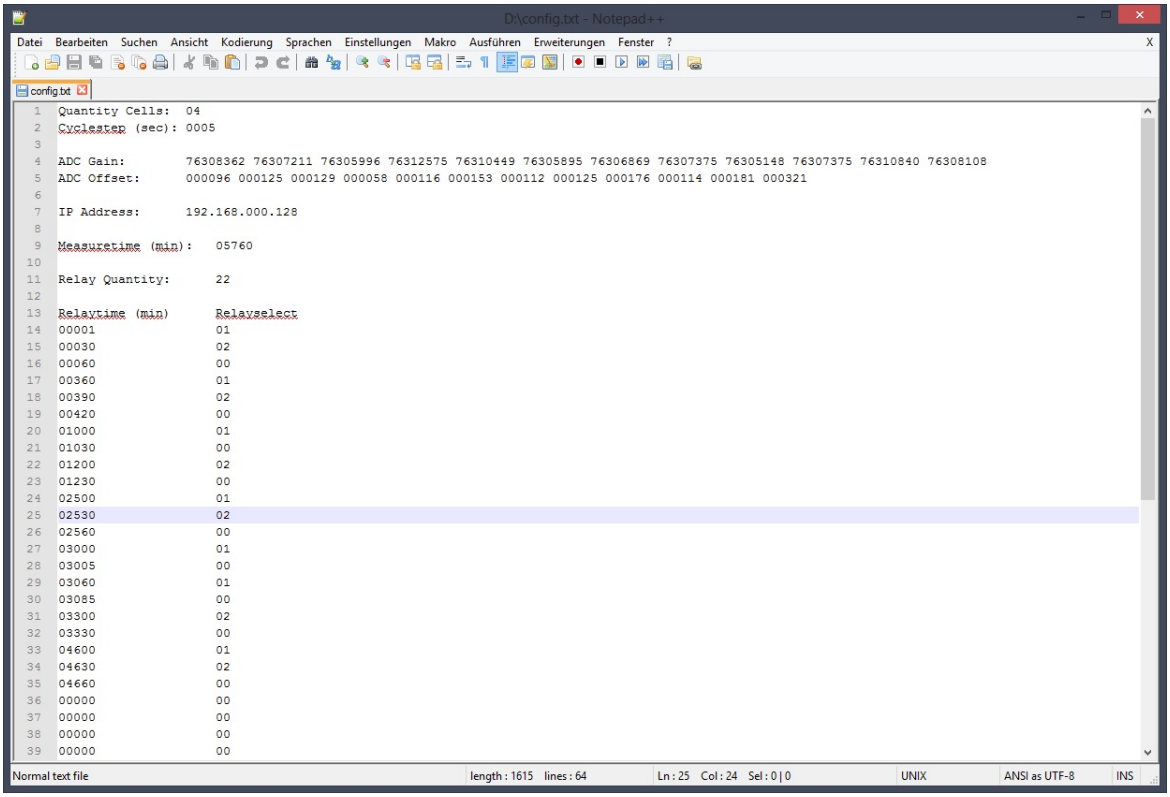


Abbildung 4.2.: Format der Konfigurationsdatei

Für das Parsen der Konfigurationsdatei ist es wichtig, das Format in Abb. 4.2 Format beizubehalten. Historisch bedingt ist der Controllerquellcode zum Parsen einfach gestaltet. Zu Beginn des Projekts sollten hierdurch nur zwei Parameter eingelesen werden. Im Laufe des Projekts ist aber die Anzahl der Einstellungen gestiegen, die Methode zum Parsen wurde jedoch nicht geändert. Es ist erforderlich, dass die Positionen der einzulesenden Parameter und Beschreibungen in der Konfigurationsdatei nicht geändert werden. Die Konfigurationsdatei muss im Stammverzeichnis der SD-Karte unter dem Namen **“config.txt“** hinterlegt werden. Ist eine Konfigurationsdatei nicht vorhanden, so kann diese vom Mikrocontroller mit Standardwerten erzeugt werden. Die kanalabhängigen Kalibrierdaten für den externen ADC sind in der Controllersoftware hinterlegt und werden in die Konfigurationsdatei abgespeichert. Restliche Standardwerte werden mit “0“ gefüllt.

4.3.2. Dialogeinstellung

Über dieUART-Schnittstelle lässt sich die BCMV2 mit einem Terminal Programm, wie z.B. hTerm, über Befehle steuern. Dabei muss das Terminal mit folgenden Einstellungen (Tab. 4.2) versehen werden:

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

Tabelle 4.2.: Einstellungen UART-Schnittstelle

Menü Main

Im Idle Modus befindet sich die Eingabemaske im Menü “Main“. Im Terminal Programm wird das aktuelle Menü (Abb. 4.3) angegeben. Über den Befehl **“help“** kann eine Auflistung der Befehle angefordert werden. Die im Menü Main hinterlegten Befehle sind in der Tab. 4.3 hinterlegt:

```
************************************************
***         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 4.3.: Terminalausgabe im Verzeichnis ’Main’

Befehl	Beschreibung
help	Listet alle möglichen Befehle auf
h	Abkürzung für 'help'
?	Abkürzung für 'help'
browser	Aufrufen des Kontextmenüs SD-Karten Browser
config	Aufrufen des Konfigurationsmenüs
alive	Abfrage, ob Zyklusmessung aktiv
start	Start der Zyklusmessung
stop	Manueller Stop der Zyklusmessung

Tabelle 4.3.: Befehlssatz im Kontextmenü 'Main'

Mit den Befehlen **"browser"** und **"config"** kann in die entsprechenden Untermenüs mit eigenem Befehlssatz gesprungen werden. Zwei primäre Befehle in diesem Menü sind die Befehle **"start"** und **"stop"**, mit welchen die zyklische Messdatenerfassung gestartet werden kann. Mit dem Befehl **"alive"'** ist auf dieser Ebene eine Möglichkeit eingebracht, mit der überprüft werden kann, ob aktuell eine Messdatenerfassung aktiv ist.

Für die Erkennung und Auswertung eines Befehls einer Terminaleingabe gibt es eine eigene Interruptroutine, welche in Abb. 4.4 dargestellt wird:

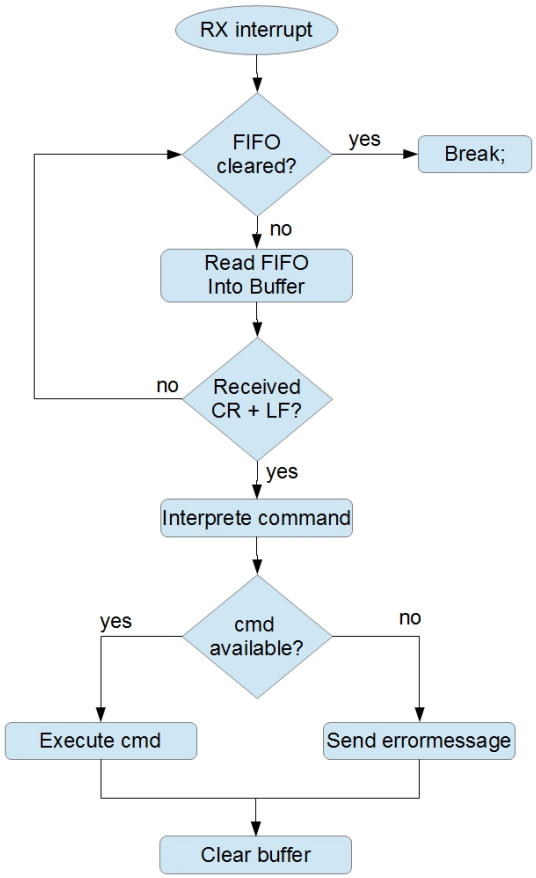


Abbildung 4.4.: Befehlsauswertung über UART und Terminal

Wird erkannt, dass ein Befehl ankommt, wird dieser aus dem 16 Byte breiten First In - First Out (FIFO)-Speicher der UART-Schnittstelle in einen Puffer eingelesen. Mit jedem eingelesen Byte wird überprüft, ob die Terminierungszeichen, welche mit dem Befehl mitgeschickt werden, erreicht worden sind. Ist der FIFO geleert und wurde kein Terminierungszeichen erreicht, so beendet sich der Interrupt. Der FIFO wird wieder beschrieben und der Interrupt löst erneut aus. Nach Erreichen eines kompletten Befehls wird überprüft, ob der eingegebene Befehl im Befehlssatz vorhanden ist. Falls ja, so wird dieser ausgeführt. Sonst wird eine Error-Benachrichtigung ausgegeben. Anschließend wird der Puffer wieder geleert.

Menü SD-Karten Browser

Befehl	Beschreibung
help	Listet alle möglichen Befehle auf
h	Abkürzung für 'help'
?	Abkürzung für 'help'
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'

Tabelle 4.4.: Befehlssatz im Kontextmenü 'SD-Card'

Im Menü SD-Karten Browser liegt ein Befehlssatz vor (Tab. 4.4), mit dem die Daten auf der SD-Karte verwaltet werden können. Mit dem Befehl **"ls"** können alle vorhandenen Dateien auf der SD-Karte über UART ausgegeben werden. Ein Beispiel zeigt Abb. 4.5:

```
 Enter Command (type <help> to see list of commands):
SD-Card: > ls
----A 2013/06/10 12:35       1615  config.txt
----A 2013/06/10 12:37       3741  06101237.txt

   2 File(s),       5356 bytes total
   0 Dir(s),     1925136K bytes free

 Enter Command (type <help> to see list of commands):
SD-Card: >
```

Abbildung 4.5.: Terminalausgabe im Verzeichnis 'Browser'

Mit weiteren Befehlen können die Dateien auf der SD-Karte verwaltet werden. So können auch Dateien erstellt und gelöscht werden. Der Inhalt einer Datei kann über den Befehl **"cat"** ausgelesen werden. Dieser Befehl sollte jedoch bei besonders großen Dateien nicht im laufenden Zyklierbetrieb ausgeführt werden, da die Ausgabe über UART viel Zeit in Anspruch nehmen kann und unter Umständen die zyklische Messdatenerfassung verzögert.

Menü Config

Im Konfigurationsmodus kann mit dem entsprechenden Befehlssatz in Tab. 4.5 die BCMV2 manuell konfiguriert werden.

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 (01 - 12)
cyclestep	Setzte Zeitabstand zwischen zyklischer Messung in Sekunden (0000-9999)
gain	Setze kanalabhängig Verstärkungsfaktor für ADC (z.B. 'gain 01 76273500')
offset	Setze kanalabhängig Offset für ADC (z.B. 'offset 01 000120')
ip	Setze IP-Adresse ('192.168.000.128') ⇒ Neustart erforderlich
relais	Manuell Lastrelais aktivieren (z.B. 'relais 1')
relsel	Wähle für die Ablaufsteuerung an entsprechender Stelle und Zeitpunkt ein Lastrelais(z.B. 'relsel 00 00001 1')
relque	Setze die Anzahl der Lastrelaisumschaltzeiten (01 - 99)
time	Setze Laufzeit der gesamten Messung in Minuten (0-99999)
save	Speichere aktuelle Konfiguration auf die SD-Karte (überschreibt config.txt)
exit	Zurück zum Kontextmenü 'Main'

Tabelle 4.5.: Befehlssatz im Kontextmenü 'Config'

Dabei können sämtliche Parameter geändert werden die auch über die Konfigurationsdatei auf der SD-Karte eingelesen werden. Über den Befehl **"print"** wird die aktuelle Konfiguration der BCMV2 über UART ausgegeben. Über die Funktion **"save"** kann die aktuelle Konfiguration auf der SD-Karte gespeichert werden. Die vorhandene **"config.txt"** wird dabei überschrieben. Eine Änderung der IP-Adresse erfordert einen Neustart des Zykliergerätes, um die Ethernet-Schnittstelle neu zu initialisieren.

4.3.3. Messdatenerfassung

Ist die BCMV2 vollständig konfiguriert, so kann die Messung über UART (Kap. 4.3.2), über das LED-Display (Kap. 4.4) oder Ethernet (Kap. 4.5.2) gestartet werden.

Es wird automatisiert eine neue Log-File auf der SD-Karte generiert. Auf Grund des vorliegenden FAT-Filesystems in der Version 0.4 ist ein Dateiname im Standardformat Short File Name (SFN) gefordert. Die Dateinamen müssen demnach im Format 8.3 gewählt werden, also acht Buchstaben oder Ziffern, gefolgt von einem Punkt und anschließender Dateiendung mit drei Zeichen. Für die Messdaten wird ein Log-File mit dem aktuellem Datum zum Startzeitpunkt im SFN-Format erzeugt:

“MMDDHHMM.txt“

Zu Beginn wird der Log-File ein Header fester Länge hinzugefügt, in welchem die eingestellten relevanten Konfigurationsparameter der Messung wie auch die genaue Uhrzeit zu Beginn der Messung zusammengefasst werden (Abb. 4.6):

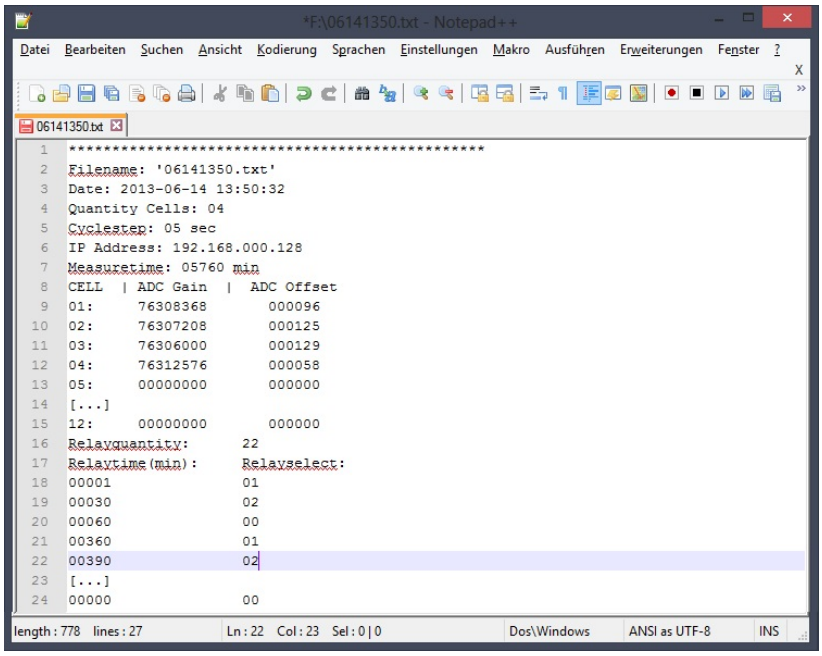


Abbildung 4.6.: Header der erzeugten Log-File

Mit dem Ausführen des Startbefehls werden weiterhin die für zyklische Messdatenerfassung nötigen Peripherien aktiviert. Das Flussdiagramm in der Abb. 4.7 soll dies veranschaulichen:

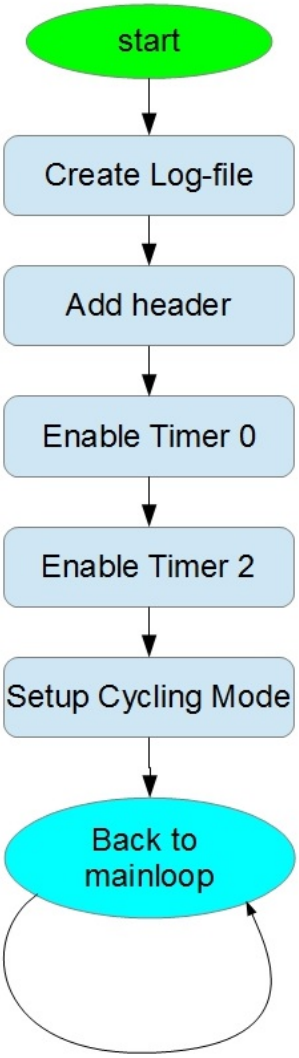


Abbildung 4.7.: Aktivierung zyklische Messdatenaufnahme

Der Timer 0 (aus Abb. 4.7) wird dabei so initialisiert, dass dieser zyklisch bzw. periodisch in Abhängigkeit des eingestellten Zeitabstandes “Cyclestep“ auftritt. Mit diesem Interrupt wird die ganze Messdatenerfassung ausgeführt. Dieser Interrupt hat die höchste Priorität. Innerhalb dieses Interrupts wird weitere Peripherie angesprochen. Über die Abb. 4.8 wird erläutert, wie mittels dieses Interrupts die eigentliche Messdatenerfassung erfolgt:

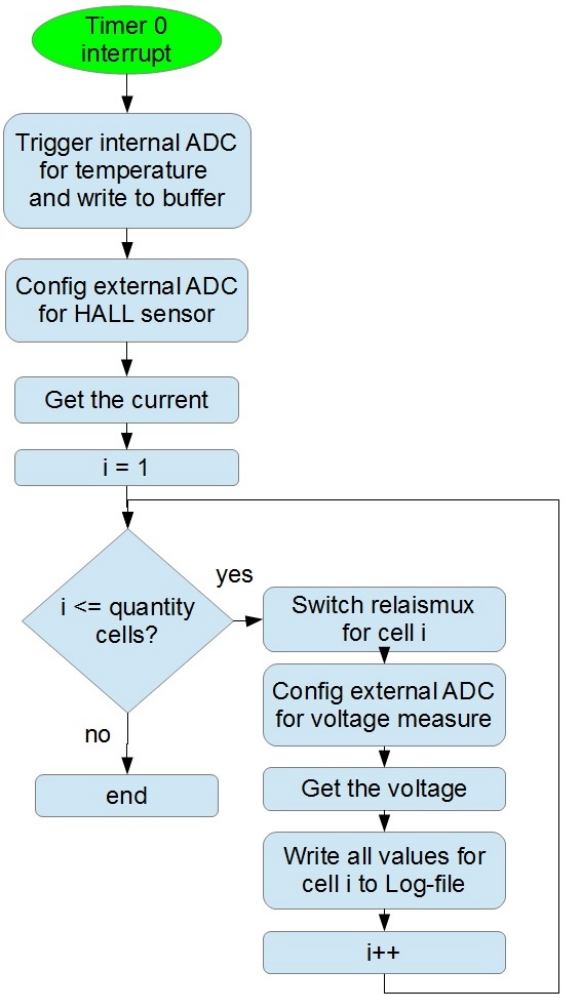


Abbildung 4.8.: Ablauf der Messdatenaufnahme

Der Interrupt des Timers 0 triggert die internen ADC des Mikrocontroller an, die mit den NTC-Temperatursensoren verbunden sind. Die Temperaturwerte werden in einem globalen Array zwischengespeichert. Nachfolgend wird der externe ADC zum Auswerten des HALL-Sensors konfiguriert und der Strom wird gemessen und in einer globalen Variable gespeichert. In Abhängigkeit der eingestellten Zellenanzahl mittels des Konfigurationsparameter “quantity_cells“ läuft nun zellenweise die Spannungsmessung jeder einzelnen Zelle. Dafür wird der Relaismux entsprechend der geforderten Zelle geschaltet, der ADC für die Spannungsmessung konfiguriert, die Spannung gemessen und direkt in die Log-File des Messvorganges mit Strom und Temperatur sowie Uhrzeit gemäß Kap. 2.2.5 gespeichert. Nach Messung aller Zellenwerte wird der Interrupt beendet.

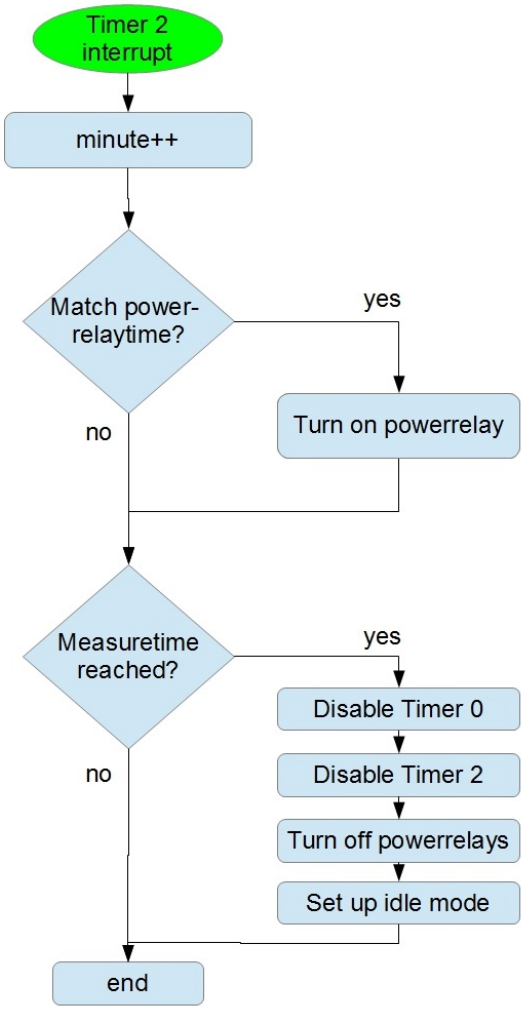


Abbildung 4.9.: Ablauf Lastkreissteuerung

Über den Timer 2 Interrupt, welcher jede Minute erfolgt, kann der Lastkreis gesteuert werden. Hierbei arbeitet die BCMV2 mit der LDMV2 zusammen. Der Ablauf des Interrupt ist in der Abb. 4.9 dargestellt. Dieser Interrupt wertet die in der Konfigurationsdatei eingestellten Umschaltzeitpunkte aus und schaltet bei Übereinstimmung mit der aktuellen Zeit das entsprechende Lastrelais. Die Priorität dieses Interrupt ist so eingestuft, dass eine laufende Messung nicht unterbrochen wird. Weiterhin wird mit diesem Interrupt kontrolliert, ob die eingestellte Messdauer “Measuretime“ abgelaufen ist. In diesem Fall werden die für die Zyklenablaufsteuerung entsprechenden Timer deaktiviert, alle Lastrelais getrennt und die BCMV2 in den Idle Modus versetzt.

4.3.4. Dauer der Messdatenerfassung (“worst case“)

Die kleinst einstellbare Zykluszeit (“Cyclestep“) beträgt eine Sekunde. Es muss also gewährleistet werden, dass eine einzelne Messdatenerfassung die Dauer von einer Sekunde nicht überschreitet, bzw. auch nötigen Puffer an Zeit für weitere Aktionen neben der eigentlichen Messdatenerfassung zwischen zwei Zyklen zulässt.

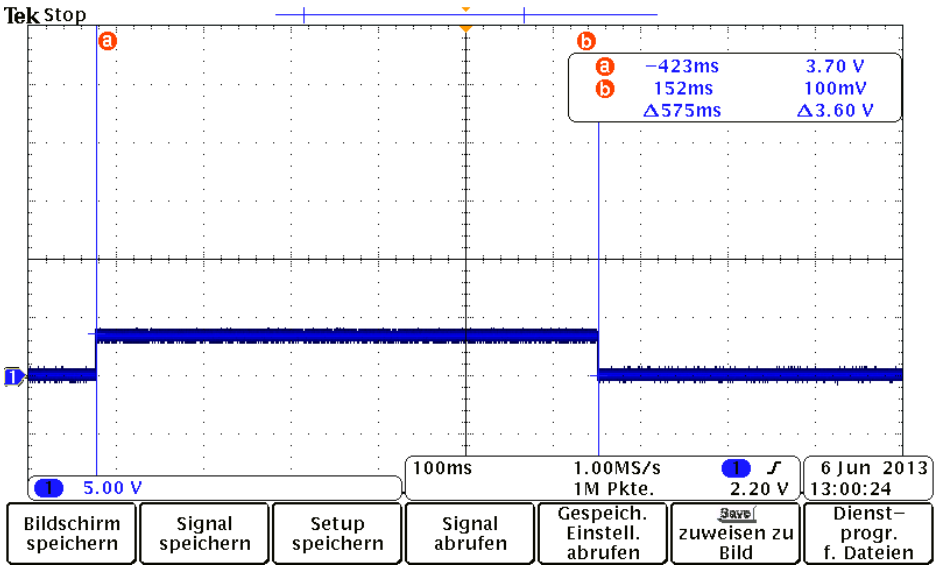


Abbildung 4.10.: Dauer Messdatenerfassung “worst case“

Anhand der Abb. 4.10 wurde die benötigte Zeit der Messdatenerfassung des “worst case“-Falles mit einem Oszilloskop ermittelt. Der worst case-Fall tritt auf, wenn eine Messung von 12 angeschlossenen Zellen erfolgen soll. Für die Messung einer Zelle wird eine bestimmte Zeit benötigt, worauf sich die benötigte Gesamtzeit aufsummiert. Dabei spielen auch weitere Funktionen eine Rolle, die das Schalten des Relaismultiplexers und dem Beschreiben der SD-Karte entsprechend Abb. 4.8 beanspruchen.

Mit dem auf der Platine der BCMV2 vorhandenen Trigger-Ausgang ist eine Möglichkeit vorhanden, einen GPIO-Pin auszuwerten. Somit kann die Zeit zwischen Aktivieren des Pins zu Beginn der Messdatenerfassung, und dem Deaktivieren des Pins im Anschluss der Messdatenerfassung wie in Abb. 4.10 ermittelt werden. Hieraus ergibt sich eine ermittelte Zeit von 575 ms. Diese Zeit liegt unter einer Sekunde und bietet auch Raum für das Abhandeln weiterer Software auf dem Mikrocontroller.

Unter Umständen kann diese Zeit reduziert werden. Die meiste Zeit benötigt der externe 16 Bit ADC [3], der mit einer internen Taktrate von 123 Hz seine Konversionen durchführt. Die Taktrate lässt sich bis zu 470 Hz erhöhen. Dies zieht jedoch den Effekt nach, dass sich die Genauigkeit der Messung durch erhöhtes Rauschen verschlechtert, was die Anforderungen an das Zykliergerät nicht erfüllen würde. Weiterhin könnte die Wartezeit verringert werden, welche nach Schalten des Relaismultiplexers erfolgt. Durch diese softwarebasierte Wartezeit soll gewährleistet werden, dass die Spannung am ADC ausreichend genau einschwingt. Pro Zelle beträgt diese Wartezeit 10 ms. Bei zwölf Zellen werden also 120 ms gewartet. Auch hier ist es aber wichtig, die Genauigkeit der ADC-Messung nicht zu vernachlässigen. Eine Verringerung der Wartezeit ist möglich, wird aber nicht empfohlen.

4.4. Steuerung über das LED-Display

Das LED-Display am Frontpanel bietet die Möglichkeit einer Ausgabe von Informationen über den Zustand der BCMV2. Über drei Taster kann dabei die Anzeige gesteuert werden. Die LED-Anzeige wird über einen Timer mit einer Frequenz von 15 Hz aktualisiert. Über einen Zustandsautomaten, der seine Zustände (Tab. 4.6) in Abhängigkeit der Taster ändert, wird ein kleines Informationsmenü erzeugt:

Zustand	Anzeige
START	Startbildschirm der BCMV2
CLOCK	Anzeige der Systemuhrzeit
MEAS	Status des Cycling Mode
CELLA	Wiedergabe aktuelle Werte Zelle 1-3
CELLB	Wiedergabe aktuelle Werte Zelle 4-6
CELLC	Wiedergabe aktuelle Werte Zelle 7-9
CELLD	Wiedergabe aktuelle Werte Zelle 10-12
CURRENT	Aktueller Strom im Lastkreis
IP	Anzeige der initialisierten IP-Adresse
CONFIG	Übersicht wichtigster Konfigurationsparameter und Restzeit

Tabelle 4.6.: Zustände des Automaten für die LED-Anzeige

In der Abb. 4.11 ist das Zustandsdiagramm für das LED-Display zu sehen. Über die Taster **B1** bis **B3** kann durch die einzelnen Anzeigen geblättert werden. Im Zustand “MEAS“ kann über den linken Taster die Zyklenablaufsteuerung gestartet werden und über den rechten Taster bei Bedarf manuell gestoppt werden. Der Befehl “STOP“ muss mit “YES“ bestätigt werden, oder bei ungewolltem Betätigen von “STOP“ der Messdatenaufnahme mit “NO“ negiert werden.

Wie die einzelnen Anzeigen der Zustände dargestellt sind, kann in den Abbildungen 4.12a bis 4.12g nachvollzogen werden. Innerhalb eines jeden Zustandes steht in der untersten Display-Zeile immer die Funktion des jeweiligen Tasters.

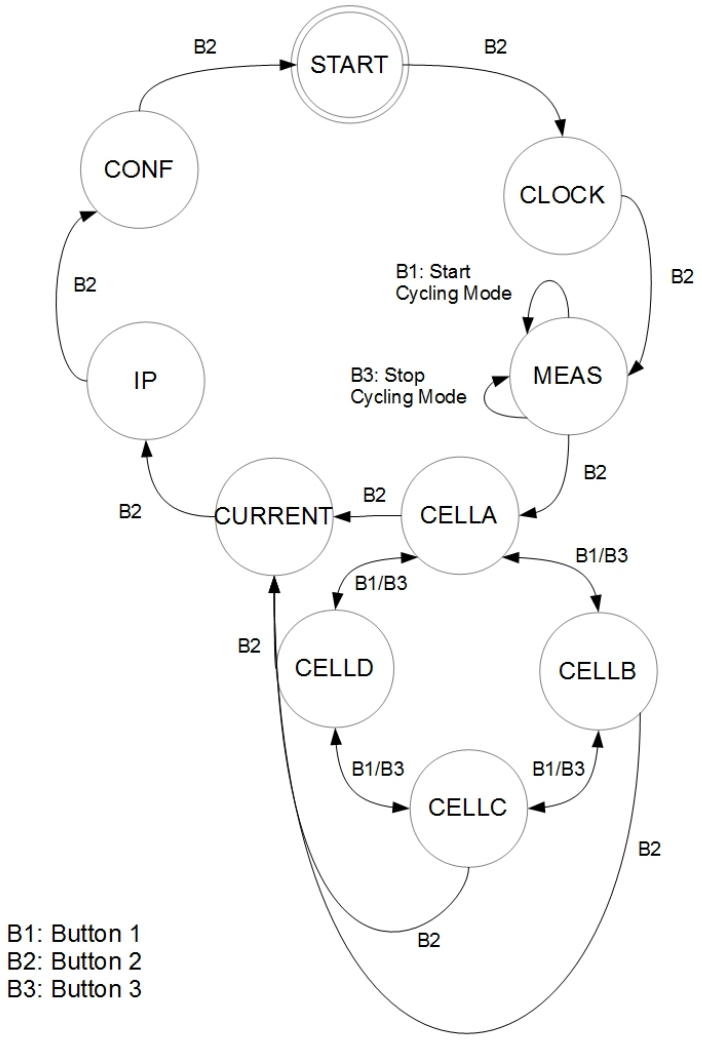


Abbildung 4.11.: Zustandsdiagramm des LED-Displays

(a) LED-Zustand: START

(b) LED-Zustand: CLOCK

(c) LED-Zustand: MEAS

(d) LED-Zustand: CELLA

(e) LED-Zustand: CURRENT

(f) LED-Zustand: IP

(g) LED-Zustand: CONF

Abbildung 4.12.: Zustandsbedingte Ausgabe LED-Display

4.5. Ethernet TCP-IP und MatLab

Um einen Fernzugriff auf die BCMV2 zu ermöglichen, wird softwaretechnisch Ethernet über den Lightweight TCP/IP stack (lwIP) eingebunden. Die entsprechende Ethernetbuchse ist auf der Platine der BCMV2 wie auch am Frontpanel des Zykliergerätes vorhanden. Über Ethernet soll ermöglicht werden, das System über ein LAN zu steuern als auch Messdaten von der SD-Karte auszulesen.

4.5.1. Einbindung von lwIP auf Controllerebene

Der lwIP Stack eignet sich durch seinen reduzierten Umfang besonders gut für eingebettete Systeme, ohne Vernachlässigung des TCP/IP Protokolls. Die Implementierung des lwIP in der Controllersoftware wurde größtenteils aus der Arbeit von Tobias Steinmann [16] übernommen und kann dort in Kap. 4.2. detailliert nachgelesen werden. Das Ethernetinterface wird während des Initialisierungsvorganges aufgesetzt. Dabei wird eine MAC-Adresse benötigt, die in den Mikrocontroller LM3S9D92 [29] mit dem Tool LMFlashProgrammer von TI programmiert werden kann. Für Laborzwecke wurde eine willkürliche 48 Bit lange MAC-Adresse generiert und einprogrammiert. Sollte die BCMV2 vermarktet werden, ist es unumgänglich, für jedes Produkt eine offizielle weltweite eindeutige MAC-Adresse beim Institute of Electrical and Electronics Engineers (IEEE) zu beantragen. In dieser Bachelorarbeit wurde die auf die Beantragung einer solchen MAC-Adresse verzichtet. Beim Initialisieren wird auch die IP-Adresse, welche in der Konfigurationsdatei im Kap. 4.3.1 definiert wurde, dem Ethernetinterface zugeordnet.

4.5.2. MatLab

Mit dem Programm MatLab kann über fertige Sourcefiles eine TCP/IP Verbindung mit der BCMV2 hergestellt werden. Weiterhin liegen Sourcefiles vor, die eine Steuerung des Zyklierprüfstandes über Funktionen ermöglichen. Die entsprechenden Sourcefiles für MatLab sind im Anhang E zu finden.

4.5.2.1. Verbindungsaufbau

Mit der Funktion $connect(ip)$ kann mit MatLab eine direkte TCP/IP-Verbindung zur BCMV2 mittels der IP aufgebaut werden. Dabei wird auf der Controllerebene das Empfangen einer Verbindungsanfrage nach Tobias Steinmann [16] wie in der Abb. 4.13 abgehandelt. Der lwIP Stack wird mit dem Systick-Timer alle 10 ms aufgerufen.

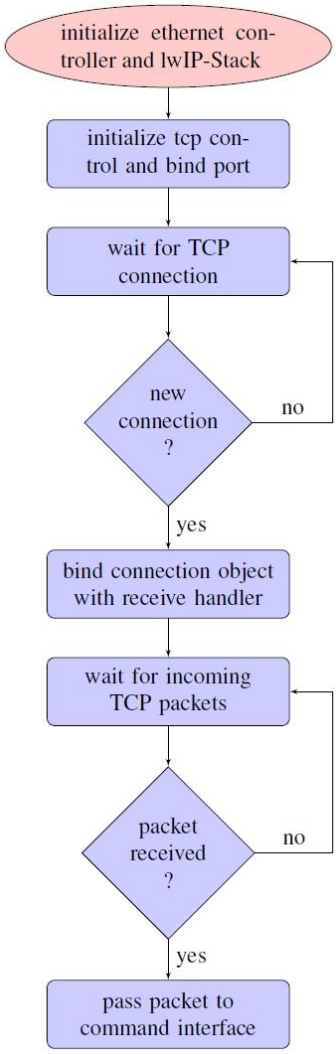


Abbildung 4.13.: Handling einer TCP-Verbindung (Quelle: T. Steinmann[16])

4.5.2.2. Steuerung

Nach erfolgreichem Verbindungsaufbau ist es nun möglich, die BCMV2 über Ethernet zu steuern. Dabei können mit MatLab mit der Funktion $send\_cmd(cons, sprintf('cmd\backslash n'))$ Befehle gesendet werden. Um die Bedienung zu erleichtern, werden mit dieser Steuermöglichkeit dieselben Befehle wie über UART und einem Terminalprogramm (Kap. 4.3.2) auf dem Rechner ausgeführt. Die Befehlssätze sind in den Tabellen 4.3, 4.4 und 4.5 ersichtlich. Entsprechend der Abb. 4.13 werden die TCP/IP-Pakete erkannt und an das Command Interface übergeben. Im Command Interface wird, gleichermaßen wie in der Abb. 4.4 über UART, der Befehl interpretiert und entsprechend ausgeführt. Dabei wird über TCP/IP dieselbe Ausgabe generiert wie über UART und in der Konsole von MatLab ausgegeben. MatLab wertet dabei die TCP/IP-Pakete über Eventhandling aus, in dem beim Eintreffen eines Paketes die Funktion $receive\_cmd(cons, event)$ ausgeführt wird.

4.5.2.3. Messdatenauswertung

Die erfassten Messdaten auf der SD-Karte sollen später in Diagrammen mittels MatLab dargestellt werden. Eine Option ist die Log-File auf der SD-Karte mit einem Rechner, auf dem MatLab installiert ist, direkt auszuwerten. Angedacht ist alle erfassten Daten zellenweise in zeitlicher Abhängigkeit zu plotten. Das hierfür entsprechende Skript befindet sich im Anhang E. Im Kap. 5.1.2 soll die Auswertung anhand eines realen Versuchsbeispiels erläutert werden.

Zusätzlich soll es möglich sein, die Messdaten per Fernzugriff über Ethernet direkt in MatLab zu laden, ohne die SD-Karte dabei aus dem Zykliergerät zu entfernen. Hierfür kann der Befehl **“cat“** über Ethernet genutzt werden, um den Inhalt der Log-File zunächst auszulesen. Bei der Implementierung dieser Funktion kam es hierbei zu Komplikationen bis zur schriftlichen Fertigstellung dieser Bachelorarbeit. In Testversuchen dieser Funktion konnte beobachtet werden, dass der Inhalt einer großen Datei nicht vollständig über lwIP wiedergegeben wird. Diesen Sachverhalt gilt es anhand der Controllersourcefiles des lwIP im Nachgang der Bachelorarbeit zu untersuchen.

4.6. Kalibrierung

4.6.1. Kanalabhängige Kalibrierung des externen ADC

Der 16-Bit ADC muss für eine möglichst genaue Messung kalibriert werden. Wie bereits in Kap. 2.1.2.1 erwähnt, ist eine Messgenauigkeit von 1 mV angestrebt. Der verwendete ADC [3] bietet hierfür eigene Register für Verstärkungsfaktor und Offset. Aufgrund des vorgeschalteten Relaismultiplexers ist eine hardware-konfigurierte Kalibrierung jedoch nicht sinnvoll. Daher erfolgt die Kalibrierung des ADCs in Software, so dass für jeden Kanal eigene Kalibrationswerte individuell festgelegt werden können.

Da die Kalibrierung in der Software erfolgt, wird an dieser Stelle untersucht, ob die Genauigkeit einer linearen Regression durch Verwendung einer möglichen quadratischen Regression verbessert werden kann. Als Referenzspannungsmessgerät wurde ein Labormultimeter des Herstellers Fluke [8] verwendet.

Abb. 4.14 zeigt die mit dem ADC gemessene Spannung in Relation der mit dem Fluke Labormultimeter gemessenen Spannung. Im oberen Diagramm ist der ADC noch nicht kalibriert. Im unteren Diagramm wurde der ADC über den vollständigen Messbereich in mehreren Quantisierungsstufen linear kalibriert. Hierbei wurden pro angelegter Spannung mehrere Werte des ADC aufgezeichnet und danach ausgewertet. Über den Soll-Ist-Vergleich wurden darauf Verstärkungsfaktor und Offset kalkuliert. Die Punkte liegen optisch dicht übereinander. Der Temperatureinfluss wurde in der Kalibration nicht berücksichtigt.

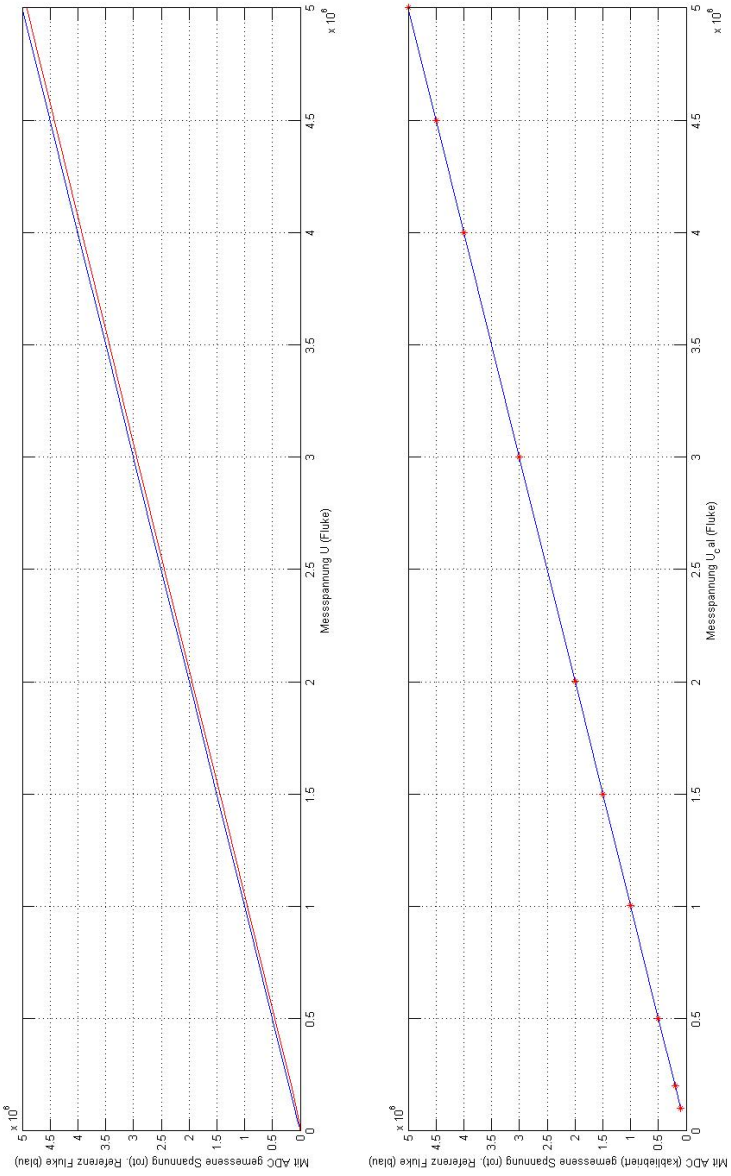


Abbildung 4.14.: Software basierte Kalibrierung des ADC für einen Kanal

Über die Funktion “Basic Fitting“ von MatLab kann über den Verlauf der gemessenen ADC-Spannungen nun betrachtet werden, inwieweit diese von dem durch das Fluke Labormultimeter vorgegebenen Soll-Wert abweichen (Abb. 4.15).

Auffällig ist, dass die Werte bis zu einer Spannung von 3,2 V dicht beieinander liegen. Über diese Schwelle hinaus weichen die Ist-Werte stark von den Soll-Werten ab. Dies ist darauf zurückzuführen, dass das verwendete Referenzmultimeter ab 3,2 V in einen anderen Messwertbereich mit geringerer Auflösung schaltet. Dadurch wird die Kalibrierung ungenauer, sofern Werte über 3,2 V mit einbezogen werden. Es ergibt sich ein Residuum von etwa 4 mV, also der qualitativen Fehlerabweichung pro erfolgter Messung.

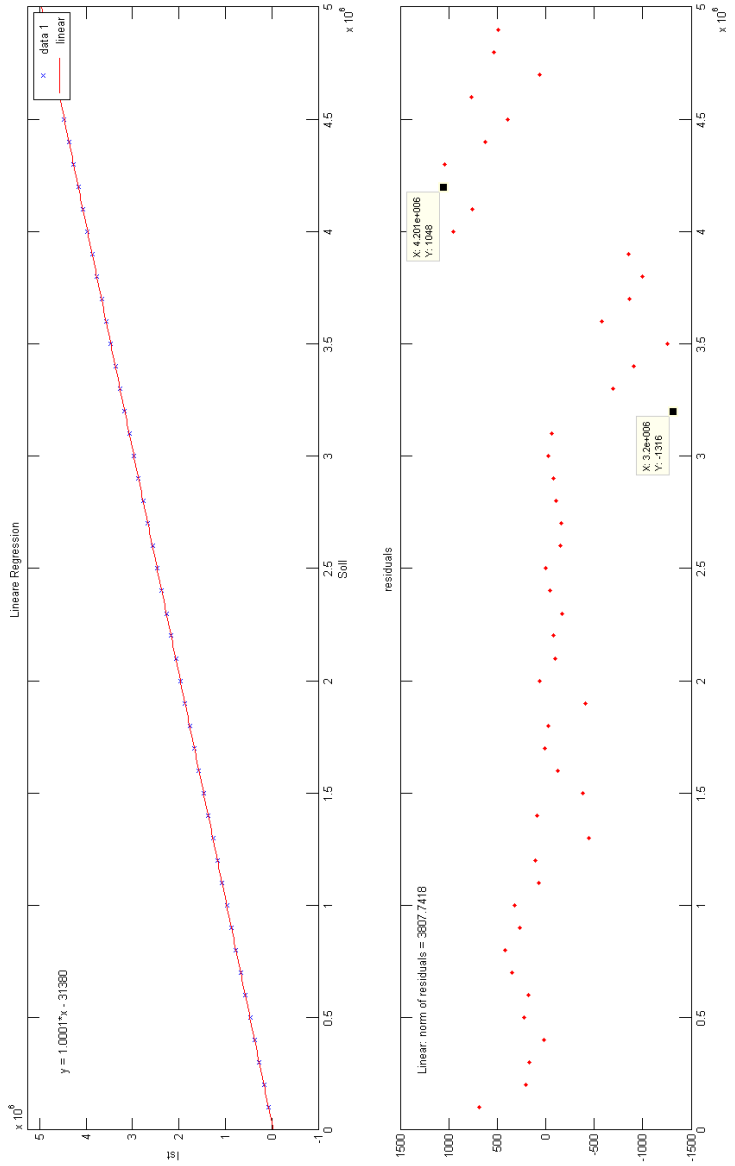


Abbildung 4.15.: Messwertabweichung linear kalibrierter ADC

Beschränkt man sich auf den Spannungsbereich von 0 bis 3,2 V, so kann eine höhere Genauigkeit erzielt werden. Diesbezüglich wurde der ADC in diesem Spannungsbereich neu linear kalibriert. Hierdurch kann die geforderte Messgenauigkeit von nahezu 1 mV erzielt werden wie es auch in Abb. 4.16 zu erkennen ist:

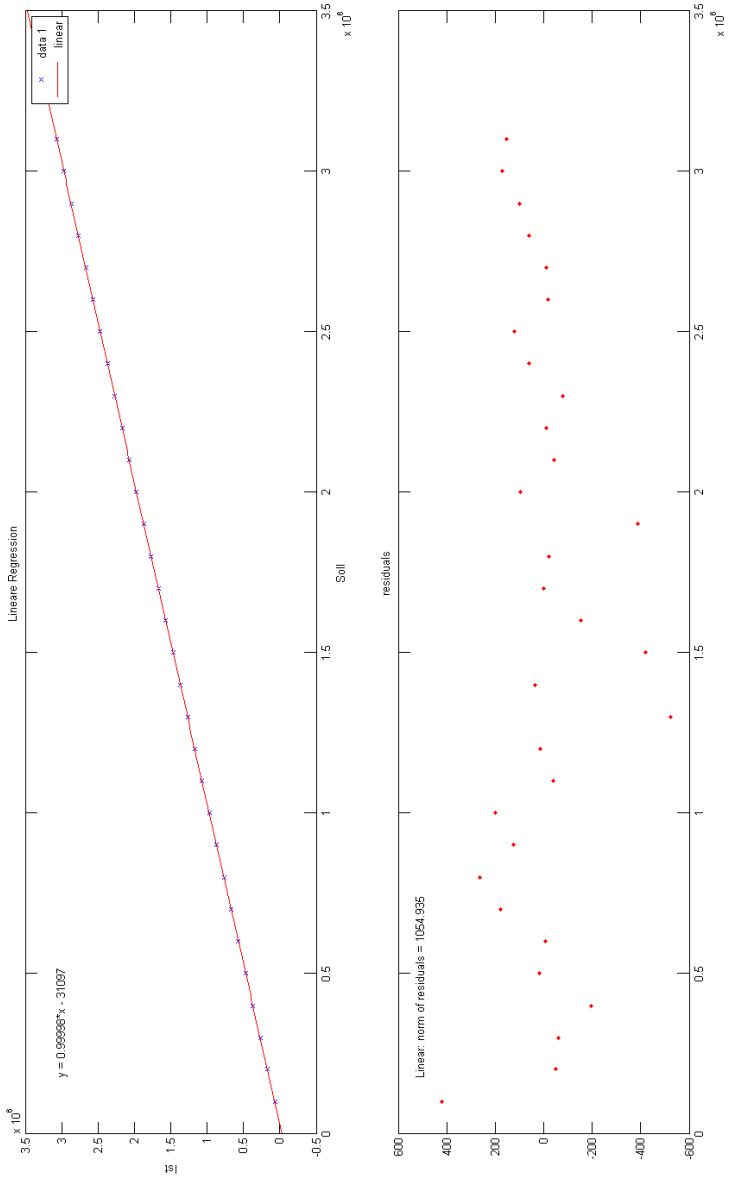


Abbildung 4.16.: Messwertabweichung linear kalibrierter ADC max 3.2 V

Eine höhere Messgenauigkeit könnte nun dadurch erzielt werden, die Kalibrierung nicht linear sondern quadratisch vorzunehmen. In der Abb. 4.17 wurde dieser Sachverhalt untersucht.

Durch Verwendung einer quadratischen Regression kann der Fehler um weitere 0.1 mV kompensiert werden. Durch diese Methode wird demnach eine der gegenüber linearen Regression zehnprozentige bessere Genauigkeit erzielt. Da die Einbindung einer quadratischen Funktion in der Controllersoftware zu einer höheren Rechenlaufzeit führt, ist diese minimal höhere Genauigkeit nicht ausreichend, um auf die Verwendung einer quadratischen Regression zurückzugreifen.

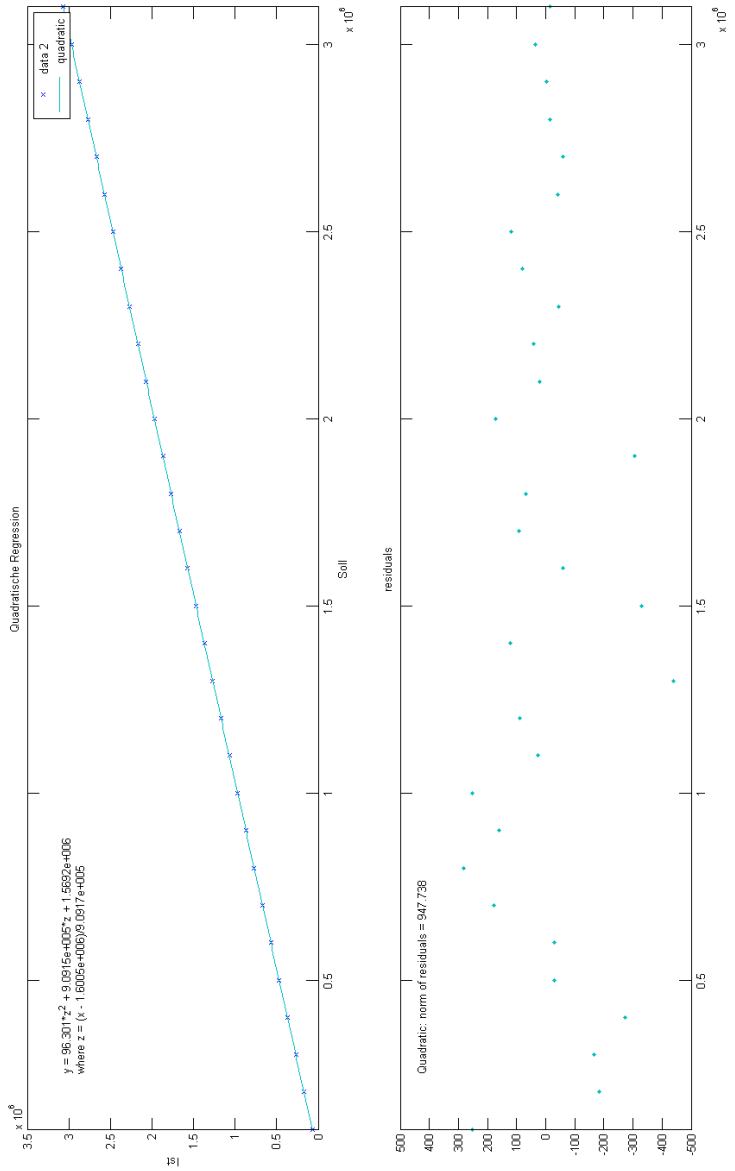


Abbildung 4.17.: Messwertabweichung quadratischer kalibrierter ADC max 3.2V

Der ADC wird der Untersuchung nach linear und kanalabhängig kalibriert. Dieser Prozess läuft halbautomatisch über die Funktion $calibrate\_adc()$ der Controllersoftware ab, mit welcher die ADC-Werte kanalweise in eine Textdatei auf der SD-Karte gespeichert werden. Mit diesen gespeicherten Werten und den abgelesenen Werten der Spannungsreferenz können über zwei Spannungwerte die Kalibrierparameter je Kanal berechnet werden (Gl. 4.2 und 4.3). Die berechneten Kalibrierwerte sind im Anhang F einzusehen.

$$Offset_{ADC} = \frac{ADC(H) * VOLTAGE(L) - ADC(L) * VOLTAGE(H)}{ADC(H) - ADC(L)} \tag{4.2}$$

$$Gain_{ADC} = \frac{ADC(L) - Offset_{ADC}}{ADC(L)} * 10^6 \tag{4.3}$$

4.6.2. HALL-Sensor

4.6.2.1. Nullpunkt-Korrektur

Während des Initialisierungsvorganges nach dem Einschalten der Zykliermaschine, erfolgt eine Nullpunktkorrektur des HALL-Sensors am 16 Bit ADC. Während dieser Einschaltphase werden keine Lastrelais geschaltet, sodass kein nennenswerter Strom fließt. Da der HALL-Sensor den Umgebungsbedingungen angepasst werden muss, erfolgt hier eine Messung des Ruhestroms, der invertiert als Offset Wert für den HALL-Sensor per Software eingestellt wird.

4.6.2.2. Automatische Messbereichsauswahl

Der verwendete HALL-Sensor verfügt, wie bereits in Kap. 2.1.3 erwähnt, über zwei Messbereiche, was eine automatische Auswahl des Messbereichs erfordert (Abb. 4.18). Hierbei wird im Blockschaltbild der Abb. 4.8 der Block “Get the current“ näher betrachtet:

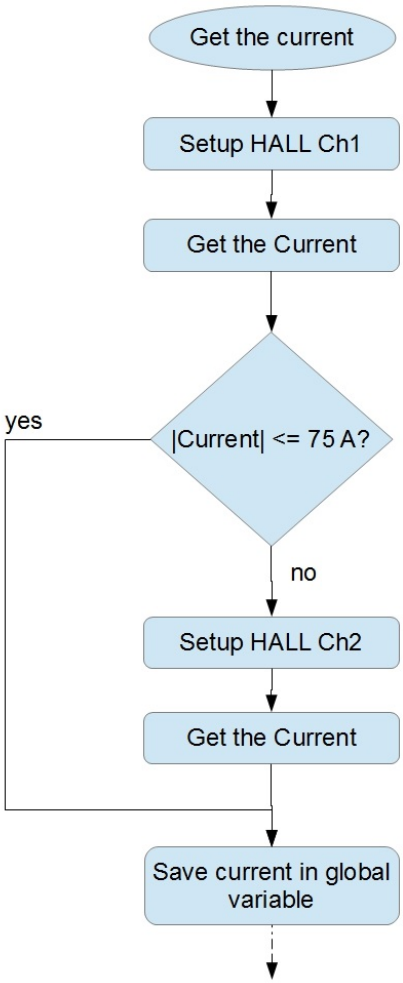


Abbildung 4.18.: Automatische Auswahl Strommessbereich

Wird im Messbereich 1 des HALL-Sensors ein Strom gemessen, der den Messbereich überschreitet, so wird automatisch auf den größeren Messbereich 2 umgeschaltet.

4.7. Real Time Clock

Die in Kap. 2.2.7 vorgestellte RTC wird vom Mikrocontroller über I2C angesteuert. Während des Initialisierungsvorganges wird einmalig nach Einschalten des Gerätes die aktuelle Uhrzeit in die Systemzeit geladen. Die Systemzeit läuft darauf unabhängig von der externen RTC weiter.

4.8. RS232 & RS485

Die Verwendung der in Kap. 2.2.4 vorgestellten Peripherie für die beiden Bustopologien RS232 und RS485 wurden innerhalb dieser Bachelorarbeit nicht gefordert. Daher, und aus Zeitgründen wurde auf eine Softwareeinbindung dieser Schnittstellen verzichtet. Die Inbetriebnahme dieser Schnittstellen wurde nicht durchgeführt und die Funktion wurde nicht erprobt.

4.9. Notlaufprogramm

Es gibt mehrere Möglichkeiten ein Notlaufprogramm zu gestalten, das nach Erkennung eines Stromausfalles ausgeführt werden kann. So könnte z.B. das Notlaufprogramm dafür sorgen, dass weiterhin Messwerte aufgenommen werden, unter der Voraussetzung, dass in der Log-File vermerkt wird, dass externe Lastgeräte durch den Stromausfall ggf. außer Betrieb gesetzt wurden. Ein anderes Konzept wäre, die Messung abrupt zu beenden und die BCMV2 in den “idle mode“ zu versetzten. Dabei sollte aktiv ein Abschalten der Lastrelais erfolgen.

Aus zeitlichen Gründen, wie auch aus technischen Gründen, wurde die Funktion eines Notlaufprogrammes innerhalb dieser Bachelorarbeit nicht implementiert. Wie in Kap. 3.3.2 erwähnt, wurde im Zykliergerät keine Backup Batterie verbaut. Aus diesen beiden Gründen muss die Implementierung eines Notlaufprogrammes nachfolgenden Bachelorarbeiten vorbehalten bleiben.

5. Erprobung des Gesamtsystems

5.1. Messreihenaufnahme an 4 Zellen

Der Zyklierprüfstand soll anhand eines Langzeitversuches an einem Batterieprüfling erprobt werden. Als Batterieprüfling wird ein Akkupack aus vier Zellen, welche in Reihe geschaltet sind, verwendet (Abb. 5.1) . Die zu verwendeten Zellen sind auf der LiFePO4-Technologie aufgebaut.

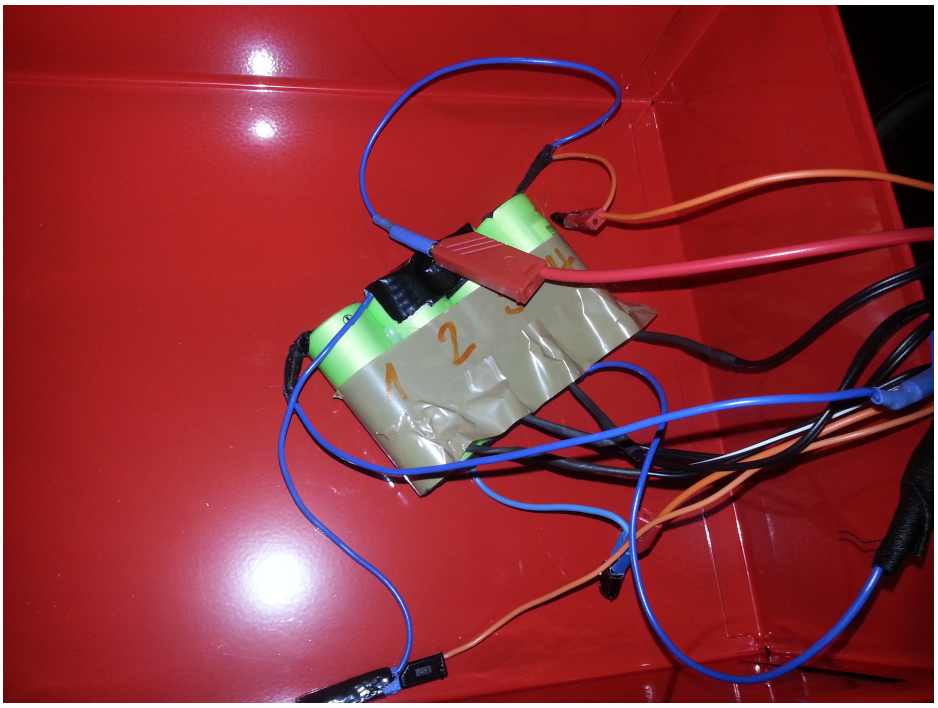


Abbildung 5.1.: Angeschlossener Batterieprüfling mit LiFePO4-Technologie

5.1.1. Erprobungsplanung

Die Zykliermaschine wird für den Versuch konfiguriert. Die Einstellung erfolgt für vier Zellen und einer Messwerterfassung im 5 Sekundentakt. Die Gesamtlaufzeit der Messung soll vier Tage in Anspruch nehmen, was 5760 Minuten entspricht. Innerhalb dieses Zeitraums werden 22 Umschaltzeitpunkte der Lastrelais definiert. Dabei werden zwei der vier Lastrelais mit einem externen Gerät verbunden (Tab. 5.1):

REL1	Ladegerät 1 A
REL2	Elektronisches Lastgerät 1 A

Tabelle 5.1.: Beschaltung der Lastrelais für Versuch mit 4 Zellen

Zusammengefasst wird die Konfigurationsdatei “config.txt“ auf der SD-Karte mit folgenden Werten aus der Tab. 5.2 versehen:

Quantity cells	04
Cyclestep	0005
Measuretime	05760
Relay Quantity	22

Tabelle 5.2.: Konfigurationsparameter für Versuch mit 4 Zellen

Um während des Versuches den Gesamtladezustand der verwendeten Batteriezellen nicht deutlich zu verändern, wird mehrmals, zu gleichen Teilen, der Batterieprüfling über jeweils 30 Minuten zunächst geladen und darauf entladen. Aufgrund der gleich eingestellten Lade- und Entladeströme soll dabei zum Ende der Messdatenaufnahme der Gesamtladezustand möglichst unverändert bleiben. Über die eingestellten Zeitparameter in der Tab. 5.3 wird der Batterieprüfling während der gesamten Laufzeit sechs mal ge- und entladen. Die entsprechen eingestellte Konfigurationsdatei befindet sich im Anhang G.1.

Umschaltzeitpunkt in min	Reiais-ID
1	1
30	2
60	0
360	1
390	2
420	0
1000	1
1030	0
1200	2
1230	0
2500	1
2530	2
2560	0
3000	1
3005	0
3060	1
3085	0
3300	2
3330	0
4600	1
4630	2
4660	0

Tabelle 5.3.: Zeitplanung Lastrelais für Versuch mit 4 Zellen

5.1.2. Darstellung der Messergebnisse

Der Langzeitversuch lief erfolgreich, und ohne dass Probleme aufgetreten bzw. bemerkt worden sind. An dieser Stelle wird mittels MatLab eine Auswertung der Log-File von der SD-Karte erstellt. Das entsprechende MatLab-Skript befindet sich im Anhang E.

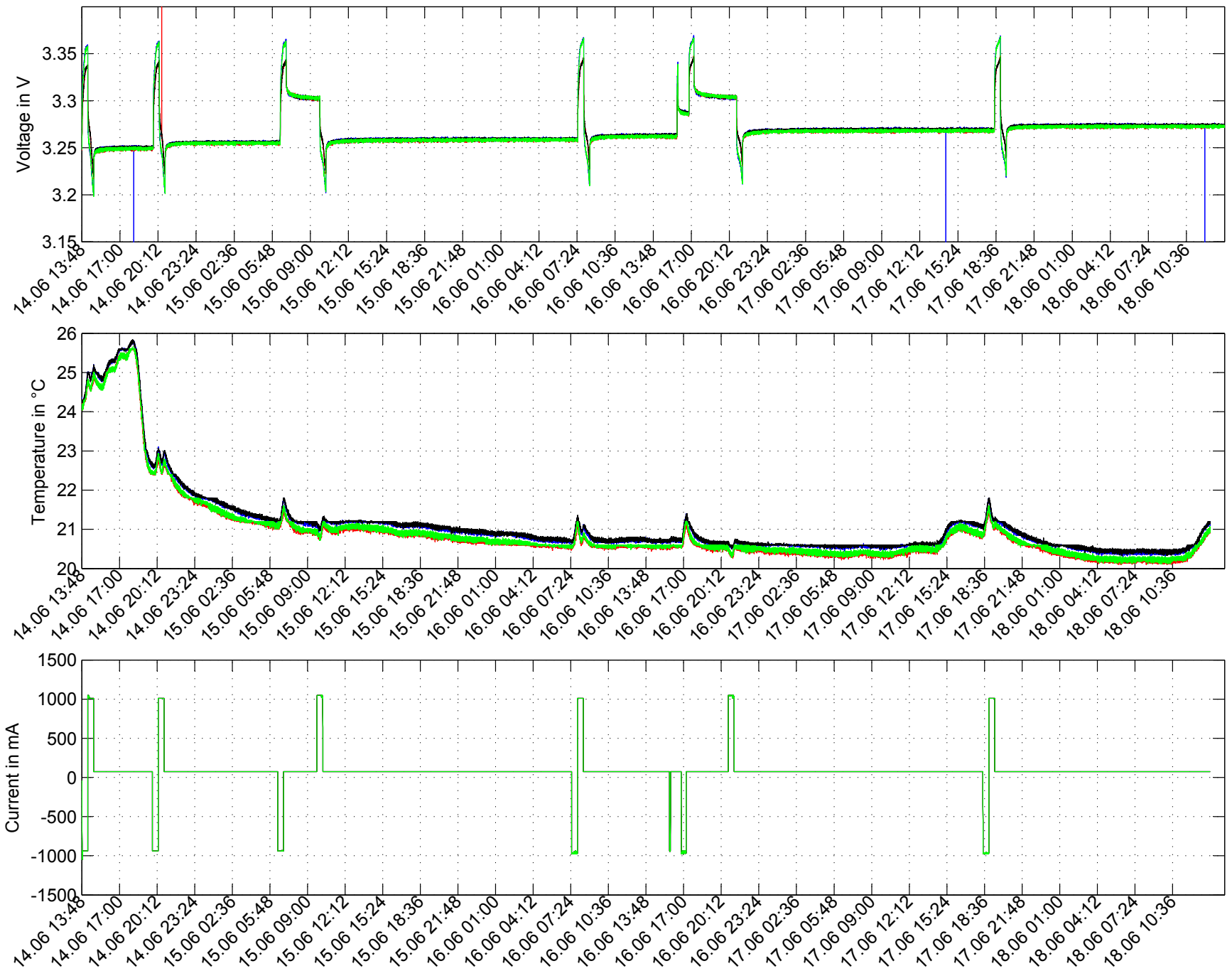
Auf der nächsten Seite befindet sich die Auswertung des Langzeitversuches (Abb. 5.2). In drei Diagrammen wird der Verlauf der einzelnen Zellspannungen und –temperaturen, sowie der Strangstrom in Abhängigkeit der Zeit dargestellt.

Die BCMV2 hat zu den eingestellten Zeitpunkten aus der Erprobungsplanung die Lastrelais geschaltet. Man erkennt deutlich in den Phasen, in denen das Ladegerät zugeschaltet wurde, einen Anstieg der Spannung jeder einzelnen Zelle. Gleiche Erkenntnisse, mit fallender Spannung, können mit dem Entladen der Zellen durch eine Last in den Phasen des entsprechend geschalteten Lastrelais erschlossen werden.

Auffällig sind die zu Anfang der Langzeitmessung vorherrschenden höheren Temperaturen. Dies liegt an der inhomogenen, wenig isolierten Einlagerung des Batterieprüflings während der Messdatenerfassung. So hat die Raumtemperatur, welche zu Anfang durch einen im selben Raum laufenden Temperaturschrankes und durch seine Abwärme manipuliert wird, einen hohen Einfluss auf die Temperatur der Zellen. Darüber hinaus können Temperaturanstiege deutlich erkannt werden, sofern ein Lade- bzw. Entladestrom fließt.

Der Ladestrom des Ladegerätes ist betragsmäßig geringfügig größer als der Entladestrom. Dadurch sind die Zellen zum Ende der Langzeitmessung mit einer höheren Ladungsmenge versehen als zum Anfang der Messung.

In Aufladephasen ist zu erkennen, dass zwei der vier Zellen eine höhere Spannungen erreichen als die anderen beiden Zellen. Ein Grund hierfür könnte sein, dass die Zellen des Batterieprüflings zuvor in zwei Strängen aufgebaut waren, die wiederum parallel betrieben wurden. Dadurch wäre die ungleichmäßige Ladungsmenge beider Stränge begründet. Für die Versuchsreihen mit dem Zyklierprüfstand wurden die parallel geschalteten Zellen in Reihenschaltung umgebaut.


Voltage in V
3.35
3.3
3.25
3.2
3.15
Temperature in °C
26
25
24
23
22
21
20
Current in mA
1500
1000
500
0
-500
-1000
-1500
14.06 13:48
14.06 17:00
14.06 20:12
14.06 23:24
15.06 02:36
15.06 05:48
15.06 09:00
15.06 12:12
15.06 15:24
15.06 18:36
15.06 21:48
16.06 01:00
16.06 04:12
16.06 07:24
16.06 10:36
16.06 13:48
16.06 17:00
16.06 20:12
16.06 23:24
17.06 02:36
17.06 05:48
17.06 09:00
17.06 12:12
17.06 15:24
17.06 18:36
17.06 21:48
18.06 01:00
18.06 04:12
18.06 07:24
18.06 10:36

5.2. Messreihenaufnahme an 4 Zellen im Temperaturschrank

In einem zweiten Versuch, durch Verwendung desselben Batterieprüflings (Kap. 5.1), soll die Zykliermaschine nochmals erprobt werden. Dabei soll der Batterieprüfling mittels eines Temperaturschrankes (Abb. 5.3) des Labors BATSEN von schwankenden Raumtemperaturen entkoppelt werden. Hierbei werden innerhalb des Versuches Messwerte in zwei Temperaturbereichen erfasst.

Abbildung 5.3.: Messaufbau Versuch mit 4 Zellen am Temperaturschrank

5.2.1. Erprobungsplanung

Für den Versuch werden wieder ein Ladegerät und ein elektronisches Lastgerät gemäß Tab. 5.4 verwendet. Die Messdatenerfassung aller vier Zellen soll in diesem Versuch jedoch mit einem Zeitabstand von 2 sec erfolgen. Die gesamte Messung soll 40 Stunden laufen, was 2400 Minuten entspricht. Um auch die Ruhespannung nach einem Lade- bzw. Entladevorgang, oder um die Einflüsse einer gezielt geregelten Temperaturänderung betrachten zu können, werden die Lastrelais mit einem Zeitabstand von sechs Stunden geschaltet. Insgesamt ergeben sich somit acht Umschaltzeitpunkte der Lastrelais gemäß Tab. 5.6.

REL1	Ladegerät 1 A
REL2	Elektronisches Lastgerät 1 A

Tabelle 5.4.: Beschaltung der Lastrelais für Versuch mit 4 Zellen und Temperaturschrank

Zusammengefasst wird die Konfigurationsdatei “config.txt“ auf der SD-Karte mit folgenden Werten aus der Tab. 5.5 versehen:

Quantity cells	04
Cyclestep	0002
Measuretime	02400
Relay Quantity	08

Tabelle 5.5.: Konfigurationsparameter für Versuch mit 4 Zellen und Temperaturschrank

Um während des Versuches den Gesamtladezustand der verwendeten Batteriezellen nicht deutlich zu ändern, wird mehrmals zu gleichen Teilen der Batterieprüfling über jeweils 15 Minuten zunächst entladen und darauf geladen. Der erste Ladevorgang soll dabei jedoch 30 Min laufen, damit der Batterieprüfling einen Ausgangsladezustand erhält. Aufgrund der gleich eingestellten Lade- und Entladeströme soll dabei zum Ende der Messdatenaufnahme der Gesamtladezustand nur leicht angehoben sein. Über die eingestellten Zeitparameter in der Tab. 5.6 wird der Batterieprüfling während der gesamten Laufzeit zweimal ge- und entladen. Die entsprechen eingestellte Konfigurationsdatei befindet sich im Anhang G.2.

Umschaltzeitpunkt in min	Temperatur	Reiais-ID
360	50	1
390	50	0
780	50	2
795	50	0
1560	20	1
1575	20	0
1980	20	2
1995	20	0

Tabelle 5.6.: Zeitplanung Lastrelais für Versuch mit 4 Zellen und Temperaturschrank

Die Temperaturregelung wird direkt am Temperaturschrank eingestellt. Der Temperaturschrank wird dabei entsprechend der Tab. 5.7 konfiguriert und zeitgleich mit dem Zyklierprüfplatz gestartet.

Umschaltzeitpunkt in min	Temperatur
0	50
1200	20

Tabelle 5.7.: Zeitplanung Temperaturregelung am Temperaturschrank

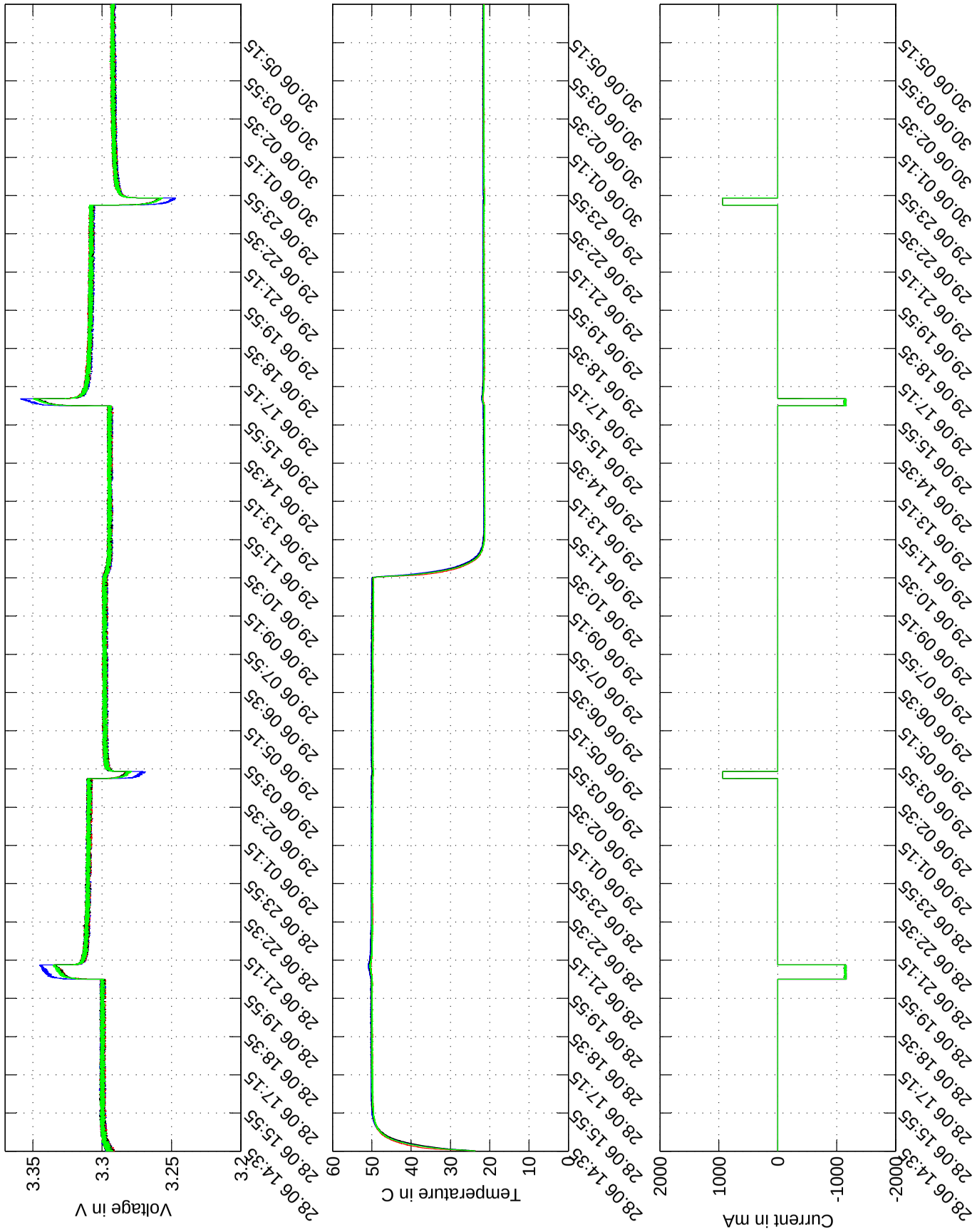
5.2.2. Darstellung der Messergebnisse

Der Langzeitversuch verlief nach Zweitanlauf erfolgreich. Im ersten Anlauf wurde versucht, die Wartezeit, die der ADC nach umschalten des Relaismultiplexers benötigt, um den Faktor 10 zu verkürzen. Durch Absenkung der Wartezeit von 10 ms auf 1 ms kam es zu einen schwerwiegenden Fehler in der Software. Die Messwerte aus dem ersten Anlauf waren somit unbrauchbar. Der Sachverhalt wird in Kap. 5.3 genauer dargestellt. Im Zweitanlauf wurde die Wartezeit wieder zurückgesetzt.

In Abb. 5.4 auf der nächsten Seite befindet sich eine grafische Auswertung der Messdaten. Anhand der Temperaturkurve ist zu erkennen, dass der Temperaturschrank gemäß seiner eingestellten Temperaturkurve gearbeitet hat. Eine hohe Temperatur sorgt für höhere Spannungen an den Zellen. Ein Anstieg der Spannungen ist zum Anfang der Langzeitmessung zu sehen. Ein Abfall der Spannungen wird bei fallender bzw. niedrigerer Temperatur beobachtet, wie z.B. zur Halbzeit des Langzeitversuches.

Wie zu erwarten, sind die Spannungsdifferenzen zwischen Laden und Entladen des Batterieprüflings temperaturabhängig. Je höher die Temperatur, desto kleiner sind die Sprünge der Spannung. Nach dem Lade- oder Entladevorgang ist die Ladungsdichte scheinbar mit höherer Temperatur effektiv größer. Der Zyklierprüfplatz eignet sich also dazu, solche temperaturabhängigen Vorkommnisse aufzuzeichnen.

Drei der vier Zellen sind während der Lastphasen auf gleichem Spannungsniveau. Die Zelle 1 hat höheres Spannungspotential. Grund hierfür könnte sein, dass die Ladungsdichten der einzelnen Zellen unterschiedlich sind. Gegen der Erwartung, dass die Ladungsmenge zum Ende der Messung nahezu gleich sein soll wie zum Anfang des Langzeitversuches, ist die Ladungsmengen kleiner. Eine Vermutung ist es, dass der Batterieprüfling durch Belastungen mittlerweile leicht geschädigt ist.


Voltage in V
3.35
3.3
3.25
3.2
Temperature in C
60
50
40
30
20
10
Current in mA
2000
1000
0
-1000
-2000
28.06 14:35
28.06 15:55
28.06 17:15
28.06 18:35
28.06 19:55
28.06 21:15
28.06 22:35
28.06 23:55
29.06 01:15
29.06 02:35
29.06 03:55
29.06 05:15
29.06 06:35
29.06 07:55
29.06 09:15
29.06 10:35
29.06 11:55
29.06 13:15
29.06 14:35
29.06 15:55
29.06 17:15
29.06 18:35
29.06 19:55
29.06 21:15
29.06 22:35
29.06 23:55
30.06 01:15
30.06 02:35
30.06 03:55
30.06 05:15

5.3. Probleme, Fehlerbehebungen und Optimierung

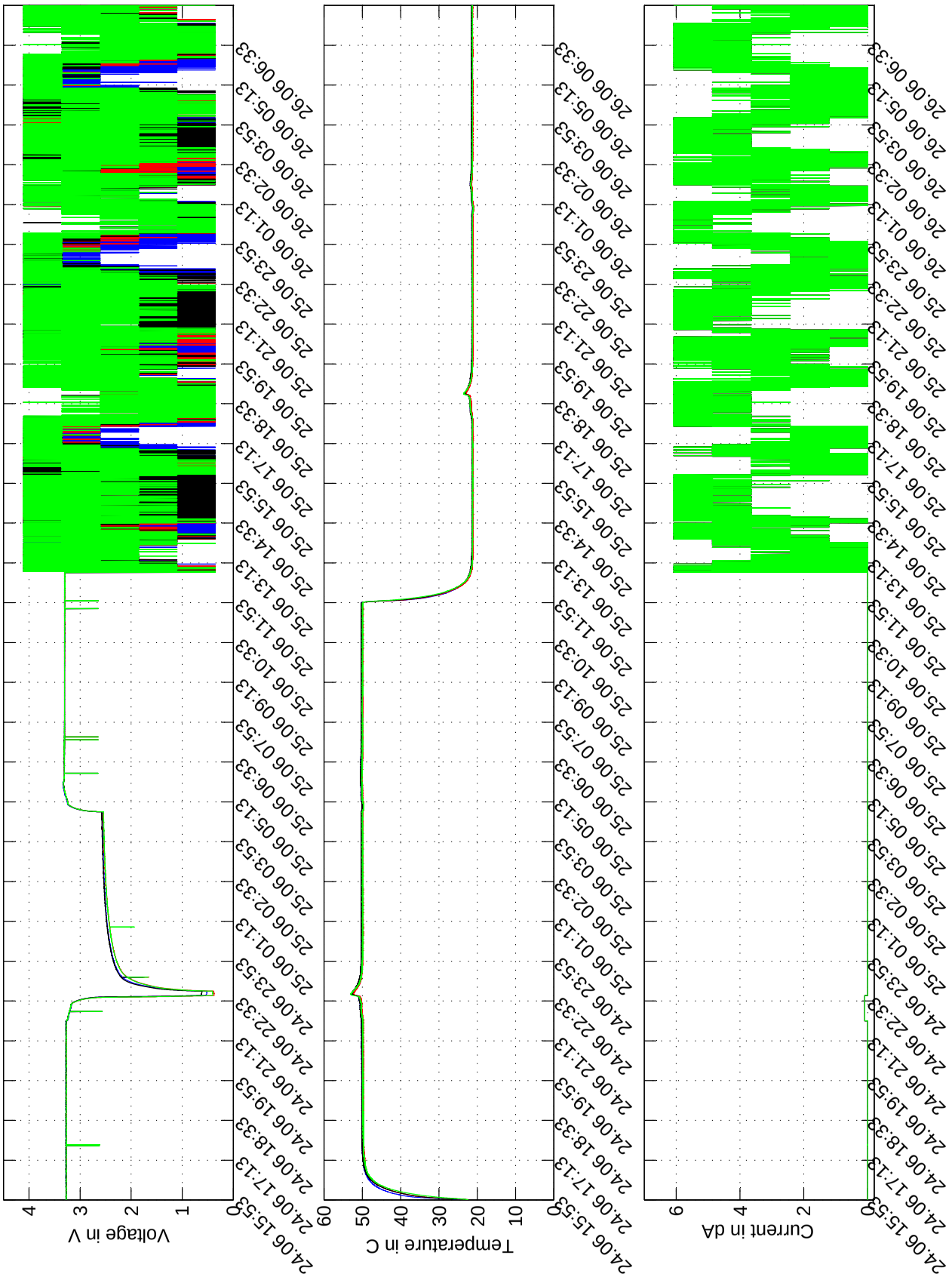
Messausreißer

In den Messdaten des ersten Langzeitversuches (Kap. 5.1) sind einzelne Messausreißer in der Spannungsmessung aufgetreten. Die Fehlerrate von 0,005 % ist jedoch mit vier fehlerhaften Messwerten bei 69119 erfassten Spannungsmesswerten sehr gering. Beim zweiten erfolgreichen Versuch (Kap. 5.2) sind ohne Optimierungsmaßnahmen keine Messfehler aufgetreten.

Fehler Versuch 2 Erstanlauf

Im zweiten Versuch wurde beim Erstanlauf die Wartezeit verkürzt, die der ADC nach Schalten des Relaismultiplexers abwartet. Dabei wurde die Zeit um den Faktor 10 von 10 ms auf 1 ms verkleinert. Die Messdatenerfassung verlief zu Anfang gut, nach ungefähr der halben Zeit wurden jedoch unrealistische Messwerte aufgezeichnet. Der Verlauf der fehlerhaften Spannungen und Ströme kann in Abb. 5.5 auf der nächsten Seite betrachtet werden.

Auffällig in der Messdatenauswertung ist, dass die Messwerte in bestimmten Quantisierungsstufen springen. Eine Vermutung ist, dass die softwaretechnisch geänderte Wartezeit zu diesen Fehler führen könnte. Möglicherweise reicht die Wartezeit nicht aus, damit sich die zu messende Spannung am ADC einschwingt. Auch kann es sein, dass die REED-Relais des Relaismultiplexers innerhalb der vorgegeben Zeit nicht richtig schalten. Unbegründet hierbei ist jedoch, dass der Fehler ab einem bestimmten Zeitpunkt fortlaufend stattfindet, zuvor jedoch die Messwerterfassung ohne Probleme erfolgt. Unter Umständen kann auch eine Fehlerübertragung der SPI-Schnittstelle zu gegebenem Zeitpunkt dazu geführt haben, den ADC mit falschen Werten zu konfigurieren. Die Fehlerursache ist nicht gänzlich bekannt, jedoch konnte dieser Fehler bei einem zweiten Anlauf des gleichen Versuches nicht wieder hervorgerufen werden. Im zweiten Anlauf wurde die Wartezeit wieder um den Faktor 10 auf 10 ms erhöht. Auf eine weiterführende Analyse des Fehlverhaltens unter Verkürzung der Wartezeit wurde verzichtet.


Voltage in V
Temperature in C
Current in dA
24.06 15:53
24.06 17:13
24.06 18:33
24.06 19:53
24.06 21:13
24.06 22:33
24.06 23:53
25.06 01:13
25.06 02:33
25.06 03:53
25.06 05:13
25.06 06:33
25.06 07:53
25.06 09:13
25.06 10:33
25.06 11:53
25.06 13:13
25.06 14:33
25.06 15:53
25.06 17:13
25.06 18:33
25.06 19:53
25.06 21:13
25.06 22:33
25.06 23:53
26.06 01:13
26.06 02:33
26.06 03:53
26.06 05:13
26.06 06:33

5.4. Kurzanleitung

Für einen schnellen Einblick der Funktionen des Zyklierprüfplatzes wurde eine umfassende Kurzanleitung geschrieben. Damit kann der Benutzer, ohne tiefere Kenntnisse aus dieser Bachelorarbeit zu besitzen, den Prüfplatz gemäß der Anleitung in Betrieb nehmen. Diese Kurzanleitung befindet sich im Regelfall in einer der beiden Schubladen des Prüfplatzes. Die Kurzanleitung kann auch im Anhang H eingesehen werden.

6. Gesamtbewertung und Fazit

In diesem Kapitel soll die erarbeitete Lösung in Bezug auf die Aufgabenstellung aus Anhang A diskutiert werden. Dabei sollen verwirklichte Ziele noch einmal erfasst werden und ausstehende Lösungen für Zielstellungen aufgezählt werden. Weiterhin soll in einem Ausblick dargestellt werden, welche möglichen zusätzlichen Ziele für Nachfolgearbeiten angesetzt und welche erbrachten Lösungen optimiert werden können.

6.1. Zusammenfassung

Die Hauptaufgabe des Zyklierprüfplatzes wurde realisiert. Es wurde ein Gerät entwickelt, welches zyklisch in bestimmten Zeitabständen Messwerte aufnimmt. Dabei wurde ermöglicht, Messdaten von bis zu zwölf in Reihe geschalteten Batteriezellen aufzuzeichnen. Über ein Relaismultiplexer wurde ein Messzugang jeder einzelnen Zelle für einen 16 - Bit ADC ermöglicht. Dadurch können die Spannungsmesswerte galvanisch getrennt und seriell erfasst werden. Durch Einsetzten eines HALL-Sensor im Lastkreis kann ebenfalls der Strom gemessen werden. Über NTC-Widerstände können einzelne Zelltemperaturen gemessen werden. Die ganzen Vorgänge werden von einem Mikrocontroller der Familie ARM Cortex M3 gesteuert. Dabei werden die vom eingebetteten System erzeugten Messdaten auf eine entnehmbare SD-Karte gespeichert. Zusätzlich hat der Nutzer Möglichkeiten über mehrere Schnittstellen den Zyklierprüfplatz zu konfigurieren. So kann über eine Konfigurationsdatei, welche auf der SD-Karte abgelegt wird, eine Konfiguration für den Zyklierbetrieb eingelesen werden. Darüber hinaus hat der Nutzer Möglichkeiten, einzelne Konfiguration über eine Menüführung durchzuführen, indem er sich über UART und einem Terminalprogramm anbindet. Ebenso kann der Zugriff als Fernzugriff über ein LAN-Netzwerk und MatLab erfolgen. Über ein Informationsdisplay am Frontpanel kann der Benutzer im laufenden Betrieb aktuelle Messwerte und den Zyklenablauf über vorhandene Taster kontrollieren.

Die Software ist in der Lage, auf der Lastplatine sich befindenden Mosfets über ein Zeitprotokoll zu steuern. So können über Lastrelais, welche an die Mosfets angebunden sind, externe angeschlossene Geräte dem Batterieprüfling zugeschaltet werden. Die Wahl der externen Geräte ist dabei frei und kann vom Nutzer bestimmt werden.

Die für den Zyklierprüfplatz nötigen Module und Lastelemente, wie auch die Stromversorgung, wurden in einem 19 Zoll Gehäuse untergebracht. Der Nutzer hat dadurch die vereinfachte Möglichkeit, über ein angebrachtes Frontpanel den Zyklierprüfplatz mit dem Batterieprüfling zu verbinden. Das Gehäuse wurde darüber hinaus in einem Metallschrank verbaut und bietet mit montierten Schubladen Stauraum für benötigte Peripherie, wie z.B. den Temperatursensoren und die für den Prüfplatz aufbereitete Kurzanleitung.

Die interne Forderung, eine Genauigkeit der Spannungsmessung auf 1 mV zu erzielen, wurde erfüllt. Ebenso wurden optionale Ziele erfüllt, wie z.B. die softwaretechnische Anbindung der Real Time Clock. Die RTC-Zeit wird beim Einschalten des Gerätes in die Systemzeit geladen, wodurch der Zyklierprüfplatz stetig mit aktueller Uhrzeit die Messdaten aufnehmen kann.

Für RS232 und RS485 wurden Anschlüsse und die nötigen Bauelemente in der Hardware konzeptioniert und umgesetzt. Da bisher keine Verwendung für diese Anschlüsse vorlag, wurde keine Software für die beiden Bustopologien implementiert.

Die Messdaten können nach einer erfolgreichen Messdatenaufnahme von der SD-Karte ausgelesen und ausgewertet werden. Das Auslesen der SD-Karte ist direkt mit einem PC und MatLab möglich und erfolgreich getestet. Auch können die Messdaten über UART und einem Terminalprogramm abgerufen werden. Die Erfassung der Messdaten über Ethernet war nicht erfolgreich. Zwar ist die generelle Kommunikation beider Teilnehmer über ein LAN-Netzwerk vorhanden und stabil, das versenden größerer Datenmengen vom Zyklierprüfplatz zum PC stellte sich jedoch als fehlerhaft heraus. Der Sachverhalt konnte aus Zeitgründen nicht diagnostiziert werden.

6.2. Erprobungserfahrung und Bewertung

Die in Kap. 5 durchgeführten Probeläufe haben bestätigt, dass der Zyklierprüfstand seine Aufgabe erfüllt. Verschiedene Versuche haben dabei gezeigt, dass der Zustand des Batterieprüflings unter eingestellter Konfiguration zeitlich fortlaufend aufgezeichnet wird. Die Zeitprotokolle werden eingehalten, sodass die Lastrelais des Lastkreises entsprechend der Konfiguration schalten. Unter Berücksichtigung der Messdatenanzahl konnte eine Fehlermessrate von ca. 0,005 % diagnostiziert werden. Die Erprobungsversuche wurden an einem Batterieprüfling mit vier Zellen durchgeführt. Versuche an Bleibatterien mit sechs Zellen wurden nicht durchgeführt.

Versuche, die Wartezeit zwischen der Messdatenerfassung zweier Zellen zu verkürzen, haben zu massiven Messfehlern geführt. Trotz dieser Messfehler gilt jedoch, dass Optimierungen der Wartezeit für einen schnelleren Zyklenablauf erfolgen könnten.

6.3. Fazit mit Ausblick

Der Zyklierprüfplatz hat mit Vollendung dieser Bachelorarbeit einen funktionierenden Zustand erreicht. Messungen können mitsamt der Zyklenablaufsteuerung durchgeführt werden. Mit der aufgebauten Hardware können weitere mögliche denkbare Funktionen eingebunden werden, welche bislang softwareseitig nicht implementiert wurden. Bei einer Änderung der Hardware muss berücksichtigt werden, dass die Produktion der in dieser Bachelorarbeit verwendeten Mikrocontroller-Familie ARM Cortex M3 mittlerweile komplett eingestellt wurde.

6.3.1. Mikrocontroller

Der direkte Nachfolger, die Familie ARM Cortex M4 [33], wurde von TI bereits vorgestellt. Eine Möglichkeit wäre auf diese Controller-Familie für den Zyklierprüfplatz umzusteigen. Die spezifischen Mikrocontroller sind zum Zeitpunkt der Erstellung dieser Bachelorarbeit noch nicht verfügbar. Eine Übersicht der Spezifikationen der Familie ARM Cortex M4 bietet die Abb. 6.1.

Aufgrund der geänderten Spezifikationen, ist eine mögliche Pin-Kompatibilität zum ARM Cortex M3 nahezu auszuschließen. Es gilt auch zu untersuchen, ob die Familie Cortex M4

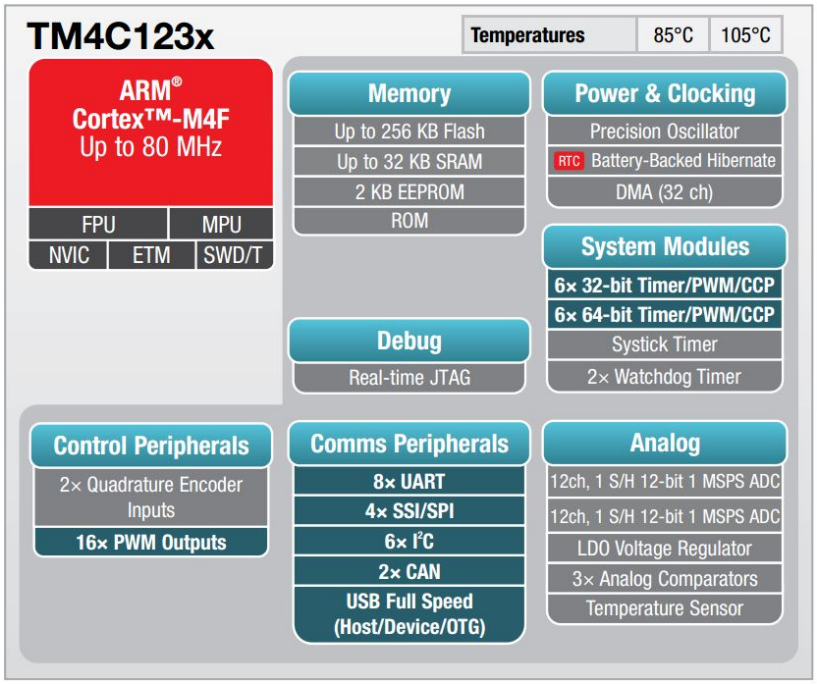


Abbildung 6.1.: Spezifikation ARM Cortex M4 (Quelle: [33])

die Stellarisware unterstützen wird. Aktuell ist hierzu keine Information verfügbar, jedoch wird eine Tivaware erwähnt, zu der keine Informationen vorliegen. Bei der Tivaware könnte es sich um ein Softwarebibliothek ähnlich der Stellarisware handeln. Diese Annahme ist jedoch rein spekulativ. Ebenso ist es fraglich, ob der Controller ARM Cortex M4 Ethernet unterstützen wird.

6.3.2. Konfigurationsdatei und Kontrolle der Parameter

Das Einlesen der Konfigurationsdatei innerhalb der Software verläuft zwar problemlos, ist aber an viele Bedingungen geknüpft. Die Formatierung der Konfigurationsdatei muss strikt eingehalten werden, bietet keine Erweiterbarkeit und kann auch nicht benutzerfreundlich kommentiert werden. Der Bedienkomfort könnte durch eine bessere Einlese-Software der Konfigurationsdatei optimiert werden, sodass die strikte Formatierung der Konfigurationsdatei aufgelöst werden könnte.

Ebenso könnten Funktionen implementiert werden, welche Benutzereingaben über UART bzw. Ethernet überprüfen. Dadurch könnte unter anderem eine Falscheingabe für Parameter

der Zyklenablaufsteuerung unterdrückt werden.

6.3.3. RS232 und RS232/RS485

Die Busschnittstellen RS232 und RS232/485 wurden in der Controllersoftware bisher nicht berücksichtigt und von dem Zyklierprüfplatz nicht benötigt. Diese Schnittstellen bieten die Möglichkeit mittels externer Geräte den Zyklierprüfplatz um bestimmte Funktionen zu erweitern. Dadurch wäre z.B. eine Steuerung des im Labor BATSEN vorhandenen Temperaturschrankes über RS232 möglich. Weiterhin könnte mit einem Fluke Labormessgerät, welches ebenfalls über eine RS232-Schnittstelle verfügt, eine automatische Kalibrierung des 16 Bit ADC erfolgen.

6.3.4. Datenübertragung über Ethernet

Wie in Kap. 4.5.2.3 erwähnt, traten während der Realisierung dieser Bachelorarbeit Probleme mit der Übertragung größerer Dateimengen über die Ethernet-Schnittstelle auf. Um einen Fernzugriff über ein LAN-Netzwerk auf die Messdaten zu ermöglichen, gilt es diesen Sachverhalt anhand der lwIP-Libraries in der Controllersoftware zu untersuchen, zu diagnostizieren und diese Problematik anhand der Erkenntnisse zu beheben.

6.3.5. Sicherheitsrelevante Funktionen

Die Hardware des Zyklierprüfplatzes ist für ein Notlaufprogramm bei Stromausfall des Stromnetzes vorbereitet. In der Software können Funktionen erschaffen werden, die bestimmen, wie ein Notlaufprogramm strukturiert ablaufen könnte.

Weiterhin könnten sicherheitsrelevante Funktionen in der Software implementiert werden, die den aktuellen Gesundheitszustand des Batterieprüflings permanent kontrollieren und gegebenenfalls den Batterieprüfling z.B. vor Tiefentladung, Überstrom oder Temperaturüberschreitung schützen, in dem aktiv in den Lastkreis durch Lastrelaisabschaltung eingegriffen wird.

Über den Einsatz einer hardware-gestützten “SafetyUnit“ könnte eine weitere Sicherheitsinstanz eingebunden werden, die unabhängig von der Controllersoftware eingreifen könnte.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] ADAM DUNKELS (Hrsg.): *Lightweight TCP/IP stack*. A. DUNKELS: ADAM DUNKELS, Version: 2012. http://savannah.nongnu.org/projects/lwip/. – Abruf: 02.07.2013

[2] ANALOG DEVICES (Hrsg.): *ADG726*. AD: ANALOG DEVICES, Version: 2002. http://www.analog.com/static/imported-files/data_sheets/ADG726_732.pdf. – Abruf: 08.06.2013

[3] ANALOG DEVICES (Hrsg.): *AD7798/7799*. AD: ANALOG DEVICES, Version: 2007. http://www.analog.com/static/imported-files/data_sheets/AD7798_7799.pdf. – Abruf: 02.06.2013

[4] 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: 02.06.2013

[5] COTO TECHNOLOGY (Hrsg.): *COTO 9007*. COTO: Coto Technology, Version: 2013. http://www.cotorelay.com/datasheets/Coto%20Technology%209007%20Spartan%20SIP%20Reed%20Relay.pdf. – Abruf: 02.06.2013

[6] ELECTRONIC ASSEMBLY (Hrsg.): *EA W204B-NLW*. EAS: ELECTRONIC ASSEMBLY, Version: 2008. http://www.lcd-module.de/fileadmin/pdf/doma/blueline-w.pdf. – Abruf: 09.06.2013

[7] EPCOS (Hrsg.): *B57703M*. EPCOS: EPCOS, Version: 2009. http://www.epcos.com/inf/50/db/ntc_09/ProbeAss__B57703__M703.pdf. – Abruf: 02.06.2013

[8] FLUKE (Hrsg.): *FLUKE 45 Handbuch*. FLUKE: FLUKE, Version: 1996. http://assets.fluke.com/manuals/45_____umger0400.pdf. – Abruf: 02.07.2013

[9] FUTURE TECHNOLOGY DEVICES INTERNATIONAL LTD (Hrsg.): *FT2232D*. FTDI Chip: Future Technology Devices International Ltd, Version: 2010. http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf. – Abruf: 08.06.2013

[10] FUTURE TECHNOLOGY DEVICES INTERNATIONAL LTD (Hrsg.): *User Guide For FTDI FT Prog Utility*. FTDI Chip: Future Technology Devices International Ltd, Version: 2011. http://www.ftdichip.com/Support/Documents/AppNotes/AN_124_User_Guide_For_FT_PROG.pdf. – Abruf: 18.06.2013

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

[12] HITACHI SEMICONDUCTOR & INTEGRATED CIRCUITS (Hrsg.): *HD44780*. HITACHI: HITACHI Semiconductor & Integrated Circuits, Version: 1998. https://www.sparkfun.com/datasheets/LCD/HD44780.pdf. – Abruf: 09.06.2013

[13] LEM (Hrsg.): *DHAB S/24*. LEM: LEM, Version: 2012. http://www.lem.com/docs/products/dhab%20s24.pdf. – Abruf: 02.06.2013

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

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

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

[17] STMICROELECTRONICS (Hrsg.): *M93C86xx*. ST: STMicroelectronics, Version: 2013. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00001142.pdf. – Abruf: 08.06.2013

[18] TEKTRONIX (Hrsg.): *Mixed Signal Oscilloscopes*. TEKTRONIX: TEKTRONIX, Version: 2012. http://www.tek.com/sites/tek.com/files/media/media/resources/MSO-DPO3000_Series_Oscilloscopes_Datasheet_3GW-21364-9_1.pdf. – Abruf: 02.07.2013

[19] TEXAS INSTRUMENTS (Hrsg.): *CD74HC4514*. TI: TEXAS INSTRUMENTS, Version: 2003. http://www.ti.com/lit/ds/symlink/cd74hc4514.pdf. – Abruf: 09.06.2013

[20] TEXAS INSTRUMENTS (Hrsg.): *SN74AHCT14*. TI: TEXAS INSTRUMENTS, Version: 2003. http://www.ti.com/lit/ds/symlink/sn74ahct14.pdf. – Abruf: 09.06.2013

[21] TEXAS INSTRUMENTS (Hrsg.): *SN75176b*. TI: TEXAS INSTRUMENTS, Version: 2003. http://www.ti.com/lit/ds/symlink/sn65176b.pdf. – Abruf: 08.06.2013

[22] TEXAS INSTRUMENTS (Hrsg.): *LMS1587*. TI: TEXAS INSTRUMENTS, Version: 2005. http://www.ti.com/lit/ds/symlink/lms1585a.pdf. – Abruf: 18.06.2013

[23] TEXAS INSTRUMENTS (Hrsg.): *MAX3232E*. TI: TEXAS INSTRUMENTS, Version: 2007. http://www.ti.com/lit/ds/symlink/max3232e.pdf. – Abruf: 08.06.2013

[24] TEXAS INSTRUMENTS (Hrsg.): *BQ32000*. TI: TEXAS INSTRUMENTS, Version: 2010. http://www.ti.com/lit/ds/symlink/bq32000.pdf. – Abruf: 09.06.2013

[25] TEXAS INSTRUMENTS (Hrsg.): *bq77910EVM User's Guide*. TI: TEXAS INSTRUMENTS, Version: 2011. http://www.ti.com/lit/ug/sluu368/sluu368.pdf. – Abruf: 25.05.2013

[26] TEXAS INSTRUMENTS (Hrsg.): *LM3S9B92 ROM User's Guide*. TI: TEXAS INSTRUMENTS, Version: 2011. http://www.ti.com/lit/ug/spmu125c/spmu125c.pdf. – Abruf: 02.07.2013

[27] TEXAS INSTRUMENTS (Hrsg.): *Stellaris® LM3S9B92 Evaluation Kit User's Manual*. TI: TEXAS INSTRUMENTS, Version: 2011. http://www.ti.com/lit/ug/spmu035c/spmu035c.pdf. – Abruf: 25.05.2013

[28] TEXAS INSTRUMENTS (Hrsg.): *Stellaris® LM3S9B92 Microcontroller DATA SHEET*. TI: TEXAS INSTRUMENTS, Version: 2012. http://www.ti.com/lit/ds/symlink/lm3s9b92.pdf. – Abruf: 25.05.2013

[29] 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

[30] TEXAS INSTRUMENTS (Hrsg.): *TPS73801*. TI: TEXAS INSTRUMENTS, Version: 2012. http://www.ti.com/lit/ds/symlink/tps73801.pdf. – Abruf: 02.06.2013

[31] TEXAS INSTRUMENTS (Hrsg.): *CSD18504Q5A*. TI: TEXAS INSTRUMENTS, Version: 2013. http://www.ti.com/lit/ds/symlink/csd18504q5a.pdf. – Abruf: 09.06.2013

[32] TEXAS INSTRUMENTS (Hrsg.): *StellarisWare®*. TI: TEXAS INSTRUMENTS, Version: 2013. http://www.ti.com/tool/sw-lm3s. – Abruf: 18.06.2013

[33] TEXAS INSTRUMENTS (Hrsg.): *Tiva C Series ARM Microcontrollers*. TI: TEXAS INSTRUMENTS, Version: 2013. http://www.ti.com/lit/sg/spmt285/spmt285.pdf. – Abruf: 27.06.2013

[34] WIKIPEDIA (Hrsg.): *Lineare Regression*. Wiki: WIKIPEDIA, Version: 2013. http://de.wikipedia.org/wiki/Lineare_Regression. – Abruf: 02.07.2013

Abkürzungsverzeichnis

ADC Analog Digital Converter

BATSEN Drahtlose Zellensensoren für Fahrzeugbatterien

BCMV1 Battery Cycling Machine V1.0

BCMV2 Battery Cycling Machine V2.0

CTS Clear to Send

CS Chip select

CCSv5 Code Composer Studio v5

Δ-Σ-ADC Delta-Sigma-ADC

EEPROM Electrically Erasable PROgrammable Memory

FAT File Allocation Table

FIFO First In - First Out

GPIO General Purpose Input/Output

HAW Hamburg Hochschule für Angewandte Wissenschaften Hamburg

I2C Inter-Integrated Circuit

ICDI-Board In-Circuit Debug Interface Board

IEEE Institute of Electrical and Electronics Engineers

JTAG Joint Test Action Group - Programmier-/Debug-Schnittstelle

LDO Low-dropout

LDMV1 Load Driver Module V1

LDMV2 Load Driver Module V2

LI-ION Lithium-Ionen

LiFePO4 Lithium-Eisenphosphat

lwIP Lightweight TCP/IP stack

NTC Negative Temperature Coefficient Thermistor

PLL Phase-Locked Loop

RTC Real Time Clock

RS232 Recommended Standard 232

RS485 Recommended Standard 485

RTS Ready to Send

SFN Short File Name

SOC State of Charge

SOH State of Health

SPI Serial Peripheral Interface

TI Texas Instruments

TTL Transistor-Transistor-Logik

UART Universal Asynchronus Receiver Transmitter

USB Universal Serial Bus

Anhang

A. Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

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

2. Mai 2013

Bachelorarbeit: Thomas Wisniewski

Zyklierprüfstand für Batteriezellen mit Steuerung durch einen ARM-Controller sowie Messdatenverwaltung und Netzwerkanbindung

Motivation

Im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten Forschungsvorhabens BATSEN (drahtlose Zellensensoren für Fahrzeugbatterien) sind Versuche mit Fahrzeugbatterien und Einzelzellen vorgesehen. Wesentlich dafür ist die Erfassung von zahlreichen Messwerten und einstellbaren sowie reproduzierbaren Betriebsbedingungen.
Die Bachelorarbeit soll einen Prüfstand für Batteriezellen und Batteriesysteme entwickeln. Insbesondere soll ein Mess- und Überwachungssystem für mehrere Zellenspannungen, Zelltemperaturen realisiert werden. Lade- bzw. Entladeströme sollen über Lastrelais gesteuert werden können. Durch eine Umschaltmatrix ist ein messtechnischer Zugang zu den Zellen zu realisieren.
Ein leistungsfähiger Mikrocontroller aus der ARM-Familie soll die Steuerung übernehmen und dabei zahlreiche Schnittstellen bedienen. Über eine Netzwerkanbindung soll eine günstige Anbindung an die Datenverwaltung auf einem PC erfolgen.

Aufgabe

Die Aufgabe der Bachelorarbeit gliedert sich wie folgt:

1) **Erfassung der Vorarbeiten und Analyse der Rahmenbedingungenn**

   - Recherche in Fach- und Firmenliteratur
   - Darstellung der bereits im Projekt erarbeiteten Rahmenbedingungen
   - Abschätzung von Anforderungen, Erweiterungen und Änderungen sowie des Realisierungsaufwandes
   - Auswahl passender Bauelemente und bewährter Schaltungsteile (Stromversorgung, externer 16Bit-ADC, externe NTC-Temperatursensoren, REED-Relais)

2) **Voruntersuchungen und Konzeption**

   - Anpassung und Erarbeitung des neuen Schaltungskonzepts
   - Entwurf einer Hauptplatine mit Mikrocontroller
   - Entwurf einer Relaisumschaltlogik mit zusammengefasster Lösung für Multiplexer und Treiber
   - Konzept mehrerer umschaltbarer Busanbindungsmöglichkeiten und Interfaces (RS232, RS485, Ethernet und SD-Karte mit SPI)
   - Einbindung und Schnittstelle für ein externes Lasttreibermodul und Steuerlogik
   - Anbindung eines LED-Displays

- Planungsanpassung für das Gehäuse
- Konzept Notlaufprogramm bei Stromausfall in Hard- und Software
- Vorbereitung externe Real-Time-Clock mit eigener Backup-Supply in HW

3) **Schaltungsentwicklung, Aufbau und Inbetriebnahme**

- Schaltungs- und Platinenentwurf
- Platinenbestückung für zwei Musteraufbauten
- Sicherungen gegen Überstrom, Verpolung und manuelle Abschaltfunktionen
- Mechanischer Aufbau eines gesamten Systems inkl. Gehäuse und Peripherie
- Inbetriebnahme der Platinen im Labor

4) **Konzeption, Entwurf und Implementation der Controllersoftware**

- Entwicklung der Software für den Controller
- Ablaufsteuerung des Zyklierbetriebs durch Dialogeinstellung und mit Hilfe einer Konfigurationdatei
- Steuerung der angebundenen Peripherie, wie externer 16Bit-ADC, externe NTC-Temperatursensoren, REED-Relais, RS232, RS485, Ethernet und SD-Karte mit SPI, Display
- Kommunikation über Ethernet über TCP-IP-Libraries sowie MATLAB-Skripten auf dem PC
- Funktion zur Kalibrierung, wie kanalabhängige Kalibrierung des externen ADCs und Nullpunkt-Korrektur für den HALL-Sensor sowie automatische Auswahl Strommessbereich
- Optional: Unterstützung der Real-Time-Clock in Software

5) **Erprobung des Gesamtsystems**

- Erprobungsplanung und Funktionsnachweis durch Messreihenaufnahme an realen Batterien (4 und 6 Zellen)
- Erfassung von Problemen, ggf. Fehlerbehebung und Optimierung
- Darstellung, Dokumentation und Diskussion der Messergebnisse von Lade- und Entladezyklen
- Kurzanleitung zur Unterstützung späterer Benutzer

6) **Gesamtbewertung und Fazit**

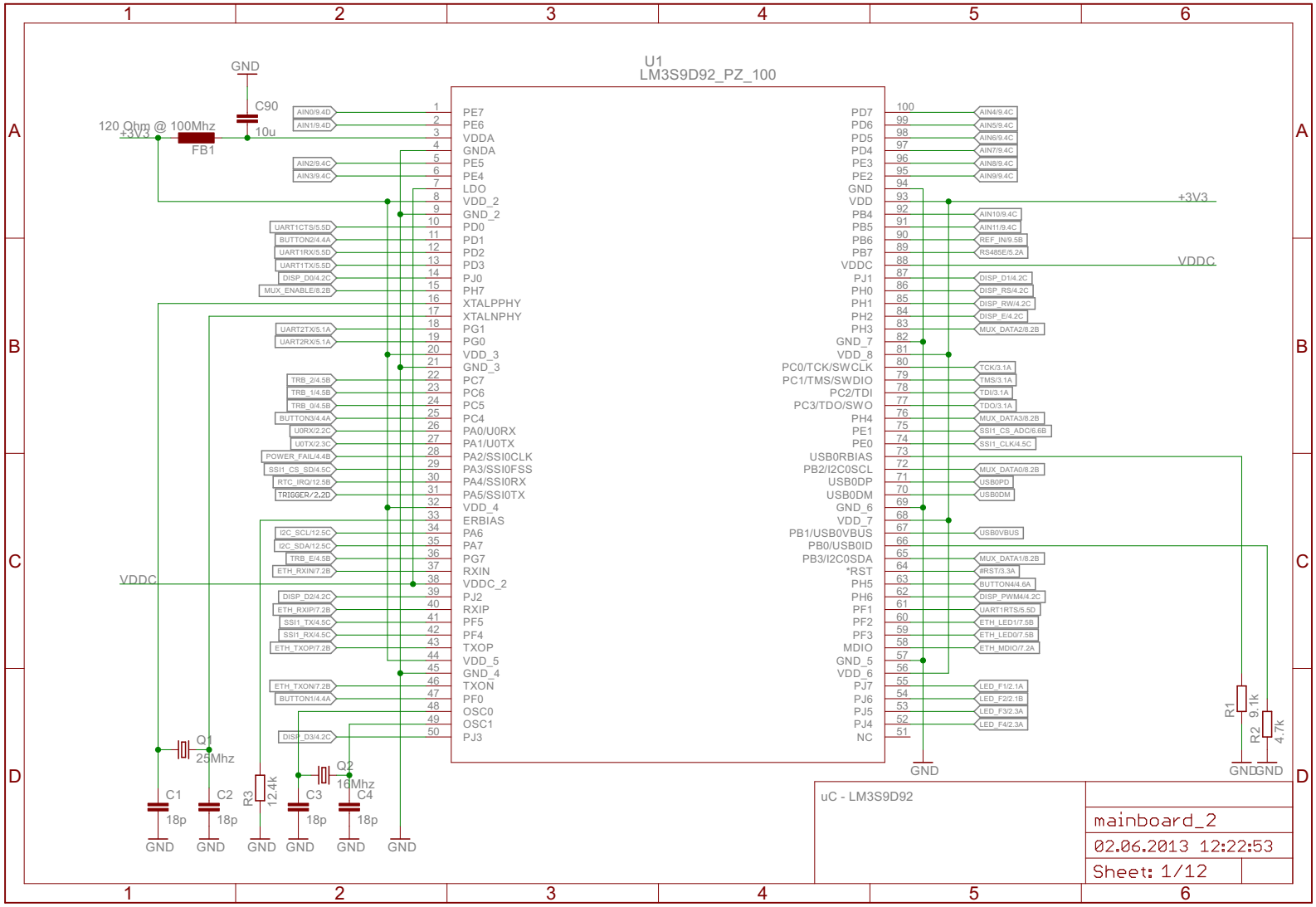
- Diskussion der Lösung in Bezug auf die Zielstellung
- Darstellung der Erprobungserfahrung und Bewertung der Ergebnisse
- Bewertung der Nutzbarkeit und Weiterführung

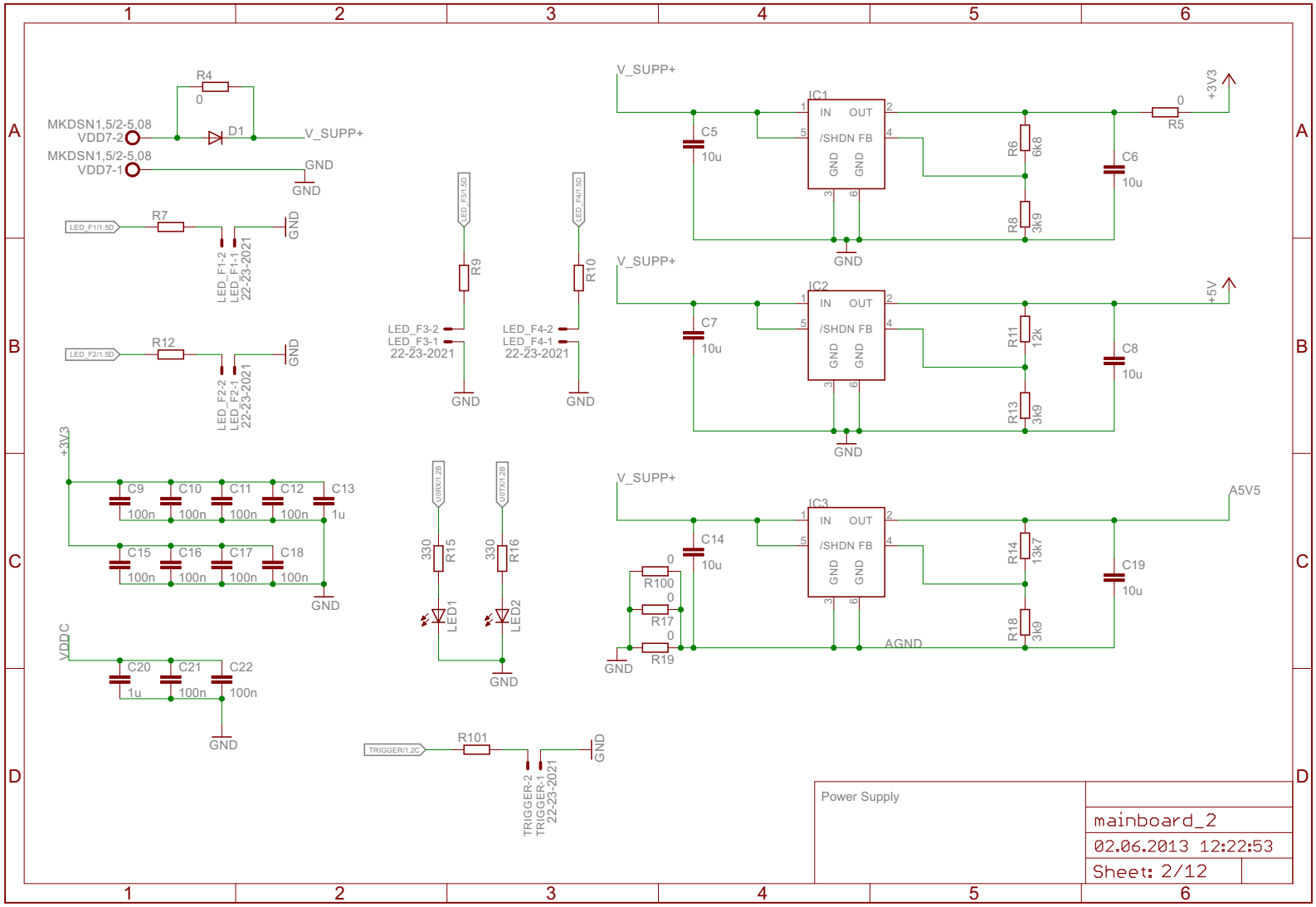
Dokumentation

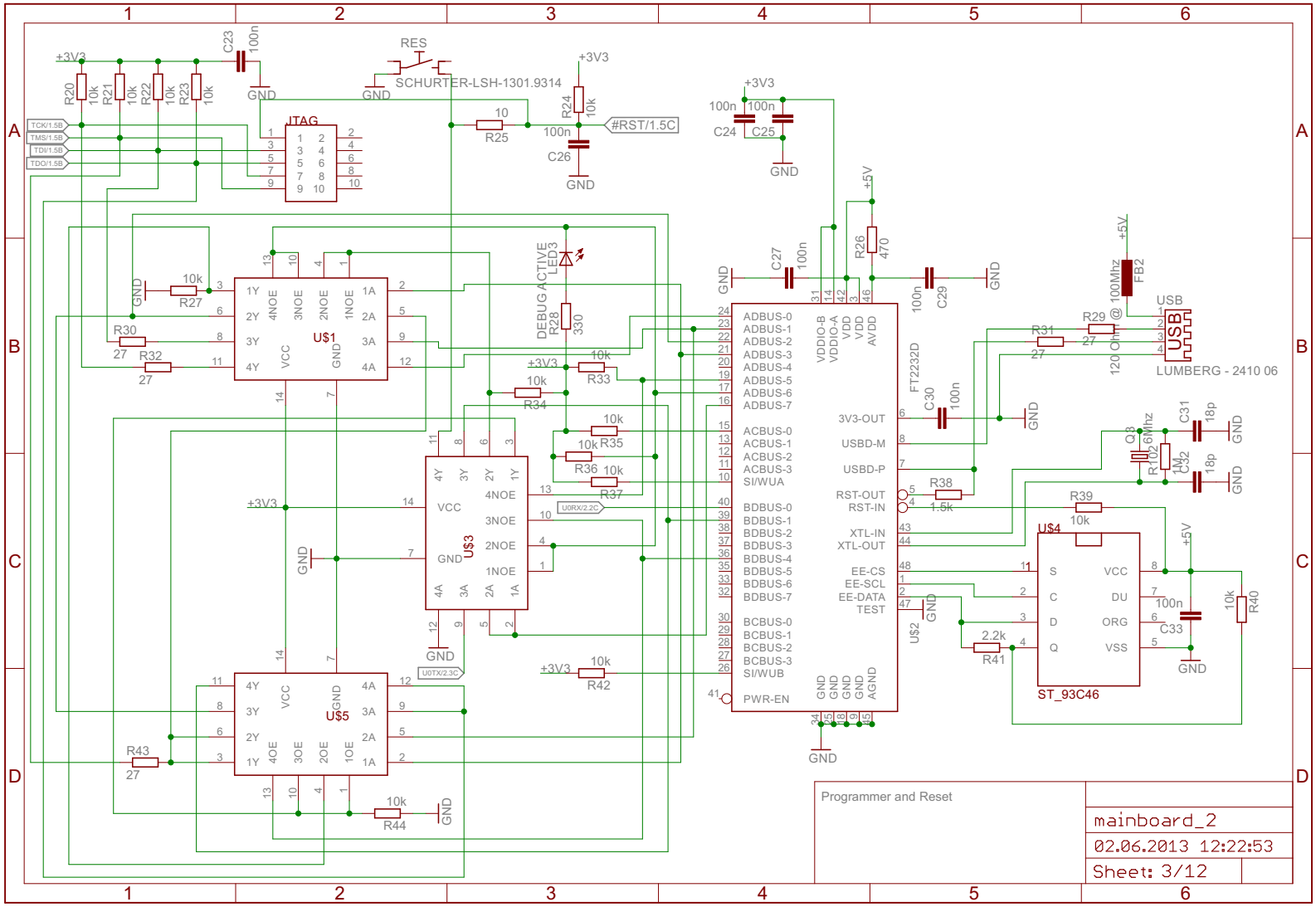
Die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Die gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die gesetzten Rahmenbedingungen, die Grundkonzeption, auftretende Probleme und wesentliche Entwurfsentscheidungen sollen beschrieben werden. Die Erprobungsergebnisse sind in exemplarischem Umfang zu erfassen und auszuwerten. Passende Unterstützungsunterlagen für die Nutzung und für weitere Arbeiten sind zu erstellen.

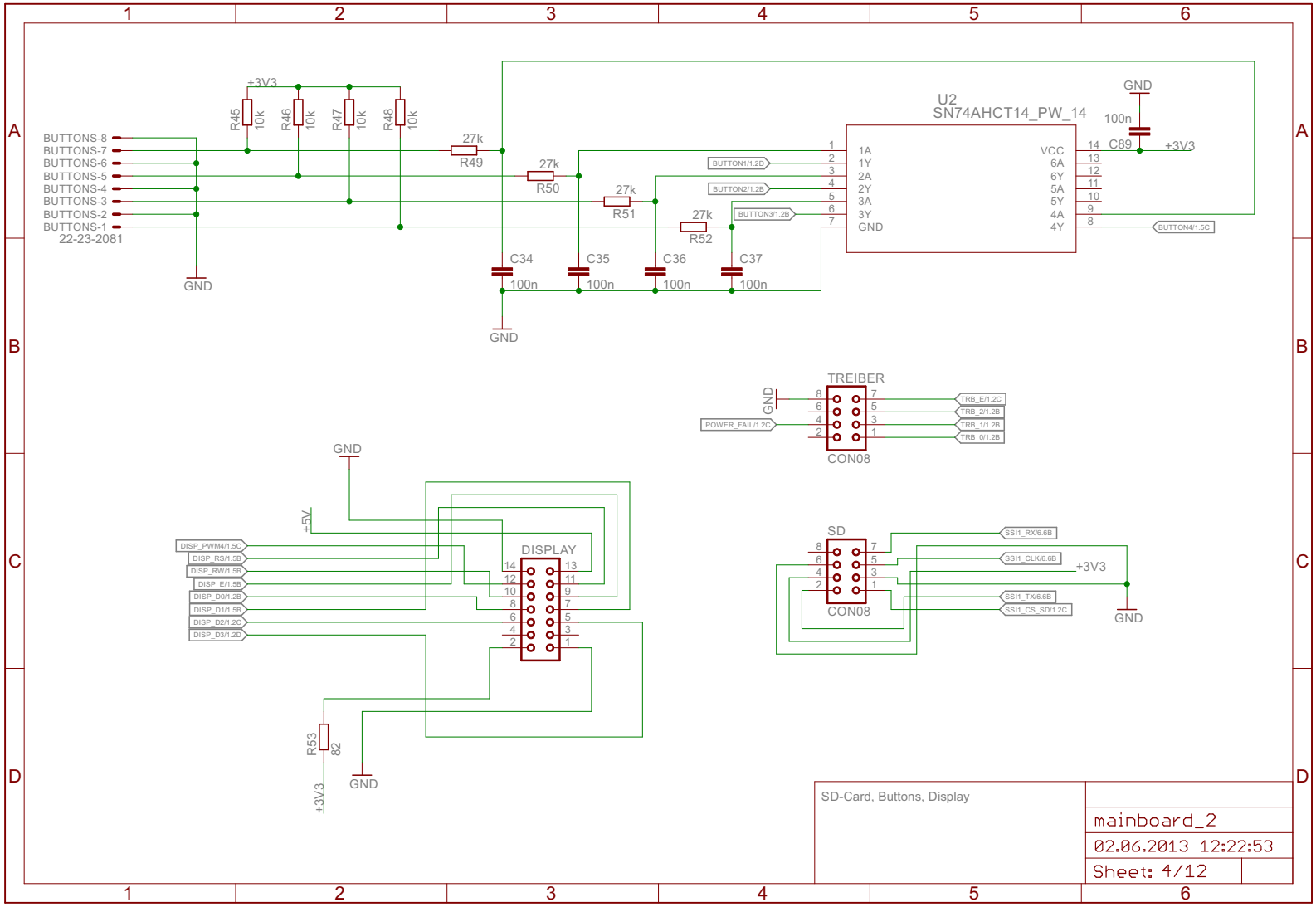
B. Schaltplan

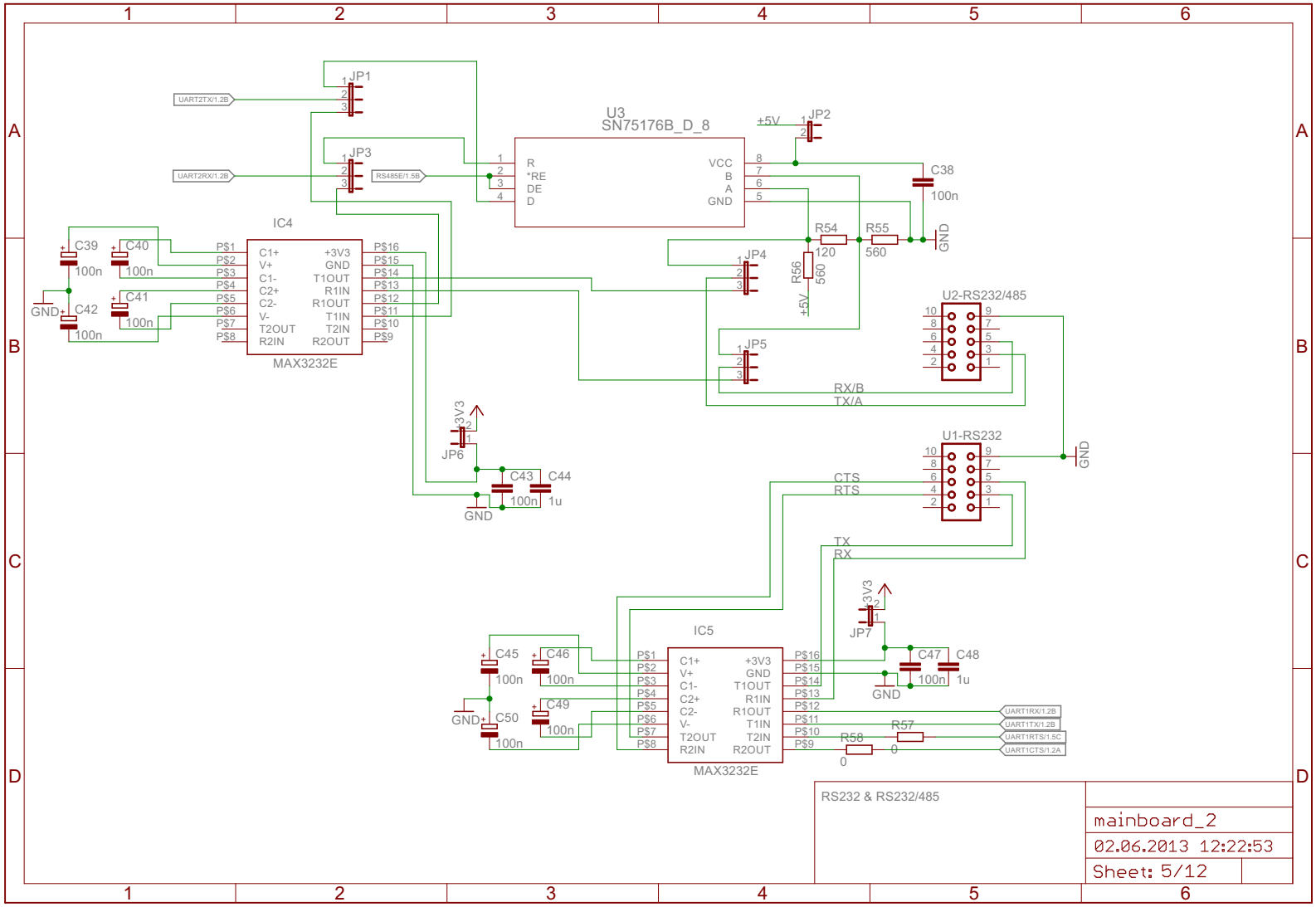
B.1. Mainboard

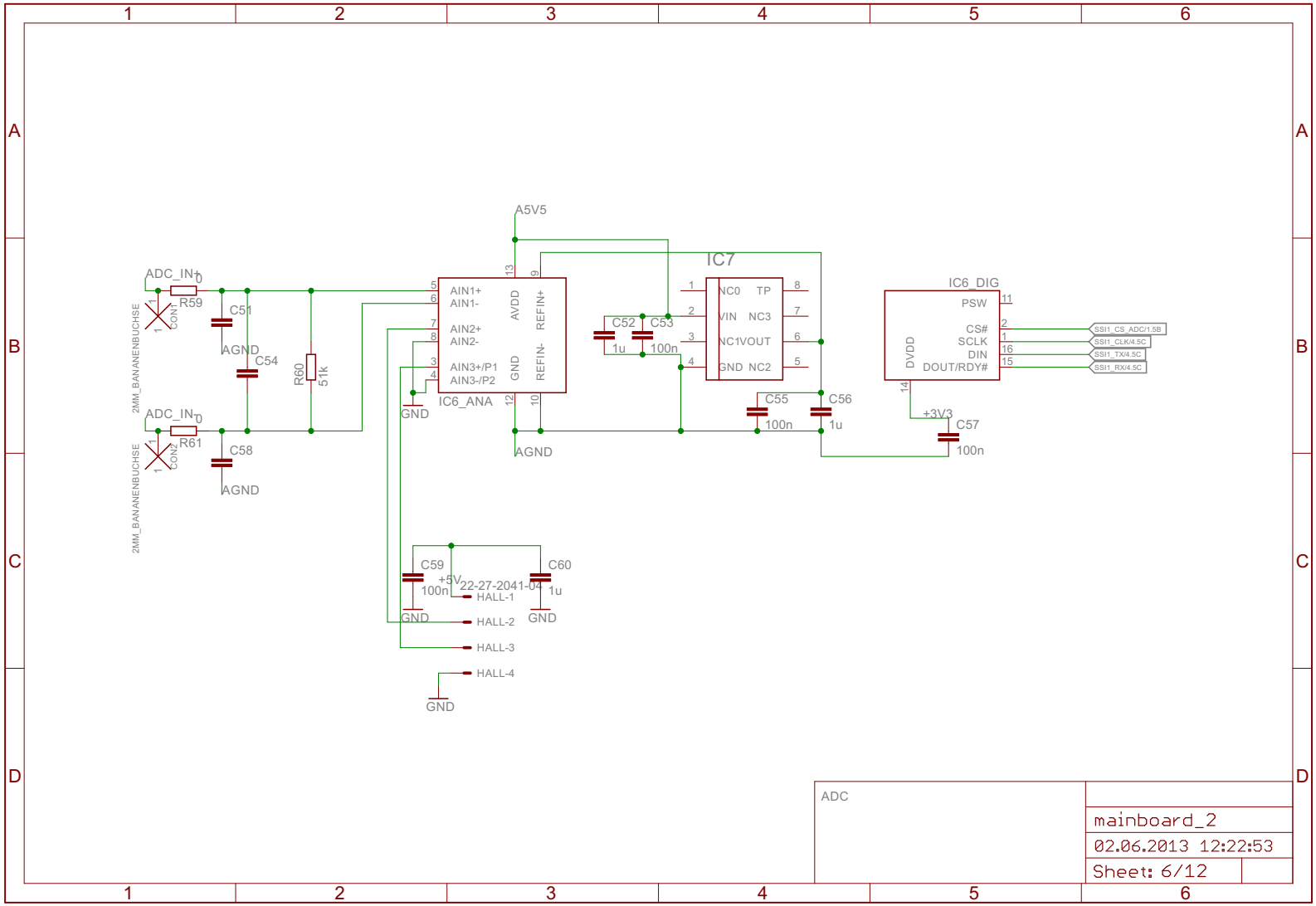

U1
LM3S9D92_PZ_100
GND
C90
10u
120 Ohm @ 100Mhz
+3V3
FB1
VDDC
Q1
25Mhz
Q2
16Mhz
C1
18p
C2
18p
R3
12.4k
C3
18p
C4
18p
R1
9.1k
R2
4.7k
+3V3
VDDC
uC - LM3S9D92
mainboard_2
02.06.2013 12:22:53
Sheet: 1/12

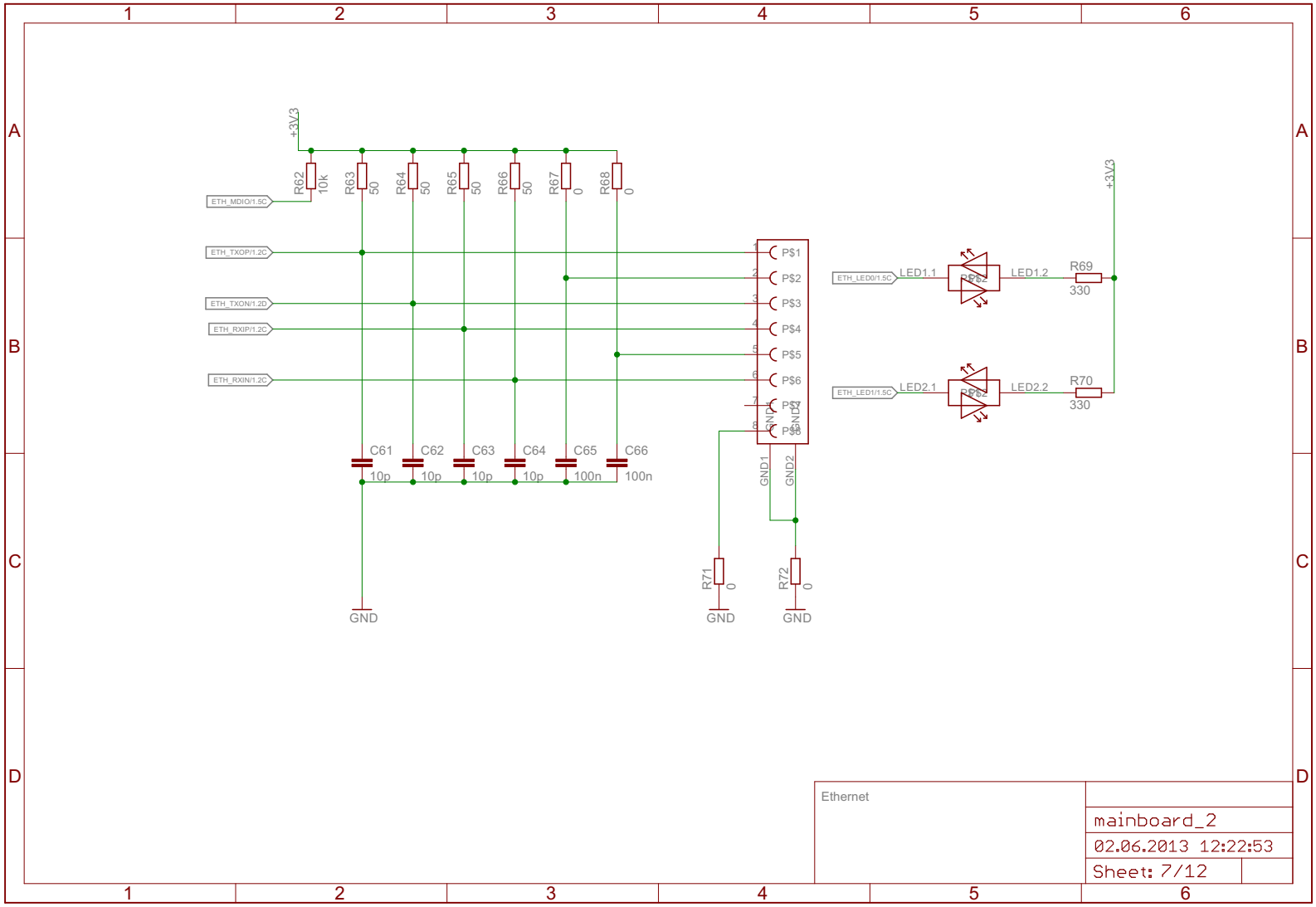

1
2
3
4
5
6
A
B
C
D
R4
0
MKDSN1,5/2-5,08
VDD7-2
D1
V_SUPP+
MKDSN1,5/2-5,08
VDD7-1
GND
GND
LED_F1/1.5D
R7
LED_F1-2
LED_F1-1
22-23-2021
GND
LED_F2/1.5D
R12
LED_F2-2
LED_F2-1
22-23-2021
GND
LED_F3/1.5D
R9
LED_F3-2
LED_F3-1
22-23-2021
GND
LED_F4/1.5D
R10
LED_F4-2
LED_F4-1
22-23-2021
GND
V_SUPP+
IC1
IN
OUT
/SHDN
FB
GND
GND
C5
10u
R6
6k8
R8
3k9
C6
10u
0
R5
+3V3
GND
V_SUPP+
IC2
C7
10u
R11
12k
R13
3k9
C8
10u
+5V
GND
+3V3
C9
C10
C11
C12
C13
100n
100n
100n
100n
1u
C15
C16
C17
C18
100n
100n
100n
100n
GND
VDDC
C20
C21
C22
1u
100n
100n
GND
U0RX/1.2B
U0TX/1.2B
330
R15
330
R16
LED1
LED2
GND
V_SUPP+
IC3
C14
10u
R14
13k7
R18
3k9
C19
10u
A5V5
0
R100
0
R17
0
R19
AGND
GND
TRIGGER/1.2C
R101
TRIGGER-2
TRIGGER-1
22-23-2021
GND
Power Supply
mainboard_2
02.06.2013 12:22:53
Sheet: 2/12

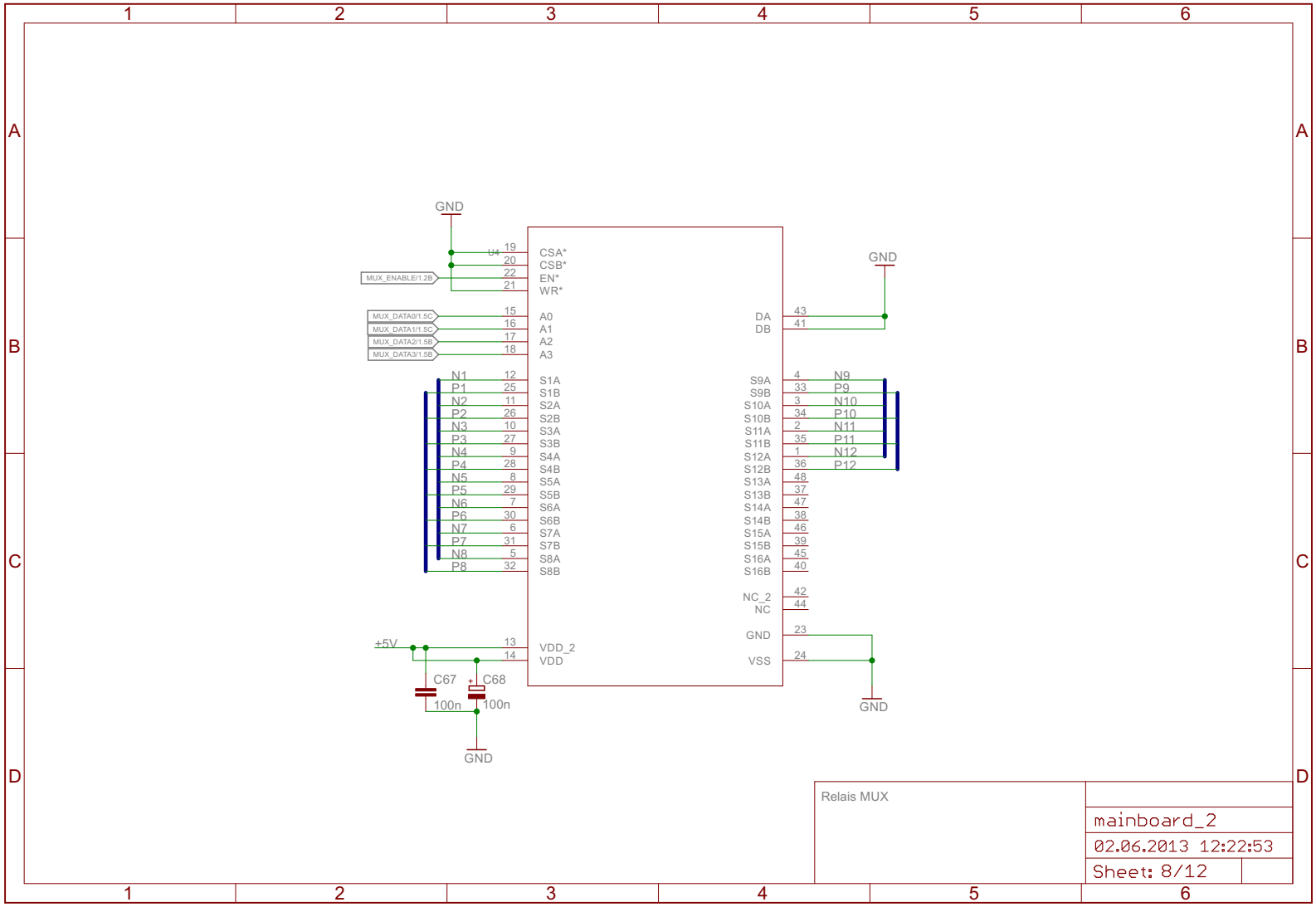

Programmer and Reset
mainboard_2
02.06.2013 12:22:53
Sheet: 3/12
JTAG
SCHURTER-LSH-1301.9314
RES
#RST/1.5C
DEBUG ACTIVE LED3
FT2232D
ST_93C46
LUMBERG - 2410 06
USB
U$1
U$2
U$3
U$4
U$5

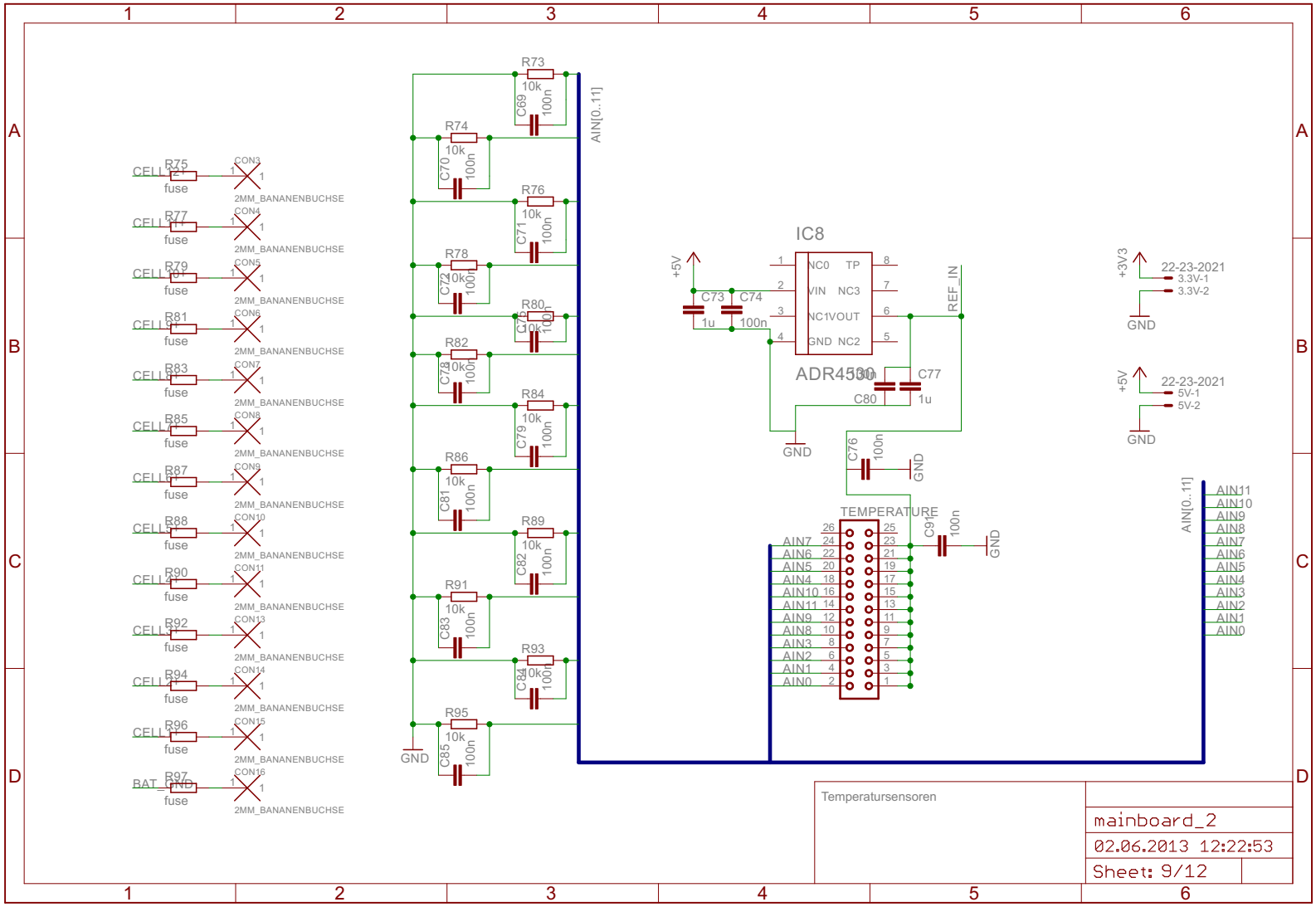

BUTTONS-8
BUTTONS-7
BUTTONS-6
BUTTONS-5
BUTTONS-4
BUTTONS-3
BUTTONS-2
BUTTONS-1
22-23-2081
GND
+3V3
R45 10k
R46 10k
R47 10k
R48 10k
27k R49
27k R50
27k R51
27k R52
C34 100n
C35 100n
C36 100n
C37 100n
U2
SN74AHCT14_PW_14
1A
1Y
2A
2Y
3A
3Y
GND
VCC
6A
6Y
5A
5Y
4A
4Y
100n C89
BUTTON1/1.2D
BUTTON2/1.2B
BUTTON3/1.2B
BUTTON4/1.5C
TREIBER
CON08
POWER_FAIL/1.2C
TRB_E/1.2C
TRB_2/1.2B
TRB_1/1.2B
TRB_0/1.2B
SD
SSI1_RX/6.6B
SSI1_CLK/6.6B
SSI1_TX/6.6B
SSI1_CS_SD/1.2C
DISPLAY
+5V
DISP_PWM4/1.5C
DISP_RS/1.5B
DISP_RW/1.5B
DISP_E/1.5B
DISP_D0/1.2B
DISP_D1/1.5B
DISP_D2/1.2C
DISP_D3/1.2D
R53 82
SD-Card, Buttons, Display
mainboard_2
02.06.2013 12:22:53
Sheet: 4/12

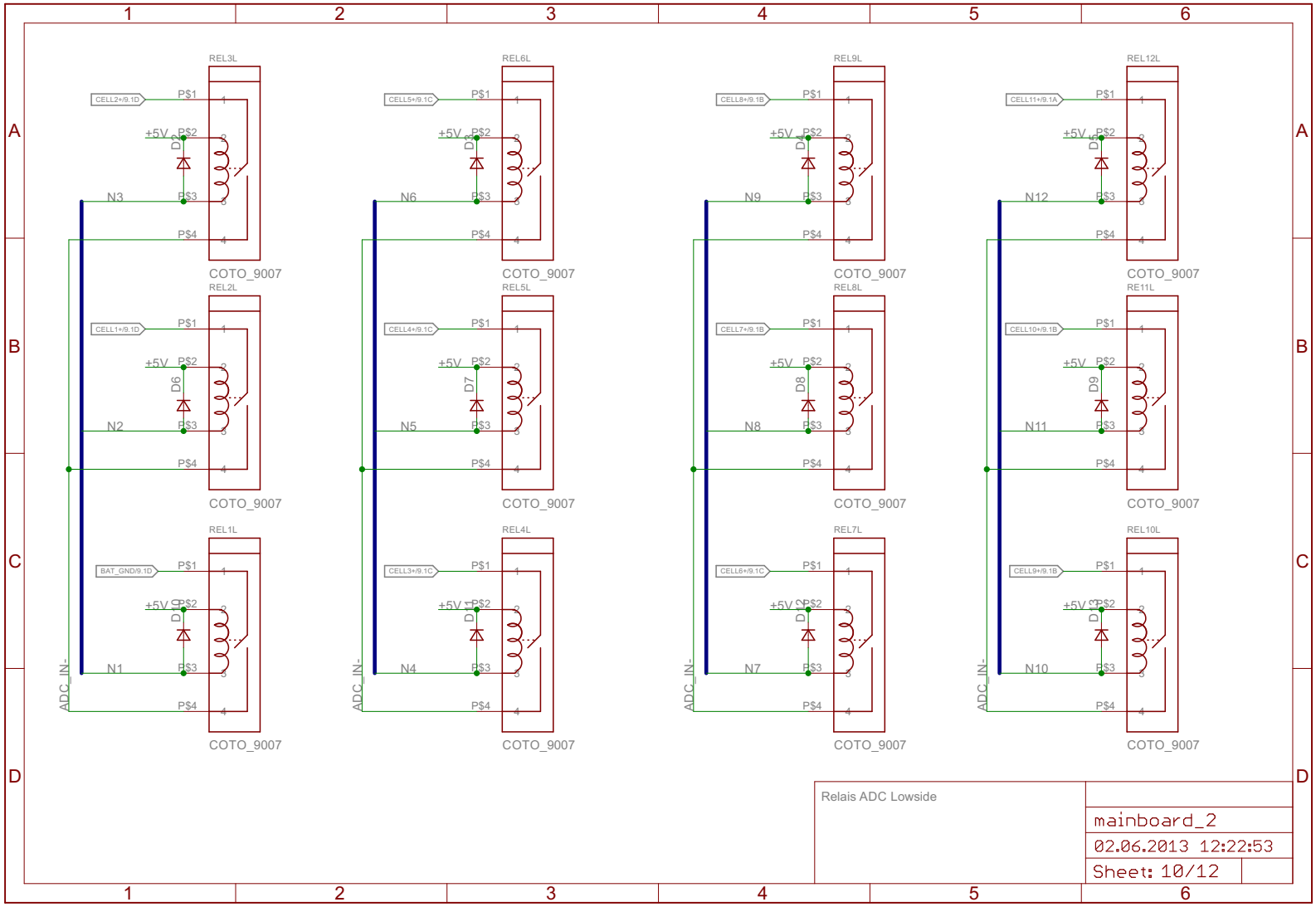

JP1
UART2TX/1.2B
JP3
UART2RX/1.2B
RS485E/1.5B
U3
SN75176B_D_8
R
*RE
DE
D
VCC
B
A
GND
+5V
JP2
C38
100n
R54
120
R55
560
R56
560
JP4
JP5
RX/B
TX/A
U2-RS232/485
IC4
C39
C40
C41
C42
100n
GND
P$1
P$2
P$3
P$4
P$5
P$6
P$7
P$8
C1+
V+
C1-
C2+
C2-
V-
T2OUT
R2IN
+3V3
GND
T1OUT
R1IN
R1OUT
T1IN
T2IN
R2OUT
P$16
P$15
P$14
P$13
P$12
P$11
P$10
P$9
MAX3232E
3V3
JP6
C43
C44
1u
U1-RS232
CTS
RTS
TX
RX
JP7
IC5
C45
C46
C47
C48
C49
C50
R57
R58
0
UART1RX/1.2B
UART1TX/1.2B
UART1RTS/1.5C
UART1CTS/1.2A
RS232 & RS232/485
mainboard_2
02.06.2013 12:22:53
Sheet: 5/12

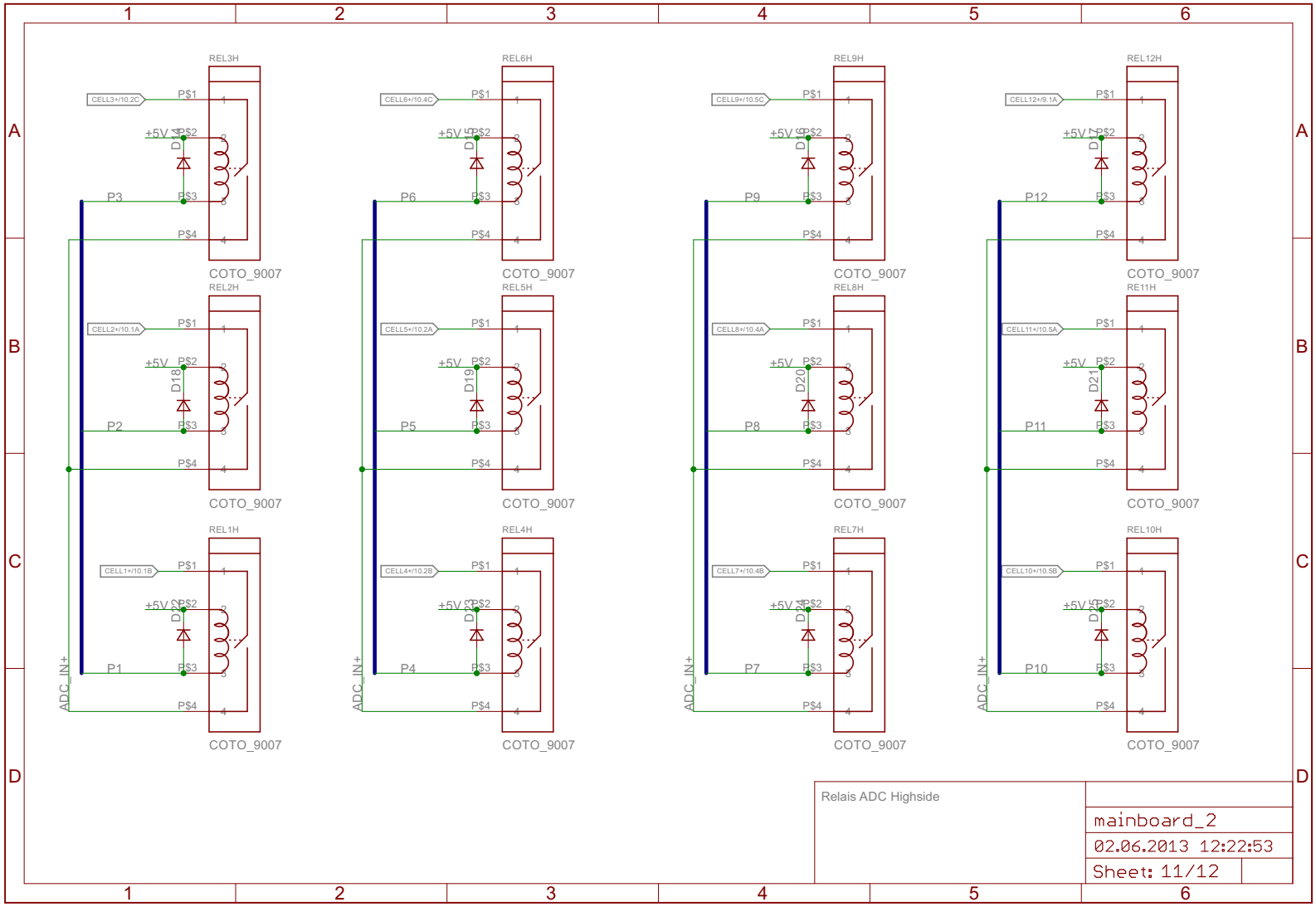

1
2
3
4
5
6
A
B
C
D
A5V5
IC7
ADC_IN0
R59
C51
AGND
C54
R60
51k
2MM_BANANENBUCHSE
CON1
ADC_IN1
R61
C58
AGND
AIN1+
AIN1-
AIN2+
AIN2-
AIN3+/P1
AIN3-/P2
AVDD
GND
REFIN+
REFIN-
IC6_ANA
AGND
C52
1u
C53
100n
NC0
TP
VIN
NC3
NC1
VOUT
GND
NC2
C55
100n
C56
1u
IC6_DIG
PSW
CS#
SCLK
DIN
DOUT/RDY#
DVDD
SSI1_CS_ADC/1.5B
SSI1_CLK/4.5C
SSI1_TX/4.5C
SSI1_RX/4.5C
+3V3
C57
100n
C59
100n
+5V
22-27-2041-04
HALL-1
HALL-2
HALL-3
HALL-4
C60
1u
GND
ADC
mainboard_2
02.06.2013 12:22:53
Sheet: 6/12

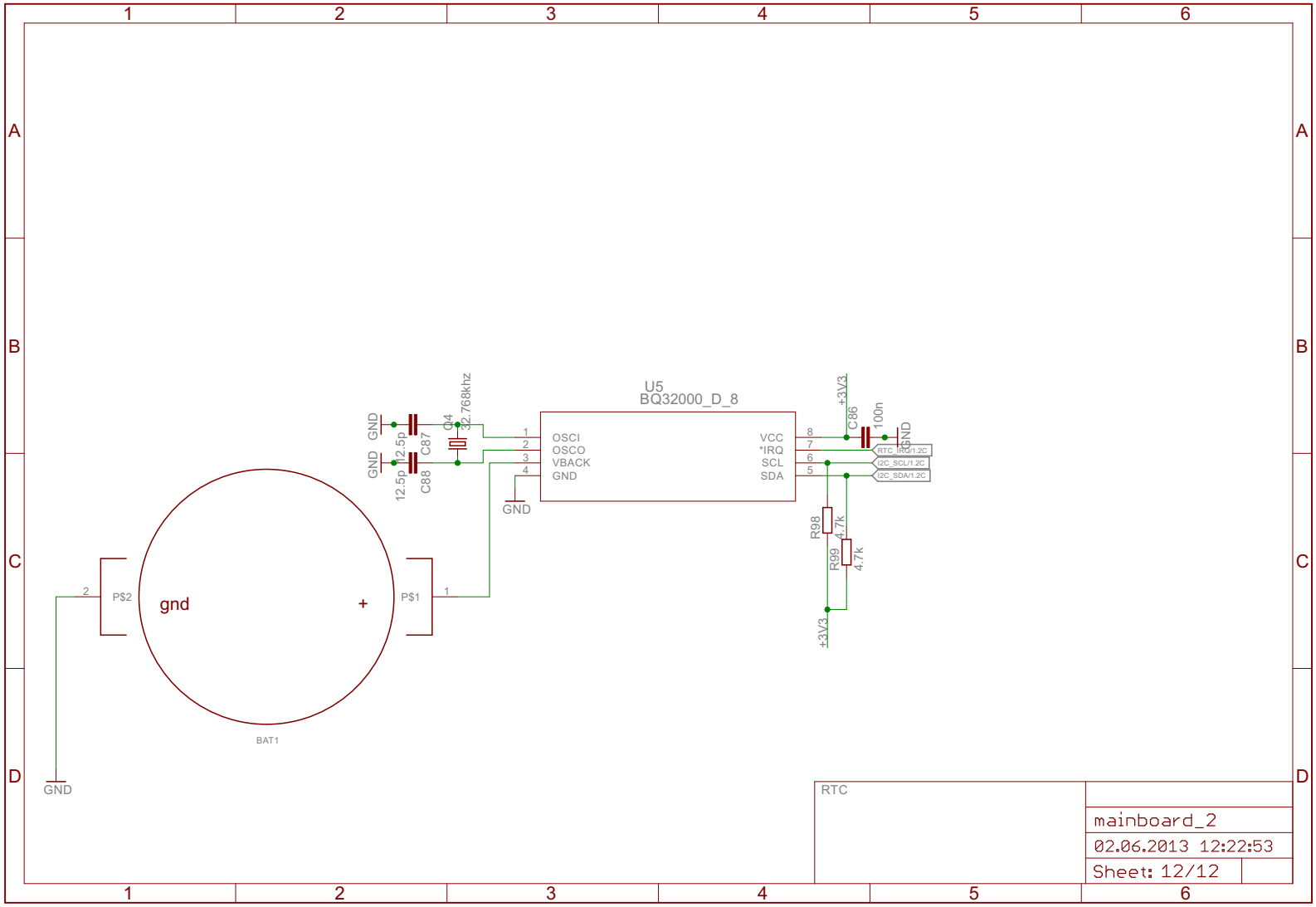

1
2
3
4
5
6
A
B
C
D
+3V3
R62
10k
R63
50
R64
50
R65
50
R66
50
R67
0
R68
0
ETH_MDIO/1.5C
ETH_TXOP/1.2C
ETH_TXON/1.2D
ETH_RXIP/1.2C
ETH_RXIN/1.2C
P$1
P$2
P$3
P$4
P$5
P$6
P$7
P$8
GND1
GND2
C61
10p
C62
10p
C63
10p
C64
10p
C65
100n
C66
100n
R71
0
R72
0
GND
ETH_LED0/1.5C
LED1.1
LED1.2
R69
330
ETH_LED1/1.5C
LED2.1
LED2.2
R70
330
Ethernet
mainboard_2
02.06.2013 12:22:53
Sheet: 7/12


1
2
3
4
5
6
A
B
C
D
GND
U4
CSA*
CSB*
EN*
WR*
MUX_ENABLE/1.2B
MUX_DATA0/1.5C
MUX_DATA1/1.5C
MUX_DATA2/1.5B
MUX_DATA3/1.5B
A0
A1
A2
A3
S1A
S1B
S2A
S2B
S3A
S3B
S4A
S4B
S5A
S5B
S6A
S6B
S7A
S7B
S8A
S8B
VDD_2
VDD
DA
DB
S9A
S9B
S10A
S10B
S11A
S11B
S12A
S12B
S13A
S13B
S14A
S14B
S15A
S15B
S16A
S16B
NC_2
NC
GND
VSS
N1
P1
N2
P2
N3
P3
N4
P4
N5
P5
N6
P6
N7
P7
N8
P8
N9
P9
N10
P10
N11
P11
N12
P12
+5V
C67
100n
C68
100n
Relais MUX
mainboard_2
02.06.2013 12:22:53
Sheet: 8/12

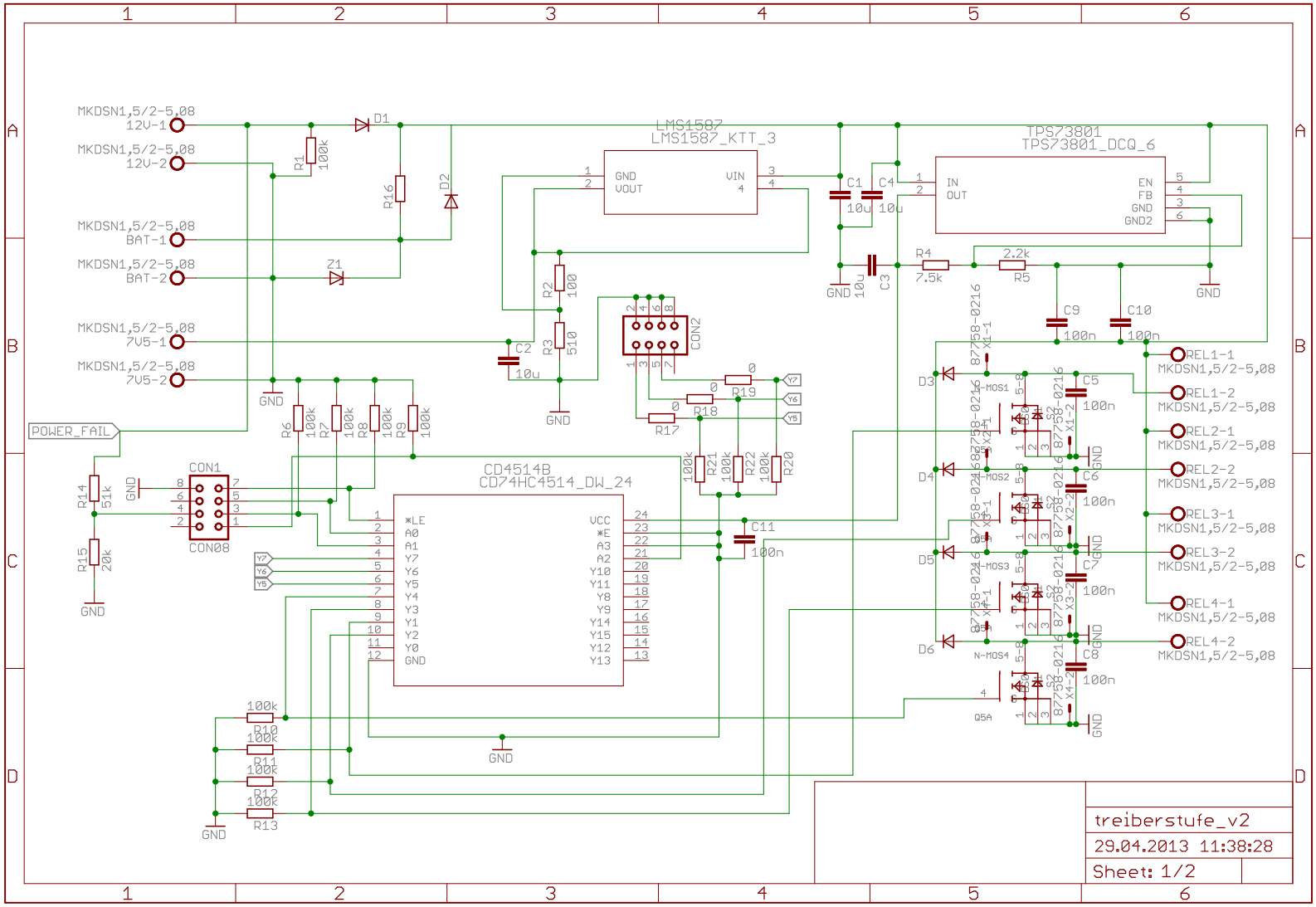

IC8
ADR4530
TEMPERATURE
AIN[0..11]
REF_IN
+5V
+3V3
GND
22-23-2021
3.3V-1
3.3V-2
5V-1
5V-2
2MM_BANANENBUCHSE
fuse
Temperatursensoren
mainboard_2
02.06.2013 12:22:53
Sheet: 9/12

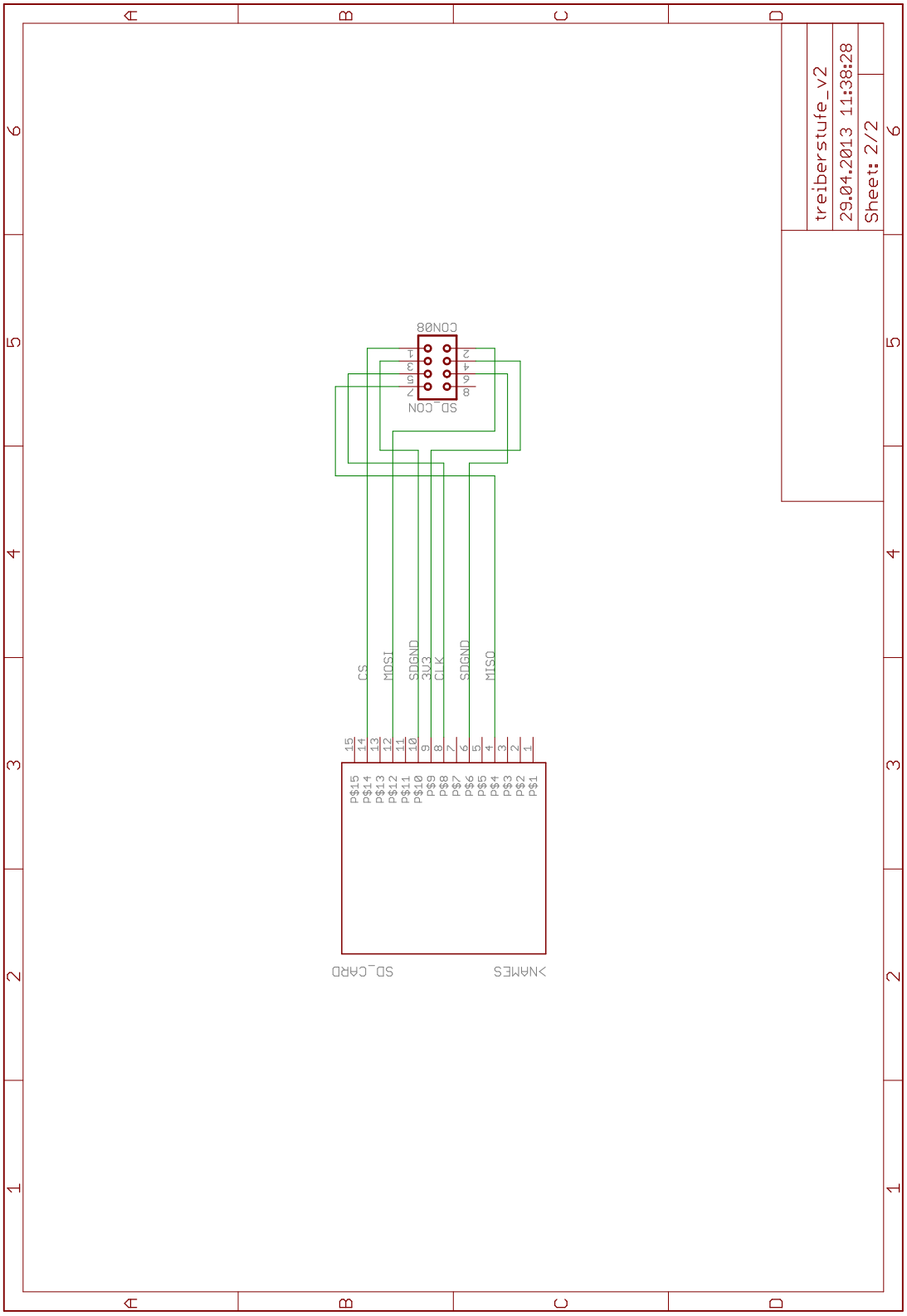

REL3L
CELL2+/9.1D
P$1
+5V
P$2
D2
N3
P$3
P$4
COTO_9007
REL6L
CELL5+/9.1C
N6
REL9L
CELL8+/9.1B
N9
REL12L
CELL11+/9.1A
N12
REL2L
CELL1+/9.1D
D6
N2
REL5L
CELL4+/9.1C
D7
N5
REL8L
CELL7+/9.1B
D8
N8
RE11L
CELL10+/9.1B
D9
N11
REL1L
BAT_GND/9.1D
D10
N1
ADC_IN-
REL4L
CELL3+/9.1C
D11
N4
REL7L
CELL6+/9.1C
D12
N7
REL10L
CELL9+/9.1B
D13
N10
Relais ADC Lowside
mainboard_2
02.06.2013 12:22:53
Sheet: 10/12


REL3H
REL6H
REL9H
REL12H
REL2H
REL5H
REL8H
RE11H
REL1H
REL4H
REL7H
REL10H
COTO_9007
CELL3+/10.2C
CELL6+/10.4C
CELL9+/10.5C
CELL12+/9.1A
CELL2+/10.1A
CELL5+/10.2A
CELL8+/10.4A
CELL11+/10.5A
CELL1+/10.1B
CELL4+/10.2B
CELL7+/10.4B
CELL10+/10.5B
+5V
ADC_IN+
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
D14
D15
D16
D17
D18
D19
D20
D21
D22
D23
D24
D25
Relais ADC Highside
mainboard_2
02.06.2013 12:22:53
Sheet: 11/12


1
2
3
4
5
6
A
B
C
D
U5
BQ32000_D_8
OSCI
OSCO
VBACK
GND
VCC
*IRQ
SCL
SDA
C87
12.5p
C88
12.5p
Q4
32.768khz
C86
100n
R98
4.7k
R99
4.7k
+3V3
GND
RTC_IRQ/1.2C
I2C_SCL/1.2C
I2C_SDA/1.2C
P$1
P$2
gnd
+
BAT1
RTC
mainboard_2
02.06.2013 12:22:53
Sheet: 12/12

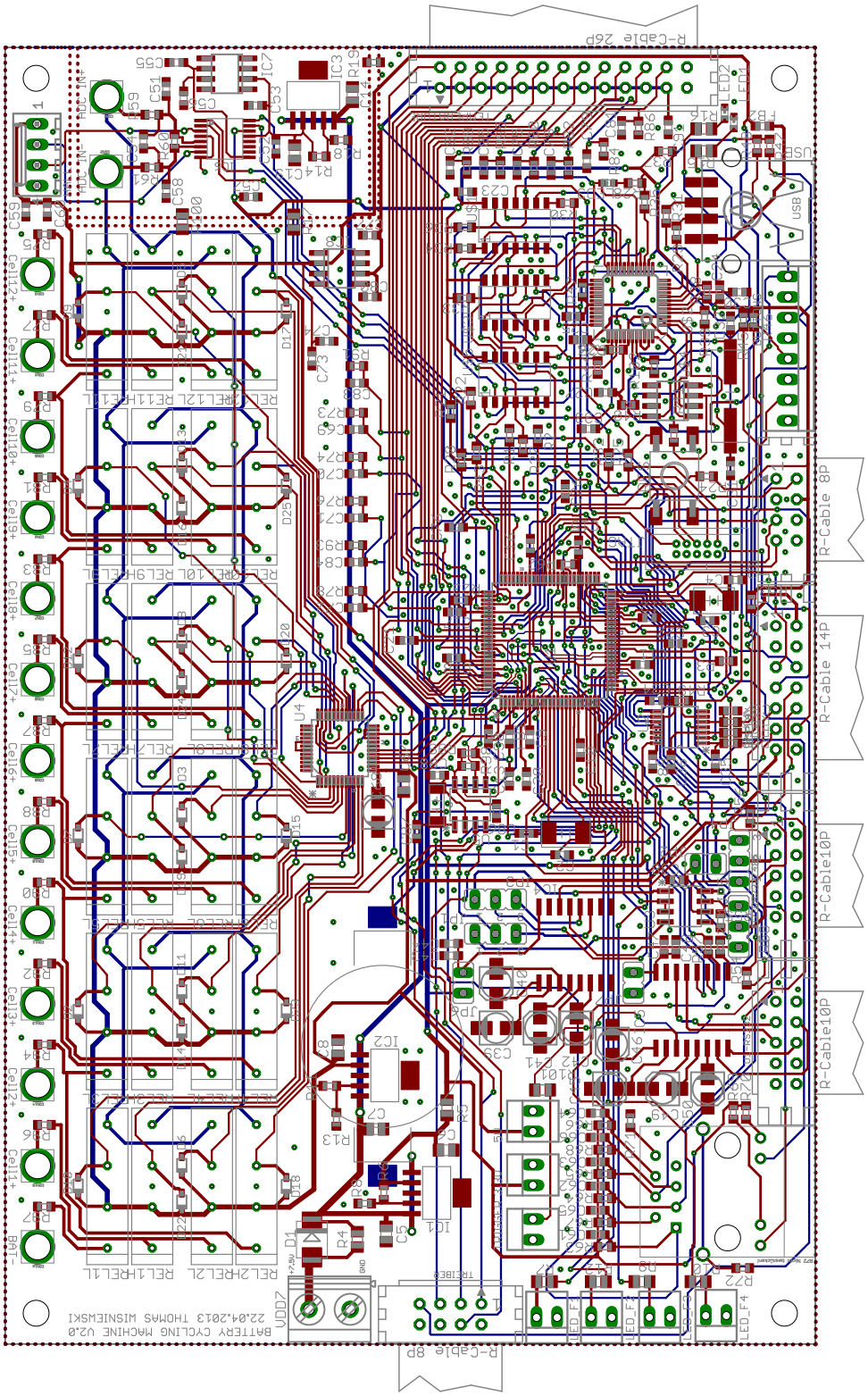
B.2. Lasttreibermodul/SD-Kartenhalter

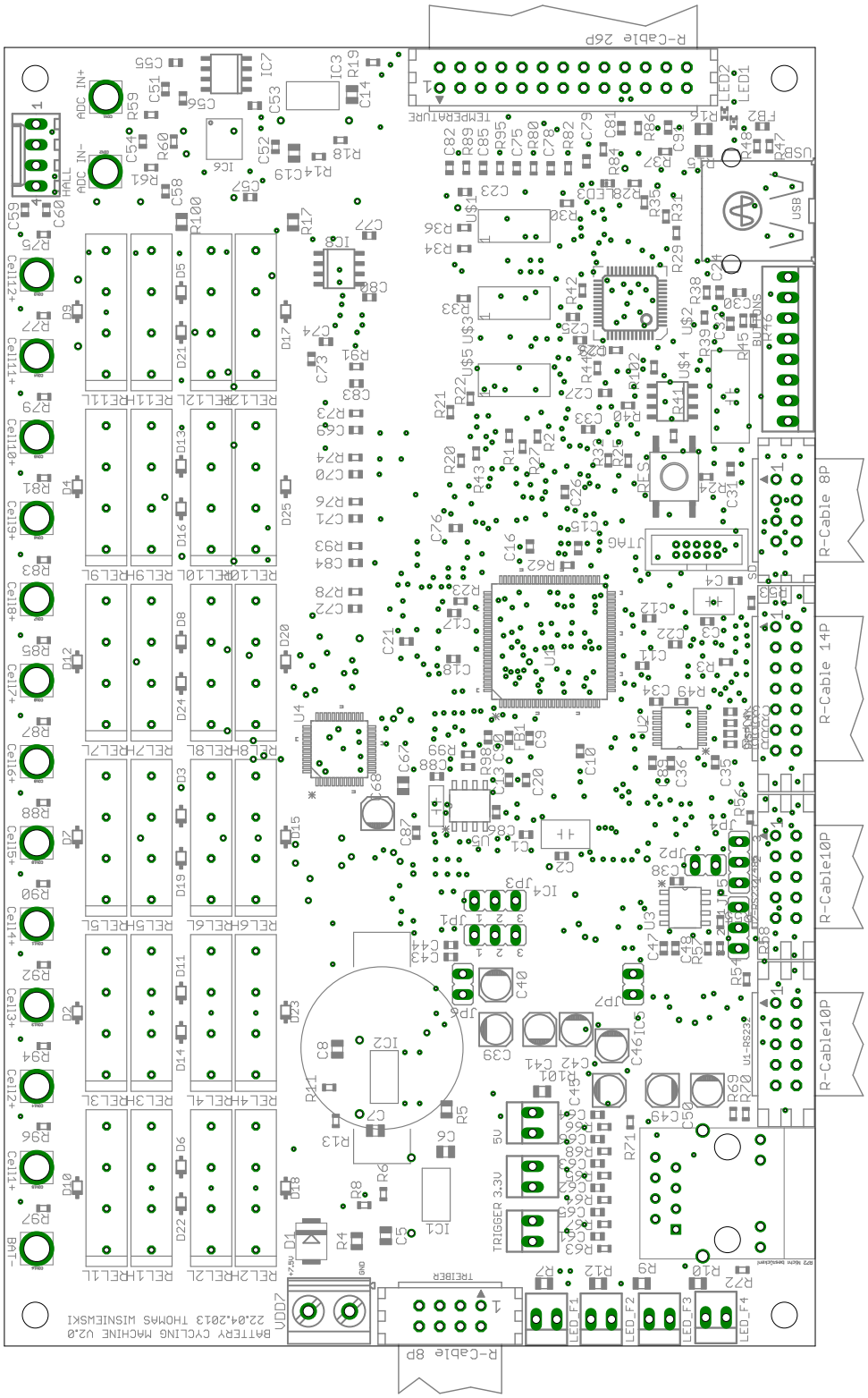

MKDSN1,5/2-5,08
12V-1
MKDSN1,5/2-5,08
12V-2
MKDSN1,5/2-5,08
BAT-1
MKDSN1,5/2-5,08
BAT-2
MKDSN1,5/2-5,08
7V5-1
MKDSN1,5/2-5,08
7V5-2
D1
D2
Z1
R1 100k
R16
LMS1587
LMS1587_KTT_3
GND
VOUT
VIN
C1 10u
C4 10u
C3 10u
TPS73801
TPS73801_DCQ_6
IN
OUT
EN
FB
GND
GND2
R4 7.5k
R5 2.2k
C9 100n
C10 100n
R2 100
R3 510
C2 10u
CON2
R19 0
R18 0
R17 0
R6 100k
R7 100k
R8 100k
R9 100k
POWER_FAIL
CON1
CON08
R14 51k
R15 20k
CD4514B
CD74HC4514_DW_24
*LE
A0
A1
Y7
Y6
Y5
Y4
Y3
Y1
Y2
Y0
VCC
*E
A3
A2
Y10
Y11
Y8
Y9
Y14
Y15
Y12
Y13
R21 100k
R22 100k
R20 100k
C11 100n
R10 100k
R11 100k
R12 100k
R13 100k
87758-0216
X1-1
X1-2
X2-1
X2-2
X3-1
X3-2
X4-1
X4-2
N-MOS1
N-MOS2
N-MOS3
N-MOS4
D3
D4
D5
D6
C5 100n
C6 100n
C7 100n
C8 100n
REL1-1
REL1-2
REL2-1
REL2-2
REL3-1
REL3-2
REL4-1
REL4-2
MKDSN1,5/2-5,08
treiberstufe_v2
29.04.2013 11:38:28
Sheet: 1/2

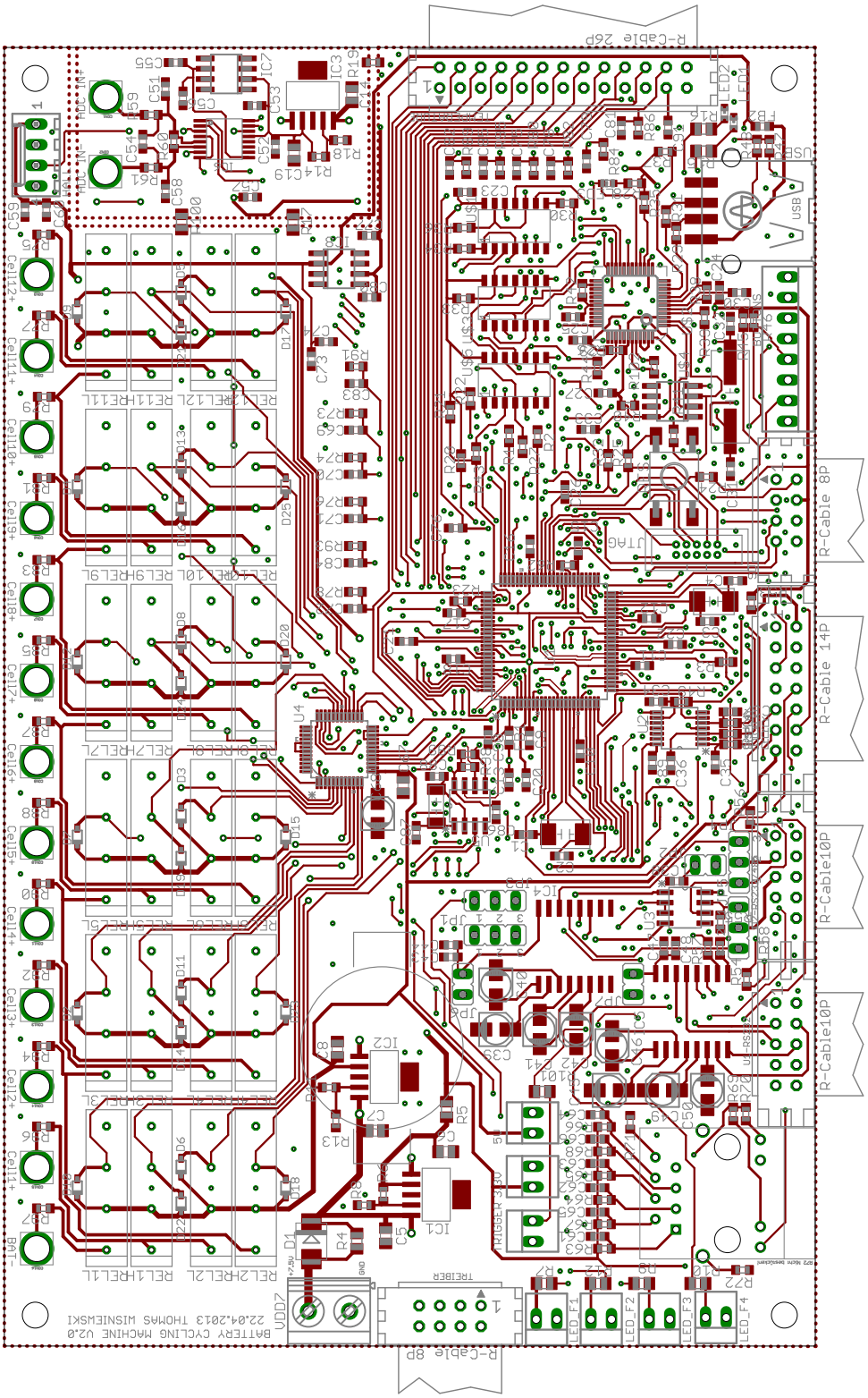

treiberstufe_v2
29.04.2013 11:38:28
Sheet: 2/2
CON08
SD_CON
CS
MOSI
SDGND
3V3
CLK
SDGND
MISO
SD_CARD
>NAMES

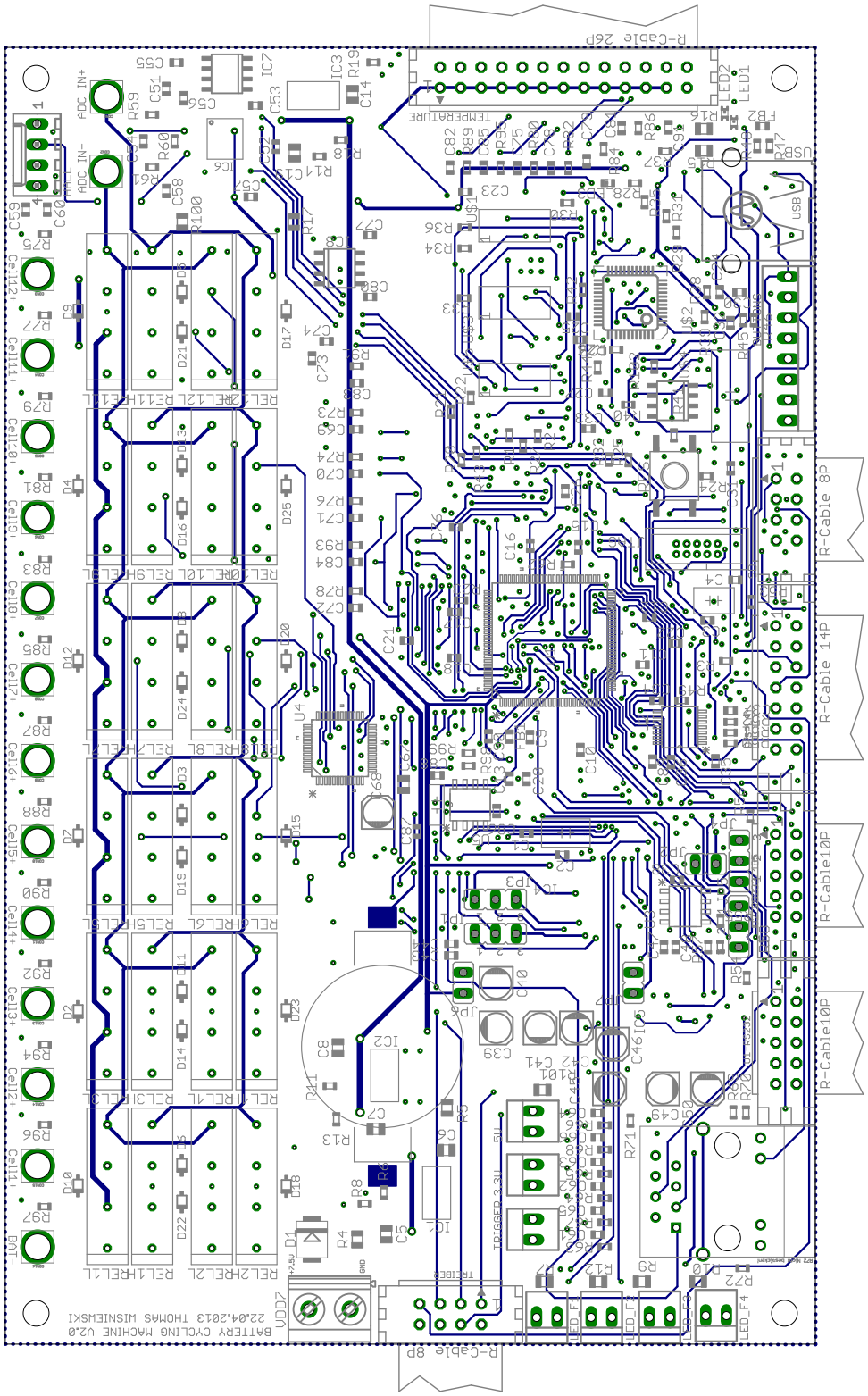
C. Platinenlayout

C.1. Mainboard

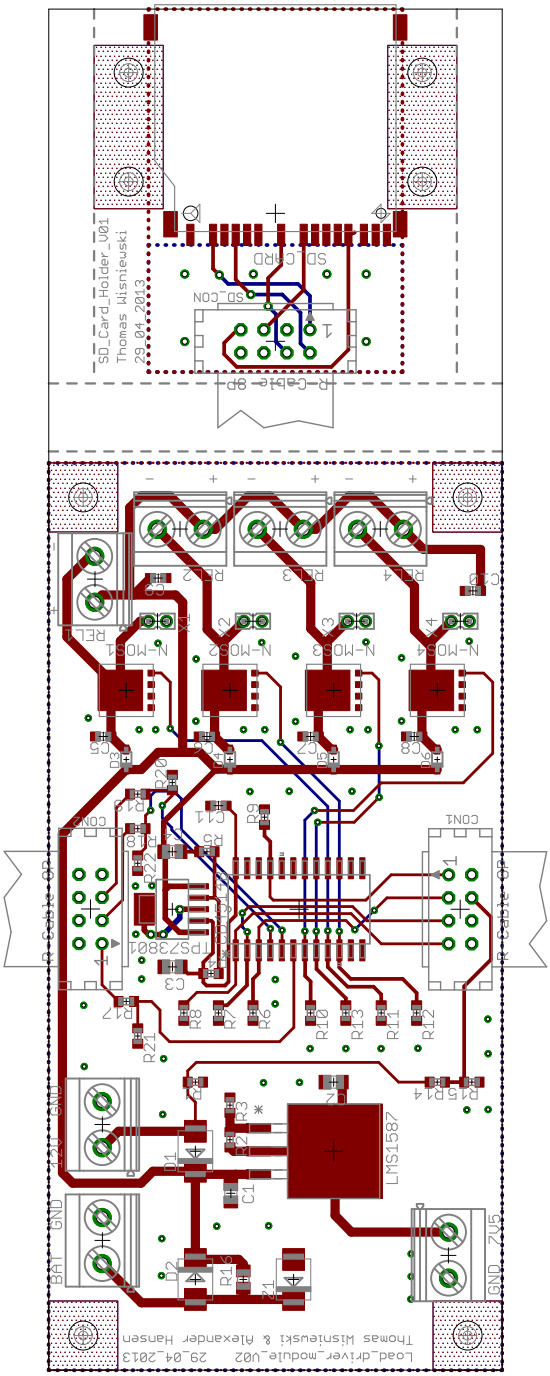

R-Cable 26P
R-Cable 8P
R-Cable 14P
R-Cable10P
R-Cable10P
R-Cable 8P
BATTERY CYCLING MACHINE V2.0
22.04.2013 THOMAS WISNIEWSKI

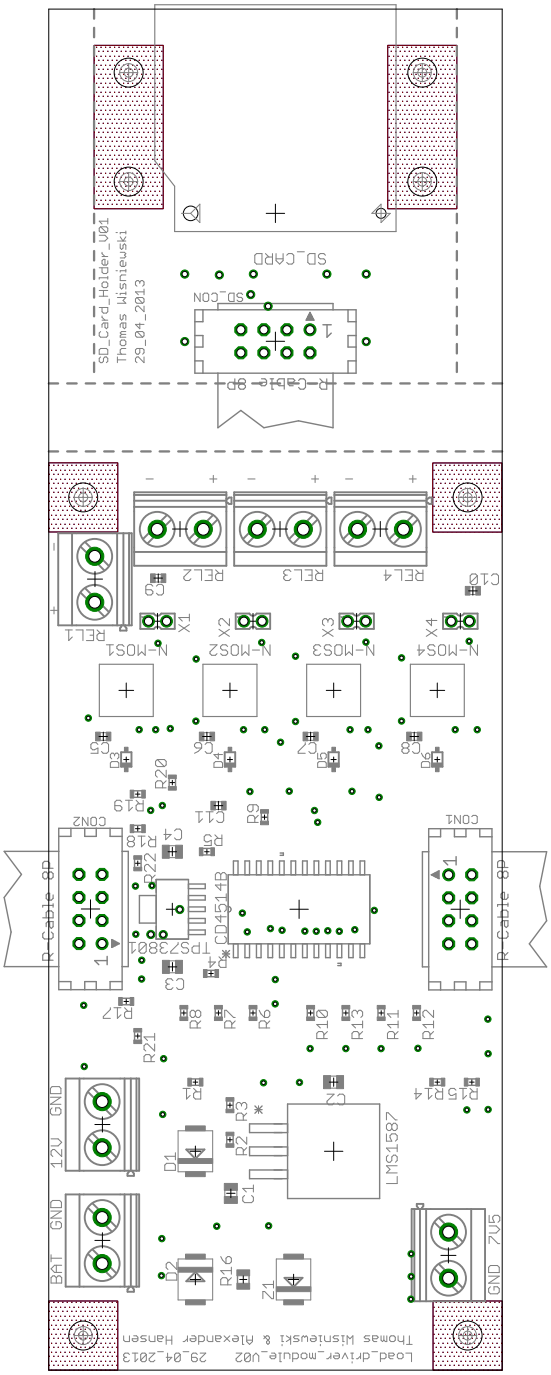

R-Cable 26P
R-Cable 8P
R-Cable 14P
R-Cable10P
R-Cable10P
R-Cable 8P
BATTERY CYCLING MACHINE V2.0
22.04.2013 THOMAS WISNIEWSKI
TRIGGER 3.3V
5V
LED_F1
LED_F2
LED_F3
LED_F4


R-Cable 26P
R-Cable 8P
R-Cable 14P
R-Cable10P
R-Cable10P
R-Cable 8P
BATTERY CYCLING MACHINE V2.0
22.04.2013 THOMAS WISNIEWSKI


R-Cable 26P
R-Cable 8P
R-Cable 14P
R-Cable10P
R-Cable10P
R-Cable 8P
BATTERY CYCLING MACHINE V2.0
22.04.2013 THOMAS WISNIEWSKI

C.2. Lasttreibermodul/SD-Kartenhalter


SD_Card_Holder_V01
Thomas Wisniewski
29.04.2013
SD_CARD
SD_CON
R_Cable_8P
RELAY
N-MOS1
N-MOS2
N-MOS3
N-MOS4
CON2
CON1
R_Cable_OP
TPS73801
LM51587
BAT_GND
GND_ZW5
Load_driver_module_V02 29_04_2013
Thomas Wisniewski & Alexander Hansen


SD_Card_Holder_V01
Thomas Wisniewski
29_04_2013
SD_CARD
SD_CON
R_Cable_8P
RELL1
RELL2
RELL3
RELL4
N-MOS1
N-MOS2
N-MOS3
N-MOS4
CON2
CON1
R-Cable_8P
TPS73801
CD4514B
LMS1587
12V GND
BAT GND
GND 7V5
Load_driver_module_V02 29_04_2013
Thomas Wisniewski & Alexander Hansen

D. Controller Source Files

clocktimer.h

```
 1 /*
 2  * timer.h
 3  *
 4  *  Created on: 04.03.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef TIMER_H_
 9 #define TIMER_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void init_clock_timer(void);
17 void TimerClock(void);
18 void reinit_timer0(void);
19
20 int Cmd_start(int argc, char *argv[]);
21 int Cmd_stop(int argc, char *argv[]);
22
23 #endif /* TIMER_H_ */
24
```

clocktimer.c

```
 1 /*
 2 Project:                          battery_cycling_sw_v3
 3 File:                             clocktimer.c
 4
 5 Auhtor:                           Thomas Wisnewski
 6 Credits:                          Matthias Schneider
 7                                   Tobias Steinmann
 8                                   Fabian Schwartau
 9                                   Stellaris Ware
10
11 last modified:                   2013/03/18
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 File for Timer
24
25 */
26
27 /*****************************************************************************
28 *
29 * Includings
30 *
31 *****************************************************************************/
32 #include <string.h>
33 #include "driverlib/rom.h"
34 #include "third_party/fatfs/src/ff.h"
35 #include "third_party/fatfs/src/diskio.h"
36 #include "inc/hw_ints.h"
37 #include "inc/hw_memmap.h"
38 #include "inc/hw_types.h"
39 #include "driverlib/debug.h"
40 #include "driverlib/gpio.h"
41 #include "driverlib/interrupt.h"
42 #include "driverlib/pin_map.h"
43 #include "driverlib/rom.h"
44 #include "driverlib/sysctl.h"
45 #include "driverlib/timer.h"
46 #include <stdio.h>
47
48 /*****************************************************************************
49 *
50 * Own Includings
51 *
52 *****************************************************************************/
53 #include "utils/uartstdio.h"
54 #include "header/mysdcard.h"
55 #include "header/config.h"
56 #include "header/clocktimer.h"
57 #include "header/myadc.h"
58 #include "header/temperature.h"
59 #include "header/ethernet.h"
60 #include "header/control.h"
61 #include "header/relais.h"
62
```

clocktimer.c

```
63
64 // Global variables for systemclock data
65 volatile unsigned long clock_msec = 00;
66 volatile unsigned long clock_sec = 00;
67 volatile unsigned long clock_min = 00;
68 volatile unsigned long clock_hour = 18;
69 volatile unsigned long clock_day = 06;
70 volatile unsigned long clock_month = 03;
71 volatile unsigned long clock_year = 2013;
72
73 // Global variables for loadrelay timing control
74 volatile unsigned long timer2minute = 0;
75 volatile unsigned int power_relay_active = 0;
76 volatile unsigned int cycle_active = false;
77
78 // TCP-Socket für die aktuelle Steuerungs-Verbindung
79 extern struct tcp_pcb* control_connection;
80
81
82 //*****************************************************************************
83 //Extern variable for name of file for measurment saving
84 //*****************************************************************************
85 extern char MEAS_FILE[128];
86
87 //
88 // Set up and enable the timers.  Configure a timer
89 // for the measure interrupt handler.
90 // Additionally set up Timer for display refreshment at 15Hz.
91 // and minute counter.
92 //
93 void init_clock_timer(){
94
95     UARTprintf("Initializing Timer...");
96
97         //
98         // Enable the peripherals used by this example.
99         //
100        //Timer for Measurment-Interrupt
101        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
102        //Timer for LED-Display refreshment
103        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
104        //Timer for loadrelay timing control
105        SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
106
107        //
108        // Configure the 32-bit periodic timers.
109        //
110        //Configure Timer1A for defined time "Cyclestep" from config
111        TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
112        TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() * config.cyclestep);
113        //Configure Timer1A at 15 Hz
114        TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC);
115        TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()/15);
116        //configure Timer2A interrupt for every minute
117        TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC);
118        TimerLoadSet(TIMER2_BASE, TIMER_A, SysCtlClockGet()*60);
119
120        //
121        // Enables Trigger for internal ADCs
122        //
123        TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
124
```

clocktimer.c

```
125         //
126         // Setup the interrupt for the timer timeout.
127         //
128         IntEnable(INT_TIMER0A);
129         TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
130
131         IntEnable(INT_TIMER1A);
132         TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
133
134         IntEnable(INT_TIMER2A);
135         TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
136
137
138     UARTprintf("done\n");
139 }
140
141 //*****************************************************************************
142 // Function to reinitialize the Timer0A when variable "Cyclestep" reconfigured
143 //*****************************************************************************
144 void reinit_timer0(void){
145     TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() * config.cyclestep);
146 }
147
148
149 //*****************************************************************************
150 //
151 // The interrupt handler for the periodical measurement. Called depended on
152 // configured "Cyclestep".
153 //
154 //*****************************************************************************
155 void Timer0IntHandler(void)
156 {
157
158     //
159     // Disable Timer 1
160     //
161     TimerDisable(TIMER1_BASE, TIMER_A);
162     //
163     // Clear the timer interrupt.
164     //
165     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
166     // Do the measurement
167     do_measure();
168
169     //
170     // Enable Timer 1
171     //
172     TimerEnable(TIMER1_BASE, TIMER_A);
173 }
174
175 //*****************************************************************************
176 //
177 // The interrupt handler for switching power relays end automatic stop of measurement.
178 // Called every minute.
179 //
180 //*****************************************************************************
181 void Timer2IntHandler(void)
182 {
183     int i;
184     //
185     // Disable Timer 1
186     //
```

clocktimer.c

```
187     TimerDisable(TIMER1_BASE, TIMER_A);
188     timer2minute++;
189
190     //Select powerrelay depended on configured switchtime
191     for(i = 1; i <= config.relay_quantity; i++)
192     if(timer2minute == config.relay_time[i]){
193         power_relay_active = switch_power_relais(config.relay_select[i]);
194     }
195
196     //Stop measurment if measurtime reached
197     if(timer2minute == config.measuretime){
198         TimerDisable(TIMER0_BASE, TIMER_A);
199         TimerDisable(TIMER2_BASE, TIMER_A);
200
201         UARTprintf("\nMeasurement cycling stoped");
202         if(control_connection){
203             telnet_write("Measurement cycling stoped\n");
204         }
205         // Deactivate Cycling mode, deselect powerrelays
206         cycle_active = false;
207         timer2minute = 0;
208         power_relay_active = switch_power_relais(0);
209     }
210
211     //
212     // Clear the timer interrupt.
213     //
214     TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
215
216     //
217     // Enable Timer 1
218     //
219     TimerEnable(TIMER1_BASE, TIMER_A);
220
221 }
222
223 //*****************************************************************************
224 // Update the Systemclock
225 //*****************************************************************************
226 void TimerClock(void)
227 {
228         clock_msec += 10;
229         if(clock_msec == 1000){
230             clock_msec = 0;
231             clock_sec++;
232         }
233
234         if (clock_sec == 60){
235             clock_sec = 0;
236             clock_min++;
237         }
238         if(clock_min == 60){
239             clock_hour++;
240             clock_min = 0;
241         }
242         if(clock_hour == 24){
243             clock_day++;
244             clock_hour = 0;
245         }
246
247         if((clock_month == 1) || (clock_month == 3) || (clock_month == 5) || (clock_month
    == 7) || (clock_month == 8) || (clock_month == 10) || (clock_month == 12)){
```

clocktimer.c

```
248             if(clock_day == 32) {
249                 clock_month++;
250                 clock_day = 1;
251             }
252         }
253         if((clock_month == 4) || (clock_month == 6) || (clock_month == 9) || (clock_month
    == 11)){
254             if(clock_day == 31) {
255                 clock_month++;
256                 clock_day = 1;
257             }
258         }
259 
260         if(clock_month == 2){
261             int d_temp = 29;
262             if(clock_year%4 == 0)
263                 d_temp = 30;
264             if(clock_year%4 != 0)
265                 d_temp = 29;
266             if(clock_day == d_temp) {
267                 clock_month++;
268                 clock_day = 1;
269             }
270         }
271 
272         if(clock_month == 13){
273             clock_year++;
274             clock_month = 1;
275         }
276 }
277 
278 //*****************************************************************************
279 //
280 // This function implements the "start" command. It starts the measurement
281 // by activating Timer 0A.
282 //
283 //*****************************************************************************
284 int
285 Cmd_start(int argc, char *argv[])
286 {
287     int i = 0;
288 
289     //Generate Filename
290     sprintf(MEAS_FILE, "%02d%02d%02d%02d.txt", clock_month, clock_day, clock_hour,
    clock_min);
291 
292     //Generate header info
293     char header[64];
294     add_to_file(MEAS_FILE, "****************************************");
295 
296     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);
297     add_to_file(MEAS_FILE, header);
298 
299     sprintf(header, "\nQuantity Cells: %02d\nCyclestep: %02d sec", config.quantity_cells,
    config.cyclestep);
300     add_to_file(MEAS_FILE, header);
301 
302     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);
303     add_to_file(MEAS_FILE, header);
304 
```

clocktimer.c

```
305     sprintf(header, "\nCELL   | ADC Gain   |  ADC Offset");
306     add_to_file(MEAS_FILE, header);
307
308     for(i = 1; i <= 12; i++){
309         sprintf(header, "\n%02d:     %08d       %06d", i, config.gain[i],
    config.offset[i]);
310         add_to_file(MEAS_FILE, header);
311     }
312
313     sprintf(header, "\nRelayquantity:      %02d", config.relay_quantity);
314     add_to_file(MEAS_FILE, header);
315
316     sprintf(header, "\nRelaytime(min):     Relayselect:");
317     add_to_file(MEAS_FILE, header);
318
319     for(i = 1; i <= 100; i++){
320         sprintf(header, "\n%05d               %02d", config.relay_time[i],
    config.relay_select[i]);
321         add_to_file(MEAS_FILE, header);
322     }
323
324     add_to_file(MEAS_FILE, "\n*****************************************");
325     add_to_file(MEAS_FILE, "\nCE,YYYY,MM,DD,HH,MM,SS,VOL(uV),TEM,CUR(mA),R");
326     add_to_file(MEAS_FILE, "\n*****************************************\n");
327
328     //Enable Timer0A and Cycling Mode
329     TimerEnable(TIMER0_BASE, TIMER_A);
330     cycle_active = true;
331     UARTprintf("\nMeasurement cycling started");
332     if(control_connection){
333         telnet_write("Measurement cycling started\n");
334     }
335
336     //Enable Timer2A
337     TimerEnable(TIMER2_BASE, TIMER_A);
338     //
339     // Return success.
340     //
341     return(0);
342 }
343
344 //*****************************************************************************
345 //
346 // This function implements the "stop" command. It stops the measurement
347 // by deactivating Timer 0A and Timer 2A.
348 //
349 //*****************************************************************************
350 int
351 Cmd_stop(int argc, char *argv[])
352 {
353     //Disable Timer0A and Timer2A
354     TimerDisable(TIMER0_BASE, TIMER_A);
355     TimerDisable(TIMER2_BASE, TIMER_A);
356     UARTprintf("\nMeasurement cycling stoped");
357     if(control_connection){
358         telnet_write("Measurement cycling stoped\n");
359     }
360
361     //Disable Cycling mode and deselect Loadrelays
362     cycle_active = false;
363     timer2minute = 0;
364     power_relay_active = switch_power_relais(0);
```

clocktimer.c

```
365     //
366     // Return success.
367     //
368     return(0);
369 }
370
371
372
373
```

config.h

```
 1 /*
 2  * config.h
 3  *
 4  *  Created on: 06.03.2013
 5  *      Author: Thomas W.
 6  */
 7
 8
 9 #ifndef CONFIG_H_
10 #define CONFIG_H_
11
12 #include "config.h"
13
14 // sizeof(config_t) must be an even number! Otherwise the program
15 // will fail to save and/or reload the configuration!
16 typedef struct
17 {
18     long quantity_cells;
19     long cyclestep;
20     int offset[13];
21     int gain[13];
22     int ip_a;
23     int ip_b;
24     int ip_c;
25     int ip_d;
26     int relay_quantity;
27     int relay_time[101];
28     int relay_select[101];
29     int zerocurrentch1;
30     int zerocurrentch2;
31     int measuretime;
32
33 } config_t;
34
35 extern config_t config;
36
37 /*****************************************************************************
38 *
39 * Function Declarations
40 *
41 *****************************************************************************/
42 void config_read(void);
43 int config_write(void);
44 void init_config(void);
45
46 int config_cmd_line(int argc, char *argv[]);
47 int Cmd_config_exit(int argc, char *argv[]);
48 int Cmd_gain(int argc, char *argv[]);
49 int Cmd_offset(int argc, char *argv[]);
50 int Cmd_calibrate(int argc, char *argv[]);
51 int Cmd_print(int argc, char *argv[]);
52 int Cmd_quantity(int argc, char *argv[]);
53 int Cmd_cyclestep(int argc, char *argv[]);
54 int Cmd_relsel(int argc, char *argv[]);
55 int Cmd_relqua(int argc, char *argv[]);
56 int Cmd_meastime(int argc, char *argv[]);
57 int Cmd_save(int argc, char *argv[]);
58 int Cmd_ip(int argc, char *argv[]);
59
60 // Für die interne Nutzung
61 void config_erase(void);
62 void config_print_uart(void);
```

config.h

```
63 void config_write_struct(const char *conf_name, const char *conf_val);
64
65 void calibrate_adc(void);
66
67 #endif /* CONFIG_H_ */
68
69
```

config.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           config.c

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

last modified:                  2013/06/04

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code for configure operations

*/
/*****************************************************************************
*
* Includings
*
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "math.h"
#include "third_party/fatfs/src/ff.h"
#include "third_party/fatfs/src/diskio.h"
#include <utils/uartstdio.h>
#include <driverlib/ethernet.h>

/*****************************************************************************
*
* Own Includings
*
*****************************************************************************/
#include "header/myadc.h"
#include "header/config.h"
#include "header/mysdcard.h"
#include "header/mycmdline.h"
#include "header/ethernet.h"
#include "header/control.h"
#include "header/relais.h"
#include "header/myadc.h"
#include "header/clocktimer.h"


// Define Name of Configurationfile
const char* CONFIG_FILE = "config.txt";


//*****************************************************************************
```

config.c

```
63 //
64 // Defines the size of the buffer that holds the input data.
65 //
66 //*****************************************************************************
67 #define CONF_BUF_SIZE 2048
68
69 //*****************************************************************************
70 //
71 // Defines the size of the buffer that holds the input data.
72 //
73 //*****************************************************************************
74 #define CONFIG_INPUT_DATA_SIZE    64
75
76 //*****************************************************************************
77 //
78 // A temporary data buffer used for input data
79 //
80 //*****************************************************************************
81 static char g_cTmpInputData[CONFIG_INPUT_DATA_SIZE];
82
83 //*****************************************************************************
84 //
85 // This is the table that holds the command names, implementing functions,
86 // and brief description.
87 //
88 //*****************************************************************************
89 tCmdLineEntry g_sConfigCmdTable[] =
90 {
91     { "help",       Cmd_help,     "     : Display list of commands" },
92     { "h",          Cmd_help,  "        : alias for help" },
93     { "?",          Cmd_help,  "        : alias for help" },
94     { "print",      Cmd_print,     "    : Print current Configuration" },
95     { "quantity",   Cmd_quantity,     " : Set quantity of cells (01 - 12)" },
96     { "cyclestep",  Cmd_cyclestep,     ": Set measurement period in sec (0000 - 9999)" },
97     { "gain",       Cmd_gain,     "     : Set Gain (eg. 'gain 01 76273500')" },
98     { "offset",     Cmd_offset,     "   : Set Offset (eg. 'offset 01 -00100')" },
99     { "calibrate",  Cmd_calibrate,     ": Calibrate current sensor to zero" },
100    { "ip",         Cmd_ip,     "       : Set IP address (eg. 'ip 192.168.000.128')" },
101    { "relais",     Cmd_relais,     "   : Manualy activates choosen powerrelais (e.g.
   'relais 1')"},
102    { "relsel",     Cmd_relsel,     "   : Set Relayselect (eg. 'relsel 00 00001 1')"},
103    { "relqua",     Cmd_relqua,     "   : Set quantity of Relay switches (01 - 99)"},
104    { "time",       Cmd_meastime, "     : Set time of full cycling procedure in min (eg.
   'time 1440')"},
105    { "save",       Cmd_save,     "     : Save Configuration to SD-Card" },
106    { "exit",       Cmd_config_exit,"     : Exit configuration operations" },
107    { 0, 0, 0 }
108 };
109
110 extern tCmdLineEntry g_sMainCmdTable;
111 extern tCmdLineEntry *g_psCmdTable;
112
113 // string for main location
114 extern const char *g_cMainLocalBuf;
115
116 // Global location buffer
117 extern char *g_cLocalBuf;
118
119 //define string for config location
120 const char *g_cConfigLocalBuf = "Config";
121
122 // String buffer for print operations to UART and Ethernet
```

config.c

```
123 extern char print_buffer[1024];
124
125 // TCP-Socket für die aktuelle Steuerungs-Verbindung
126 extern struct tcp_pcb* control_connection;
127
128 /*
129  * Initialisieren der Konfiguration.
130  */
131 void init_config(void){
132
133     UARTprintf("Initializing Configuration:\n");
134     config_read();
135     UARTprintf("Initializing Configuration...done\n");
136
137 }
138
139
140
141 /*
142  * Löschen der Konfiguration
143  */
144 void config_erase(void)
145 {
146     delete_file(CONFIG_FILE);
147     create_file(CONFIG_FILE);
148 }
149
150 /*
151  * Auslesen der Konfiguration.
152  */
153 void config_read(void)
154 {
155     char buffer[ CONF_BUF_SIZE ];
156
157     //If Error reading CONFIG_GILE, load default Values
158     if( read_into_buffer( CONFIG_FILE, buffer ) ) {
159
160         UARTprintf("\nERROR opening %s", CONFIG_FILE);
161         UARTprintf("\nLoading default Values\n\n");
162
163         int i;
164
165         config.quantity_cells = 00;
166         config.cyclestep = 00;
167
168         config.gain[0]  = 0;
169         config.gain[1]  = 76308362;
170         config.gain[2]  = 76307211;
171         config.gain[3]  = 76305996;
172         config.gain[4]  = 76312575;
173         config.gain[5]  = 76310449;
174         config.gain[6]  = 76305895;
175         config.gain[7]  = 76306869;
176         config.gain[8]  = 76307375;
177         config.gain[9]  = 76305148;
178         config.gain[10] = 76307375;
179         config.gain[11] = 76310840;
180         config.gain[12] = 76308108;
181
182         config.offset[0]  = 0;
183         config.offset[1]  = 96;
184         config.offset[2]  = 125;
```

config.c

```
185         config.offset[3]  = 129;
186         config.offset[4]  = 58;
187         config.offset[5]  = 116;
188         config.offset[6]  = 153;
189         config.offset[7]  = 112;
190         config.offset[8]  = 125;
191         config.offset[9]  = 176;
192         config.offset[10] = 114;
193         config.offset[11] = 181;
194         config.offset[12] = 321;
195 
196 
197         config.ip_a = 192;
198         config.ip_b = 168;
199         config.ip_c = 000;
200         config.ip_d = 128;
201         config.relay_quantity = 0;
202 
203         for (i=0; i <= 50; i++){
204             config.relay_time[i] = 0;
205             config.relay_select[i] = 0;
206         }
207 
208         config.measuretime = 0;
209 
210         config_print_uart();
211 
212         return;
213     }
214 
215             //Read config data from SD-Card
216             int i, j;
217             config.quantity_cells = 10 * (buffer[17] - '0') + (buffer[18] - '0');
218             config.cyclestep = 1000* (buffer[37] - '0') +  100 * (buffer[38] - '0') + 10 *
    (buffer[39] - '0') + (buffer[40] - '0');
219 
220             i = 0;
221             j = 0;
222 
223             for(i = 0; i <= config.quantity_cells-1; i++){
224                 config.gain[i+1] = 0;
225                 config.offset[i+1] = 0;
226                 for(j = 0; j <= 7; j++)
227                     config.gain[i+1] += pow(10,7-j) * (buffer[60+j+(i*9)] - '0');
228 
229                 if((buffer[186+(i*7)]) == '-' ){
230                 for(j = 1; j <= 5; j++)
231                     config.offset[i+1] -= pow(10, 5-j) * (buffer[186+j+(i*7)] - '0');
232                 }
233                 if((buffer[186+(i*7)]) != '-' ){
234                 for(j = 0; j <= 5; j++)
235                     config.offset[i+1] += pow(10, 5-j) * (buffer[186+j+(i*7)] - '0');
236                 }
237             }
238 
239             config.ip_a = 100 * (buffer[289] - '0') + 10  * (buffer[290] - '0') +
    (buffer[291] - '0');
240             config.ip_b = 100 * (buffer[293] - '0') + 10  * (buffer[294] - '0') +
    (buffer[295] - '0');
241             config.ip_c = 100 * (buffer[297] - '0') + 10  * (buffer[298] - '0') +
    (buffer[299] - '0');
242             config.ip_d = 100 * (buffer[301] - '0') + 10  * (buffer[302] - '0') +
```

config.c

```
   (buffer[303] - '0');
243
244             config.measuretime = 10000* (buffer[327] - '0') + 1000* (buffer[328] - '0') +
   100 * (buffer[329] - '0') + 10 * (buffer[330] - '0') + (buffer[331] - '0');
245
246             config.relay_quantity = 10 * (buffer[355] - '0') + (buffer[356] - '0');
247
248             for(i = 1; i <= config.relay_quantity; i++){
249                 config.relay_time[i] = 10000* (buffer[392 + 24 * (i-1)] - '0') + 1000*
   (buffer[393 + 24 * (i-1)] - '0') +  100 * (buffer[394 + 24 * (i-1)] - '0') + 10 *
   (buffer[395 + 24 * (i-1)] - '0') + (buffer[396 + 24 * (i-1)] - '0');
250                 config.relay_select[i] =  10* (buffer[413 + 24 * (i-1)] - '0') +
   (buffer[414 + 24 * (i-1)] - '0');
251             }
252
253             config_print_uart();
254
255             return;
256 }
257
258
259 /*
260  * Schreiben der aktuellen Konfiguration in die config.txt (SD-Karte).
261  */
262 int config_write(void)
263 {
264     int i = 0;
265
266     create_file(CONFIG_FILE);
267
268     char write_quantity_cells[3];
269     sprintf(write_quantity_cells, "%02i", config.quantity_cells);
270     config_write_struct("Quantity Cells:  ", write_quantity_cells );
271
272     char write_cyclestep[5];
273     sprintf(write_cyclestep, "%04i", config.cyclestep);
274     config_write_struct("Cyclestep (sec): ", write_cyclestep );
275
276     char write_adc_gain[32];
277     add_to_file(CONFIG_FILE, "\nADC Gain:        ");
278     for(i = 1; i <= 12; i++){
279         sprintf(write_adc_gain, "%08d ", config.gain[i]);
280         add_to_file(CONFIG_FILE, write_adc_gain);
281     }
282
283     char write_adc_offset[32];
284     add_to_file(CONFIG_FILE, "\nADC Offset:      ");
285     for(i = 1; i <= 12; i++){
286         sprintf(write_adc_offset, "%06d ", config.offset[i]);
287         add_to_file(CONFIG_FILE, write_adc_offset);
288     }
289
290     char write_ip_address[48];
291     sprintf(write_ip_address, "\n\nIP Address:      %03d.%03d.%03d.%03d", config.ip_a,
   config.ip_b, config.ip_c, config.ip_d);
292     add_to_file(CONFIG_FILE, write_ip_address);
293
294
295     char write_measure_time[32];
296     sprintf(write_measure_time,  "\n\nMeasuretime (min):   %05d", config.measuretime);
297     add_to_file(CONFIG_FILE, write_measure_time );
298
```

config.c

```
299
300     char write_relay_control[32];
301     sprintf(write_relay_control, "\n\nRelay Quantity:      %02d", config.relay_quantity);
302     add_to_file(CONFIG_FILE, write_relay_control );
303
304
305     add_to_file(CONFIG_FILE,  "\n\nRelaytime (min)      Relayselect");
306     for(i = 1; i <= 51; i++){
307         sprintf(write_relay_control, "\n%05d                %02d", config.relay_time[i],
    config.relay_select[i]);
308         add_to_file(CONFIG_FILE, write_relay_control);
309     }
310
311     return 1;
312 }
313
314 /*
315  * Hilfsfunktion zum strukturieren der Konfigurationsdatei
316  */
317 void config_write_struct(const char *conf_name, const char *conf_val){
318
319     add_to_file(CONFIG_FILE, conf_name);
320     add_to_file(CONFIG_FILE, conf_val);
321     add_to_file(CONFIG_FILE, "\n");
322
323 }
324
325 /*
326  * Ausgabe der Konfiguration Ueber UART und Ethernet.
327  * Gibt tabellarisch alle Werte der aktuellen Konfiguration über
328  * die UART und ggbf. vorhandene Ethernet Verbindung aus.
329  */
330 void config_print_uart(void)
331 {
332     int i = 0;
333
334     UARTprintf("\n  Quantity_cells: %d Cells", config.quantity_cells);
335     UARTprintf("\n  Cyclestep:      %d sec", config.cyclestep);
336     UARTprintf("\n  IP Address:     %03d.%03d.%03d.%03d", config.ip_a, config.ip_b,
    config.ip_c, config.ip_d);
337     UARTprintf("\n\n    Cell |  ADC Gain | ADC Offset\n");
338
339     for(i = 1; i <= config.quantity_cells; i++){
340     UARTprintf("\n     %02d:    %08d     %06d", i, config.gain[i], config.offset[i]);
341     }
342
343     UARTprintf("\n\nMeasuretime (min):   %05d", config.measuretime);
344
345     UARTprintf("\n\nRelayquantity:       %01d", config.relay_quantity);
346
347     UARTprintf("\n\nRelaytime (min)      Relayselect");
348
349     for(i = 1; i <= config.relay_quantity; i++){
350         UARTprintf("\n%05d                %02d", config.relay_time[i],
    config.relay_select[i]);
351     }
352
353     UARTprintf("\n\n");
354
355
356
357     if(control_connection){
```

config.c

```
358
359         telnet_write("\n");
360         sprintf(print_buffer, "  Quantity_cells: %d Cells\n", config.quantity_cells);
361         telnet_write(print_buffer);
362         sprintf(print_buffer, "  Cyclestep:      %d sec\n", config.cyclestep);
363         telnet_write(print_buffer);
364         sprintf(print_buffer, "  IP Address:     %03d.%03d.%03d.%03d\n", config.ip_a,
    config.ip_b, config.ip_c, config.ip_d);
365         telnet_write(print_buffer);
366         telnet_write("\n");
367         sprintf(print_buffer, "    Cell |  ADC Gain | ADC Offset\n");
368         telnet_write(print_buffer);
369         telnet_write("\n");
370         telnet_write("\n");
371
372         sprintf(print_buffer, "");
373         char buf[64];
374         for(i = 1; i <= config.quantity_cells; i++){
375             sprintf(buf, "     %02d:    %08d     %06d\n", i, config.gain[i],
    config.offset[i]);
376             strcat(print_buffer, buf);
377         }
378
379         telnet_write(print_buffer);
380
381         telnet_write("\n");
382         telnet_write("\n");
383
384         sprintf(print_buffer, "  Measuretime (min): %05d\n", config.measuretime);
385         telnet_write(print_buffer);
386
387         telnet_write("\n");
388         telnet_write("\n");
389
390         sprintf(print_buffer, "  Relayquantity  : %01d\n", config.relay_quantity);
391         telnet_write(print_buffer);
392
393         telnet_write("\n");
394
395         sprintf(print_buffer, "Relaytime (min):      Relayselect:\n");
396         telnet_write(print_buffer);
397
398         for(i = 1; i <= config.relay_quantity; i++){
399         sprintf(print_buffer, "%05d                %02d\n", config.relay_time[i],
    config.relay_select[i]);
400         telnet_write(print_buffer);
401         }
402
403         telnet_write("\n");
404         telnet_write("\n");
405
406     }
407
408
409
410
411 }
412
413 //***************************************************************************
414 //
415 // Implementation of an command line for configure operations based on stellarisware
    sd-card example
```

config.c

```
416 //
417 //*****************************************************************************
418 int config_cmd_line(int argc, char *argv[])
419 {
420
421     // set Command line pointer to the beginning of the SD-Card/Browser command structure
422     g_psCmdTable = &g_sConfigCmdTable[0];
423
424     // set location buffer to Browser
425     g_cLocalBuf = (char*)g_cConfigLocalBuf;
426
427     return(0);
428 }
429
430 //*****************************************************************************
431 //
432 // Implementation of an command to exit to main command line entry
433 //
434 //*****************************************************************************
435 int Cmd_config_exit(int argc, char *argv[])
436 {
437     // set Command line pointer to the beginning of the main command structure
438     g_psCmdTable = &g_sMainCmdTable;
439
440     // set location buffer to main
441     g_cLocalBuf = (char*)g_cMainLocalBuf;
442
443     return(0);
444 }
445
446 //*****************************************************************************
447 //
448 // Implementation of an command to set the gain value
449 //
450 //*****************************************************************************
451 int Cmd_gain(int argc, char *argv[])
452 {
453     int j, cell;
454     //
455     // Copy the first input data into buffer.
456     //
457     strcpy(g_cTmpInputData, argv[1]);
458
459     cell = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
460
461     //
462     // Copy the second input data into buffer.
463     //
464     strcpy(g_cTmpInputData, argv[2]);
465     config.gain[cell] = 0;
466
467
468     for(j = 0; j <= 7; j++)
469         config.gain[cell] += pow(10,7-j) * (g_cTmpInputData[j] - '0');
470
471     return(0);
472 }
473
474 //*****************************************************************************
475 //
476 // Implementation of an command to set the offset value
477 //
```

config.c

```
478 //*****************************************************************************
479 int Cmd_offset(int argc, char *argv[])
480 {
481     int j, cell;
482 
483     //
484     // Copy the first input data into buffer.
485     //
486     strcpy(g_cTmpInputData, argv[1]);
487 
488     //
489     // Assign cell
490     //
491     cell = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
492 
493     //
494     // Copy the second input data into buffer.
495     //
496     strcpy(g_cTmpInputData, argv[2]);
497     config.offset[cell] = 0;
498 
499     //
500     // Configure negative offset to assigned cell
501     //
502     if(g_cTmpInputData[0] == '-' ){
503         for(j = 1; j <= 5; j++)
504         config.offset[cell] -= pow(10, 5-j) * (g_cTmpInputData[j] - '0');
505     }
506 
507     //
508     // Configure positive offset to assigned cell
509     //
510     if(g_cTmpInputData[0] != '-' ){
511         for(j = 0; j <= 5; j++)
512         config.offset[cell] += pow(10, 5-j) * (g_cTmpInputData[j] - '0');
513     }
514 
515     return(0);
516 
517 }
518 
519 //*****************************************************************************
520 //
521 // Implementation of an command to autocalibrate hall sensor offset
522 //
523 //*****************************************************************************
524 int Cmd_calibrate(int argc, char *argv[])
525 {
526 
527     //
528     // Calibrate and set offset for current
529     //
530     config.zerocurrentch1 = 0;
531     config.zerocurrentch2 = 0;
532 
533     config.zerocurrentch1 = (32768 - adc_get_single_value_ch2());
534     config.zerocurrentch2 = (32768 - adc_get_single_value_ch3());
535 
536 
537     return(0);
538 
539 }
```

config.c

```
540
541 //*****************************************************************************
542 //
543 // Implementation of an command to set the up-address
544 //
545 //*****************************************************************************
546 int Cmd_ip(int argc, char *argv[])
547 {
548     //
549     // Copy the first input data into buffer.
550     //
551     strcpy(g_cTmpInputData, argv[1]);
552
553     config.ip_a = 100 * (g_cTmpInputData[0] - '0') + 10  * (g_cTmpInputData[1] - '0') +
    (g_cTmpInputData[2] - '0');
554     config.ip_b = 100 * (g_cTmpInputData[4] - '0') + 10  * (g_cTmpInputData[5] - '0') +
    (g_cTmpInputData[6] - '0');
555     config.ip_c = 100 * (g_cTmpInputData[8] - '0') + 10  * (g_cTmpInputData[9] - '0') +
    (g_cTmpInputData[10] - '0');
556     config.ip_d = 100 * (g_cTmpInputData[12] - '0') + 10  * (g_cTmpInputData[13] - '0') +
    (g_cTmpInputData[14] - '0');
557     //@TODO Dies ist keine optimale Lösung hier - neu initialiesierung möglich?
558     UARTprintf("\n\nPlease save config and restart Battery Cycle Machine!");
559     telnet_write("Please save config and restart Battery Cycle Machine!\n");
560
561     return(0);
562 }
563
564 //*****************************************************************************
565 //
566 // Implementation of an command to print actual configuration to UART and Ethernet
567 //
568 //*****************************************************************************
569 int Cmd_print(int argc, char *argv[])
570 {
571     config_print_uart();
572     return(0);
573 }
574
575 //*****************************************************************************
576 //
577 // Implementation of an command to set the quantity of battery cells
578 //
579 //*****************************************************************************
580 int Cmd_quantity(int argc, char *argv[])
581 {
582
583     //
584     // Copy the input data into buffer.
585     //
586     strcpy(g_cTmpInputData, argv[1]);
587
588     config.quantity_cells = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
589     return(0);
590 }
591
592 //*****************************************************************************
593 //
594 // Implementation of an command to set the time between measurment ("Cyclestep")
595 //
596 //*****************************************************************************
597 int Cmd_cyclestep(int argc, char *argv[])
```

config.c

```
598 {
599     //
600     // Copy the input data into buffer.
601     //
602     strcpy(g_cTmpInputData, argv[1]);
603 
604     config.cyclestep = 1000 * (g_cTmpInputData[0] - '0') + 100 * (g_cTmpInputData[1] - '0')
    + 10 * (g_cTmpInputData[2] - '0') + (g_cTmpInputData[3] - '0') ;
605 
606     reinit_timer0();
607 
608     return(0);
609 }
610 
611 //*****************************************************************************
612 //
613 // Implementation of an command to set the measurement time
614 //
615 //*****************************************************************************
616 int Cmd_meastime(int argc, char *argv[])
617 {
618     //
619     // Copy the input data into buffer.
620     //
621     strcpy(g_cTmpInputData, argv[1]);
622 
623     config.measuretime = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
624     return(0);
625 
626 }
627 
628 //*****************************************************************************
629 //
630 // Implementation of an command to set the quantity loadrelay switches
631 //
632 //*****************************************************************************
633 int Cmd_relqua(int argc, char *argv[])
634 {
635     //
636     // Copy the input data into buffer.
637     //
638     strcpy(g_cTmpInputData, argv[1]);
639 
640     config.relay_quantity = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
641     return(0);
642 
643 }
644 
645 //*****************************************************************************
646 //
647 // Implementation of an command to set the selection of a loadrelay by given time
648 //
649 //*****************************************************************************
650 int Cmd_relsel(int argc, char *argv[])
651 {
652     int j, relayid;
653 
654     //
655     // Copy the first input data into buffer.
656     //
657     strcpy(g_cTmpInputData, argv[1]);
658 
```

config.c

```
659     //
660     // Assign relayid
661     //
662     relayid = 10 * (g_cTmpInputData[0] - '0') + (g_cTmpInputData[1] - '0') ;
663
664     //
665     // Copy the second input data into buffer.
666     //
667     strcpy(g_cTmpInputData, argv[2]);
668     config.relay_time[relayid] = 0;
669
670     //
671     // Configure time to assigned relayswitch
672     //
673     for(j = 0; j <= 4; j++)
674     config.relay_time[relayid] += pow(10, 4-j) * (g_cTmpInputData[j] - '0');
675
676     //
677     // Copy the third input data into buffer.
678     //
679     strcpy(g_cTmpInputData, argv[3]);
680
681     //
682     // Configure time to assigned relayswitch
683     //
684     config.relay_select[relayid] = (g_cTmpInputData[0] - '0');
685
686
687     return(0);
688
689 }
690
691
692 //*****************************************************************************
693 //
694 // Implementation of an command to save current configuration to SD-Card
695 //
696 //*****************************************************************************
697 int Cmd_save(int argc, char *argv[])
698 {
699     config_write();
700     return(0);
701 }
702
703
704
705 //*****************************************************************************
706 // Function used to calibrate channelwise 16-bit adc. Use debuger and
707 // breakpoints. Values of ADC are written to adc.txt. Use two Voltages.
708 // One low (ca. 300 mV) and one high (ca. 3.2V). Experience made with FLUKE 45
709 //*****************************************************************************
710 void calibrate_adc(void){
711
712     int i = 0;
713     int j = 0;
714     char buf[32];
715
716     delete_file("adc.txt");
717     create_file("adc.txt");
718
719
720     for(j = 1; j <= 12; j++){
```

config.c

```
721
722
723         switch_relais(j);
724         sprintf(buf,"\n CELL LOW %d\n", j);
725         add_to_file("adc.txt", buf);
726
727         for(i=1 ; i<=20 ; i++){
728
729             sprintf(buf,"%d\n", adc_get_single_value());
730             add_to_file("adc.txt", buf);
731         }
732         UARTprintf("\n%d LOW done",j);
733         UARTprintf("\nnext");
734
735     }
736     UARTprintf("\nALL LOW DONE!");
737
738     for(j = 1; j <= 12; j++){
739
740
741             switch_relais(j);
742             sprintf(buf,"\n CELL HIGH %d\n", j);
743             add_to_file("adc.txt", buf);
744
745             for(i=1 ; i<=20 ; i++){
746
747                 sprintf(buf,"%d\n", adc_get_single_value());
748                 add_to_file("adc.txt", buf);
749             }
750             UARTprintf("\n%d HIGH done",j);
751             UARTprintf("\nnext");
752
753         }
754         UARTprintf("\nALL HIGH DONE!");
755
756 }
757
758
759
760
```

control.h

```
 1 /*
 2  * control.h
 3  *
 4  *  Created on: 25.12.2011
 5  *      Author: Fabian S.
 6  *      Credits: Thomas W.
 7  */
 8
 9 #ifndef CONTROL_H_
10 #define CONTROL_H_
11
12 #include "lwip/tcp.h"
13
14 /*****************************************************************************
15 *
16 * Function Declarations
17 *
18 *****************************************************************************/
19 void control_init(void);
20 err_t control_new_con(void* arg, struct tcp_pcb* newpcb, err_t err);
21 err_t control_rx(void* arg, struct tcp_pcb* tpcb, struct pbuf* p, err_t err);
22 void control_tick(unsigned char *data_check, unsigned long data_length);
23 void telnet_write(const void *data);
24
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
```

control.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           control.c

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

last modified:                  2013/03/14

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:                    Controlling TCP-link. Processing and
                                execution of commands.
*/


#include "header/control.h"
#include "inc/hw_types.h"
#include "utils/uartstdio.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "header/mycmdline.h"


struct tcp_pcb* control_tpcb;  // Server-Socket für die TCP-Verbindung
// Stream zur Speicherung aller erhaltenen Daten Ã¼ber den TCP-Socket

// TCP-Socket für die aktuelle Steuerungs-Verbindung
struct tcp_pcb* control_connection = (struct tcp_pcb*)NULL;

// Global location buffer
extern char *g_cLocalBuf;

//*****************************************************************************
//
// The buffer that holds the answer line.
//
//*****************************************************************************
char g_cNewConBuf[32];


/*
 * Initialisierung der Kommunikation.
 * Hier wird der Server-TCP-Socket und der UDP-Socket initialisiert.
 * Zudem wird auch der FIFO für die Befehle initialisiert.
 */
void control_init(void)
{
    UARTprintf("  Initializing control...");

```

control.c

```
63      control_tpcb = tcp_new();
64      tcp_bind(control_tpcb, IP_ADDR_ANY, 56936);
65      control_tpcb = tcp_listen(control_tpcb);
66      tcp_accept(control_tpcb, &control_new_con);
67      UARTprintf("done\n");
68  }
69
70  /*
71   * "Interrupt" bei Erhalt einer neuen TCP-Verbindung.
72   * Diese Funktion wird vom lwIP-Stack ausgeführt, wenn eine
73   * neue TCP-Verbindung angefordert wird. Zunachst wird eine evtl.
74   * bestehende alte Verbindung getrennt. Anschließend wird die
75   * neue Verbindung angenommen.
76   */
77  err_t control_new_con(void* arg, struct tcp_pcb* newpcb, err_t err)
78  {
79      if(control_connection)
80          tcp_close(control_connection);
81      control_connection = newpcb;
82      tcp_accepted(newpcb);
83      tcp_recv(newpcb, &control_rx);
84      UARTprintf("New control connection established!\n");
85      if(control_connection){
86      telnet_write(" Enter Command (type <help> to see list of commands):\n");
87      sprintf(g_cNewConBuf, "%s: > \n", g_cLocalBuf);
88      telnet_write(g_cNewConBuf);
89      }
90      //
91      // Print a prompt to the console.
92      //
93      UARTprintf("\n Enter Command (type <help> to see list of commands):");
94      UARTprintf("\n%s: > ", g_cLocalBuf);
95      return ERR_OK;
96  }
97
98  /*
99   * "Interrupt" bei Erhalt von Daten Über den TCP-Socket.
100  * Diese Funktion wird vom lwIP-Stack ausgefÃ¼hrt, wenn neue
101  * Daten Ã¼ber den TCP-Socket eingetroffen sind.
102  * Die Daten werden hier zunÃ¤chst im FIFO zwischengespeichert.
103  * AnschlieÃŸend wird die Funktion control_tick() ausgefÃ¼hrt, um
104  * die erhaltenen Daten auszuwerten.
105  */
106 err_t control_rx(void* arg, struct tcp_pcb* tpcb, struct pbuf* p, err_t err)
107 {
108
109         if(p == NULL)
110     {
111         UARTprintf("Control connection closed!\n");
112         control_connection = NULL;
113         return ERR_OK;
114     }
115
116     struct pbuf* current_p=p;
117
118
119     unsigned char* receive_data = (unsigned char*) current_p->payload;
120     unsigned long received_data_length = current_p->tot_len;
121
122
123     control_tick(receive_data, received_data_length);
124     tcp_recved(tpcb, p->tot_len);
```

control.c

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

discover.h

```
 1 /*
 2  * discover.h
 3  *
 4  *  Created on: 16.07.2012
 5  *      Author: Thomas W. / Fabian S. / Tobias S.
 6  */
 7
 8 #ifndef DISCOVER_H_
 9 #define DISCOVER_H_
10
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
```

discover.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           discover.c

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

last modified:                  2013/03/07

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

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



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


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

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

/*
 * Initialisierung des UDP-Sockets.
 */
void discover_init(void)
{
    volatile unsigned long ulLoop;
    UARTprintf("  Initializing discover...");
    g_upcb = udp_new();
    err_t error = udp_bind(g_upcb, IP_ADDR_ANY, 56936);
    if(error != ERR_OK)
    {
```

discover.c

```
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
```

display.h

```
 1 /*
 2  * display.h
 3  *
 4  *  Created on: 25.03.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef DISPLAY_H_
 9 #define DISPLAY_H_
10
11 #define LEDPORTCTRL      GPIO_PORTH_BASE
12 #define LEDPORTDATA      GPIO_PORTJ_BASE
13
14 #define RS               GPIO_PIN_0
15 #define RW               GPIO_PIN_1
16 #define E                GPIO_PIN_2
17
18
19 #define D4               GPIO_PIN_0
20 #define D5               GPIO_PIN_1
21 #define D6               GPIO_PIN_2
22 #define D7               GPIO_PIN_3
23
24
25 // Displaymodes
26 typedef enum
27 {
28     START,
29     CLOCK,
30     MEAS,
31     MEASSTOP,
32     CELLA,
33     CELLB,
34     CELLC,
35     CELLD,
36     CURRENT,
37     IP,
38     CONF
39 } display_handler_t;
40
41 /*******************************************************************************
42 *
43 * Function Declarations
44 *
45 *******************************************************************************/
46
47 void display_init(void);
48 void buttons_init(void);
49 void clear_display(void);
50 void write_to_display(char* inputText, unsigned char row, unsigned char col);
51 void set_position(unsigned char address);
52 void write_char(unsigned char inputData);
53 void LEDcommand(unsigned char command);
54
55 void GPIOD01Handler(void);
56 void GPIOF00Handler(void);
57 void GPIOC04Handler(void);
58 void GPIOH05Handler(void);
59
60
61 #endif /* DISPLAY_H_ */
62
```

display.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           display.c

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

last modified:                  2013/05/26

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code LED Display

*/


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


#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib/timer.h"
#include "header/display.h"
#include "stdio.h"
#include "header/config.h"
#include "header/mycmdline.h"

// Instance of display mode
static volatile display_handler_t handlerState;

//*******************************************************************************
//Extern variables for Clock
//*******************************************************************************
extern volatile unsigned long clock_msec;
extern volatile unsigned long clock_sec;
extern volatile unsigned long clock_min;
extern volatile unsigned long clock_hour;
```

display.c

```
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 unsigned long timer2minute;
76 extern volatile unsigned int cycle_active;
77
78 void display_init(void)
79 {
80     UARTprintf("Initializing Display...");
81     //
82     // Enable the peripherals used by this example.
83     //
84     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
85     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
86
87     GPIOPinTypeGPIOOutput(LEDPORTCTRL, RS | RW | E ); // set RS | R/W | E as digital output
88     GPIOPinTypeGPIOOutput(LEDPORTDATA, D4 | D5 | D6 | D7); // set  D4 | D5 | D6 | D7 as
   digital output
89
90     GPIOPinWrite(LEDPORTCTRL, RS | RW | E, 0x0); // set RS | R/W | E to logic "0"
91     GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7 , 0x0); // set  D4 | D5 | D6 | D7 to logic
   "0"
92
93     unsigned long ul_delay_count;
94
95     // ca. 20 ms delay für Display reset
96     ul_delay_count = 200000;
97     while (ul_delay_count) ul_delay_count--;
98     //--------------------------------
99
100    //
101    //Function set: 8 bit Interface = 0x30
102    //
103    GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
104    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
105
106    // ca. 0.45 ms delay warten Enable High
107    ul_delay_count = 4500;
108    while (ul_delay_count) ul_delay_count--;
109
110    GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
111
112    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);        // Reset pins
113
114    // ca. 10 ms delay warten für nächsten Befehl
115    ul_delay_count = 100000;
116    while (ul_delay_count) ul_delay_count--;
117
118    //
119    //Function set: 8 bit Interface = 0x30
120    //
121    GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
122    GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
```

display.c

```
123
124     // ca. 0.45 ms delay warten Enable High
125     ul_delay_count = 4500;
126     while (ul_delay_count) ul_delay_count--;
127
128     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
129
130     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
131
132     // ca. 5 ms delay warten für nächsten Befehl
133     ul_delay_count = 50000;
134     while (ul_delay_count) ul_delay_count--;
135
136
137     //
138     //Function set: 8 bit Interface = 0x30
139     //
140     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
141     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x3);
142
143     // ca. 0.45 ms delay warten Enable High
144     ul_delay_count = 4500;
145     while (ul_delay_count) ul_delay_count--;
146
147     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
148
149     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);    // Reset pins
150
151     // ca. 2 ms delay warten für nächsten Befehl
152     ul_delay_count = 20000;
153     while (ul_delay_count) ul_delay_count--;
154
155
156
157     //
158     //Function set: 4 bit Interface = 0x20
159     //
160     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
161     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x2);
162
163     // ca. 0.45 ms delay warten Enable High
164     ul_delay_count = 4500;
165     while (ul_delay_count) ul_delay_count--;
166
167     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
168
169     GPIOPinWrite(LEDPORTDATA, D4 | D5, 0x0);        // Reset pins
170
171
172     // ca. 1 ms delay warten für nächsten Befehl
173     ul_delay_count = 10000;
174     while (ul_delay_count) ul_delay_count--;
175
176     clear_display();
177
178     //Function set: 4 bit Interface, 2(4) rows, 5x7dot = 0x28
179     LEDcommand(0x28);
180     //Display on, cursor off, blink off = 0x0C
181     LEDcommand(0x0C);
182     //Entry Mode set: Increment, No shift = 0x06
183     LEDcommand(0x06);
184
```

display.c

```
185
186     //
187     // Enable Timer 1
188     //
189     TimerEnable(TIMER1_BASE, TIMER_A);
190
191
192     UARTprintf("done\n");
193 }
194
195 void buttons_init(void)
196 {
197     UARTprintf("Initializing Buttons...");
198
199     //
200     // Enable the GPIO that is used for the first push button (GPIOF_PIN0).
201     //
202     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
203     ROM_GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
204     ROM_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA,
205                          GPIO_PIN_TYPE_STD_WPU);
206
207     //
208     // Enable the GPIO that is used for the second push button (GPIOD_PIN1).
209     //
210     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
211     ROM_GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_1);
212     ROM_GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_1, GPIO_STRENGTH_2MA,
213                          GPIO_PIN_TYPE_STD_WPU);
214
215     //
216     // Enable the GPIO that is used for the third push button (GPIOC_PIN4).
217     //
218     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
219     ROM_GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_4);
220     ROM_GPIOPadConfigSet(GPIO_PORTC_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
221                          GPIO_PIN_TYPE_STD_WPU);
222
223
224     //
225     // Enable the GPIO that is used for the forth push button (GPIOH_PIN5).
226     //
227     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
228     ROM_GPIOPinTypeGPIOInput(GPIO_PORTH_BASE, GPIO_PIN_5);
229     ROM_GPIOPadConfigSet(GPIO_PORTH_BASE, GPIO_PIN_5, GPIO_STRENGTH_2MA,
230                          GPIO_PIN_TYPE_STD_WPU);
231
232
233     //
234     // Enable interrupts
235     //
236     ROM_IntEnable(INT_GPIOF);
237     GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_RISING_EDGE);  //trigger on rising edge
238     GPIOPinIntEnable(GPIO_PORTF_BASE,GPIO_PIN_0); //enable interrupt on PF0
239
240     ROM_IntEnable(INT_GPIOD);
241     GPIOIntTypeSet(GPIO_PORTD_BASE,GPIO_PIN_1,GPIO_RISING_EDGE);  //trigger on rising edge
242     GPIOPinIntEnable(GPIO_PORTD_BASE,GPIO_PIN_1); //enable interrupt on PD1
243
244     ROM_IntEnable(INT_GPIOC);
245     GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_4,GPIO_RISING_EDGE);  //trigger on rising edge
246     GPIOPinIntEnable(GPIO_PORTC_BASE,GPIO_PIN_4); //enable interrupt on PC4
```

display.c

```
247
248     ROM_IntEnable(INT_GPIOH);
249     GPIOIntTypeSet(GPIO_PORTH_BASE,GPIO_PIN_5,GPIO_RISING_EDGE);  //trigger on rising edge
250     GPIOPinIntEnable(GPIO_PORTH_BASE,GPIO_PIN_5); //enable interrupt on PC4
251
252     // Set statemachine to state Start
253     handlerState = START;
254     UARTprintf("done\n");
255 }
256
257 //*****************************************************************************
258 //
259 // This is the handler for the GPIO PORT F PIN 0 interrupt. (Button1)
260 //
261 //*****************************************************************************
262 void GPIOF00Handler(void){
263
264     //Disable Interrupts
265     TimerDisable(TIMER1_BASE, TIMER_A);
266     ROM_IntDisable(INT_GPIOF);
267     GPIOPinIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);
268
269     clear_display();
270
271         // Switch state
272         if(handlerState == CELLA)
273         handlerState = CELLD;
274
275         else if(handlerState == CELLD)
276         handlerState = CELLC;
277
278         else if(handlerState == CELLC)
279         handlerState = CELLB;
280
281         else if(handlerState == CELLB)
282         handlerState = CELLA;
283
284         else if(handlerState == MEAS)
285             Cmd_interprete("start");
286
287         else if(handlerState == MEASSTOP){
288             Cmd_interprete("stop");
289             handlerState = MEAS;
290         }
291
292     //Enable Interrupts
293     ROM_IntEnable(INT_GPIOF);
294     TimerEnable(TIMER1_BASE, TIMER_A);
295 }
296
297 //*****************************************************************************
298 //
299 // This is the handler for the GPIO PORT D PIN 1 interrupt. (Button2)
300 //
301 //*****************************************************************************
302 void GPIOD01Handler(void){
303
304     //Disable Interrupts
305     TimerDisable(TIMER1_BASE, TIMER_A);
306     GPIOPinIntClear(GPIO_PORTD_BASE,GPIO_PIN_1);
307
308
```

display.c

```
309     clear_display();
310
311         // Switch state
312         switch(handlerState){
313         case START: handlerState = CLOCK; break;
314         case CLOCK: handlerState = MEAS; break;
315         case MEAS:
316         case MEASSTOP: handlerState = CELLA; break;
317         case CELLA: handlerState = CURRENT;
318         case CELLB: handlerState = CURRENT;
319         case CELLC: handlerState = CURRENT;
320         case CELLD: handlerState = CURRENT; break;
321         case CURRENT: handlerState = IP; break;
322         case IP: handlerState = CONF; break;
323         case CONF: handlerState = START; break;
324
325         }
326
327     //Enable Interrupts
328     TimerEnable(TIMER1_BASE, TIMER_A);
329 }
330
331
332 //*****************************************************************************
333 //
334 // This is the handler for the GPIO PORT C PIN 4 interrupt. (Button3)
335 //
336 //*****************************************************************************
337 void GPIOC04Handler(void){
338     //Disable Interrupts
339     TimerDisable(TIMER1_BASE, TIMER_A);
340     ROM_IntDisable(INT_GPIOC);
341     GPIOPinIntClear(GPIO_PORTC_BASE,GPIO_PIN_4);
342
343     clear_display();
344
345         // Switch state
346         if(handlerState == CELLA)
347         handlerState = CELLB;
348
349         else if(handlerState == CELLB)
350         handlerState = CELLC;
351
352         else if(handlerState == CELLC)
353         handlerState = CELLD;
354
355         else if(handlerState == CELLD)
356         handlerState = CELLA;
357
358         else if(handlerState == MEAS)
359         handlerState = MEASSTOP;
360
361         else if(handlerState == MEASSTOP)
362         handlerState = MEAS;
363
364     //Enable Interrupts
365     ROM_IntEnable(INT_GPIOC);
366     TimerEnable(TIMER1_BASE, TIMER_A);
367 }
368
369 //*****************************************************************************
370 //
```

display.c

```
371 // This is the handler for the GPIO PORT H PIN 5 interrupt. (Button4)
372 // NO FUNCTION YET!
373 //
374 //*****************************************************************************
375 void GPIOH05Handler(void){
376     //Disable Interrupts
377     TimerDisable(TIMER1_BASE, TIMER_A);
378     ROM_IntDisable(INT_GPIOH);
379     GPIOPinIntClear(GPIO_PORTH_BASE,GPIO_PIN_5);
380 
381         clear_display();
382 
383     //Enable Interrupts
384     ROM_IntEnable(INT_GPIOH);
385     TimerEnable(TIMER1_BASE, TIMER_A);
386 }
387 
388 
389 //*****************************************************************************
390 //
391 // The  handler for the Timer1A interrupt.
392 // Refresh the Content of the LED-Display dependen on current state
393 //
394 //*****************************************************************************
395 void Timer1IntHandler(void){
396 
397     char display_buffer[21];
398     int i = 0;
399 
400     //Write content to Displays
401     switch(handlerState) {
402 
403         case START:
404 
405             write_to_display("********************", 0, 0);
406             write_to_display("***    Battery   ***", 1, 0);
407             write_to_display("**  Cycle Machine **", 2, 0);
408             write_to_display("*****   MENU   *****", 3 ,0);
409             break;
410 
411         case CLOCK:
412 
413             sprintf(display_buffer, "      %02d:%02d:%02d      " , clock_hour, clock_min,
    clock_sec);
414             write_to_display(display_buffer, 0, 0);
415             sprintf(display_buffer, "     %02d.%02d.%04d     " , clock_day, clock_month,
    clock_year);
416             write_to_display(display_buffer, 1, 0);
417             write_to_display("        MENU        ", 3 ,0);
418             break;
419 
420         case MEAS:
421 
422             sprintf(display_buffer, "Measurment Cycling: ");
423             write_to_display(display_buffer, 0, 0);
424 
425             if(cycle_active)
426                 sprintf(display_buffer, "       active       ");
427             else
428                 sprintf(display_buffer, "     not active     ");
429             write_to_display(display_buffer, 1, 0);
430 
```

display.c

```
431
432             write_to_display("START   MENU    STOP", 3 ,0);
433             break;
434
435         case MEASSTOP:
436
437             sprintf(display_buffer, "Measurment Cycling: ");
438             write_to_display(display_buffer, 0, 0);
439
440             sprintf(display_buffer, "    Confirm STOP    ");
441
442             write_to_display(display_buffer, 1, 0);
443
444
445             write_to_display(" YES    MENU     NO ", 3 ,0);
446             break;
447
448         case CELLA:
449
450             for(i = 0; i <= 2; i++){
451                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
452                                                                         ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
453                 write_to_display(display_buffer, i, 0);
454             }
455
456             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
457             break;
458
459         case CELLB:
460
461             for(i = 3; i <= 5; i++){
462                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
463                                                                         ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
464                 write_to_display(display_buffer, i-3, 0);
465             }
466
467             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
468             break;
469
470         case CELLC:
471
472             for(i = 6; i <= 8; i++){
473                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
474                                                                         ,
    g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
475                 write_to_display(display_buffer, i-6, 0);
476             }
477
478             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
479             break;
480
481         case CELLD:
482
483             for(i = 9; i <= 11; i++){
484                 sprintf(display_buffer, "C%02d: %01d.%06dV %02d.%01dC", i+1,
    g_iCellVoltages[i]/1000000, g_iCellVoltages[i]%1000000
485                                                                         ,
```

display.c

```
   g_iCellTemperature[i]/10, g_iCellTemperature[i]%10);
486                 write_to_display(display_buffer, i-9, 0);
487             }
488
489             write_to_display("<PREV   MENU   NEXT>", 3 ,0);
490             break;
491
492         case CURRENT:
493
494             sprintf(display_buffer, "      Current:      ");
495             write_to_display(display_buffer, 0, 0);
496             sprintf(display_buffer, "    %d.%05d A     "  ,  g_iCurrent/1000  ,
   abs(g_iCurrent%1000));
497             write_to_display(display_buffer, 1, 0);
498             write_to_display("        MENU        ", 3 ,0);
499             break;
500
501         case IP:
502
503             sprintf(display_buffer, "     IP-Address:    ");
504             write_to_display(display_buffer, 0, 0);
505             sprintf(display_buffer, "   %03d.%03d.%03d.%03d  "  , config.ip_a, config.ip_b,
   config.ip_c, config.ip_d);
506             write_to_display(display_buffer, 1, 0);
507             write_to_display("        MENU        ", 3 ,0);
508             break;
509
510         case CONF:
511
512             sprintf(display_buffer, "Quantity Cells:%02d   ", config.quantity_cells);
513             write_to_display(display_buffer, 0, 0);
514             sprintf(display_buffer, "Cyclestep(sec):%04d "  , config.cyclestep);
515             write_to_display(display_buffer, 1, 0);
516             sprintf(display_buffer, "Time left(min):%05d"  , (config.measuretime -
   timer2minute ));
517             write_to_display(display_buffer, 2, 0);
518             write_to_display("        MENU        ", 3 ,0);
519             break;
520     }
521
522     //
523     // Clear the timer interrupt.
524     //
525     TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
526 }
527
528
529 //*****************************************************************************
530 //
531 // Clear the LED-Display
532 //
533 //*****************************************************************************
534 void clear_display(void){
535
536 LEDcommand(0x01);
537
538 }
539
540 //*****************************************************************************
541 //
542 // Write string to Display by row and column
543 //
```

display.c

```
544 //*****************************************************************************
545 void write_to_display(char* inputText, unsigned char row, unsigned char col) {
546     unsigned char address_d = 0;        // address of the data in the screen.
547 
548     //define address
549     switch(row)
550         {
551             case 0: address_d = 0x80 + col;     // at zeroth row
552             break;
553             case 1: address_d = 0xC0 + col;     // at first row
554             break;
555             case 2: address_d = 0x94 + col;     // at second row
556             break;
557             case 3: address_d = 0xD4 + col;     // at third row
558             break;
559             default: address_d = 0x80 + col;    // returns to first row if invalid row
    number is detected
560             break;
561         }
562 
563     //set position by address
564     set_position(address_d);
565 
566     // Place a string, letter by letter.
567     while(*inputText)
568         write_char(*inputText++);
569 }
570 
571 //*****************************************************************************
572 //
573 // Set the position by address
574 //
575 //*****************************************************************************
576 void set_position(unsigned char address) {
577 
578     unsigned long ul_delay_count;
579 
580     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
581     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0xf0) >> 4);
582 
583     // ca. 0.45 ms delay warten Enable High
584     ul_delay_count = 4500;
585     while (ul_delay_count) ul_delay_count--;
586 
587     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
588 
589     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
590 
591 
592     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
593     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (address & 0x0f));
594 
595     // ca. 0.45 ms delay warten Enable High
596     ul_delay_count = 4500;
597     while (ul_delay_count) ul_delay_count--;
598 
599     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
600 
601     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x00);  // Reset pins
602 
603 }
604 
```

display.c

```
605 //*****************************************************************************
606 //
607 // Write string to Display
608 //
609 //*****************************************************************************
610 void write_char(unsigned char inputData) {
611
612     unsigned long ul_delay_count;
613
614     GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);               // Enable the Display and
    writeoperation
615     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0xf0) >> 4);
616
617     // ca. 0.45 ms delay warten Enable High
618     ul_delay_count = 4500;
619     while (ul_delay_count) ul_delay_count--;
620
621     GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);            // Disable the Display
622
623
624     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
625
626
627
628     GPIOPinWrite(LEDPORTCTRL, E | RS, 0x05);               // Enable the Display and
    writeoperation
629     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (inputData & 0x0f));
630
631     // ca. 0.45 ms delay warten Enable High
632     ul_delay_count = 4500;
633     while (ul_delay_count) ul_delay_count--;
634
635     GPIOPinWrite(LEDPORTCTRL, E | RS, 0x00);            // Disable the Display
636
637
638     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , 0x0);  // Reset pins
639
640 }
641
642 //*****************************************************************************
643 //
644 // Send Command to Display
645 //
646 //*****************************************************************************
647 void LEDcommand(unsigned char command){
648
649     unsigned long ul_delay_count;
650
651     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
652     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0xf0) >> 4);
653
654     // ca. 0.45 ms delay warten Enable High
655     ul_delay_count = 4500;
656     while (ul_delay_count) ul_delay_count--;
657
658     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
659
660     GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0);  // Reset pins
661
662
663     // ca. 0,1 ms delay warten für nächsten Befehl
664     ul_delay_count = 1000;
```

display.c

```
665     while (ul_delay_count) ul_delay_count--;
666
667
668     GPIOPinWrite(LEDPORTCTRL, E, E);                // Enable the Display
669     GPIOPinWrite(LEDPORTDATA,  D4 | D5 | D6 | D7 , (command & 0x0f));
670
671     // ca. 0.45 ms delay warten Enable High
672     ul_delay_count = 4500;
673     while (ul_delay_count) ul_delay_count--;
674
675     GPIOPinWrite(LEDPORTCTRL, E, 0x00);             // Disable the Display
676
677     GPIOPinWrite(LEDPORTDATA, D4 | D5 | D6 | D7, 0x0);  // Reset pins
678
679     // ca. 1 ms delay warten für nächsten Befehl
680     ul_delay_count = 10000;
681     while (ul_delay_count) ul_delay_count--;
682 }
683
684
685
```

ethernet.h

```
 1 /*
 2  * ethernet.h
 3  *
 4  *  Created on: 07.03.2013
 5  *      Author: Thomas W. / Tobias S. / Fabian S.
 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
20 #endif /* ETHERNET_H_ */
21
```

ethernet.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           ethernet.c

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

last modified:                  2013/05/28

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

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


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

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


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

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

ethernet.c

```
63 static unsigned long g_ulTwirlPos = 0;
64
65
66 // Aktuelle IP Adresse
67 static unsigned long g_ulLastIPAddr = 0;
68
69 unsigned long ulUser0, ulUser1;
70 // MAC Adresse
71 unsigned char pucMACArray[8];
72
73
74 /*
75  * Initialize Ethernetfunction
76  */
77 int init_ethernet(void)
78 {
79         unsigned long ul_delay_count;
80
81         UARTprintf("Initializing Ethernet:\n");
82
83         SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOE;
84         volatile unsigned long ulLoop;
85         ulLoop = SYSCTL_RCGC2_R;
86
87         GPIO_PORTE_DIR_R = 0x00;  // Auf Input setzen
88         GPIO_PORTE_DEN_R = 0xFF;  // Einschalten der GPIOs
89         GPIO_PORTE_PUR_R = 0xFF; // Pull Ups aktivieren
90
91         // Einschalten und Resetten des Eternet Controllers
92         ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_ETH);
93         ROM_SysCtlPeripheralReset(SYSCTL_PERIPH_ETH);
94
95
96         ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
97         GPIOPinConfigure(GPIO_PF2_LED1);
98         GPIOPinConfigure(GPIO_PF3_LED0);
99         GPIOPinTypeEthernetLED(GPIO_PORTF_BASE, GPIO_PIN_2 | GPIO_PIN_3);
100
101        // Lesen der MAC-Adresse aus den User-Registern
102        // Diese scheint ab Werk programmiert zu sein
103        ROM_FlashUserGet(&ulUser0, &ulUser1);
104        if((ulUser0 == 0xffffffff) || (ulUser1 == 0xffffffff))
105        {
106            // We should never get here.  This is an error if the MAC address has
107            // not been programmed into the device.  Exit the program
108            UARTprintf("  MAC Address Not Programmed!\n");
109            return 0;
110        }
111
112        // Umwandlung der 24/24 MAC Adresse aus dem NV RAM in eine 32/16 MAC
113        // Adresse. Anchließend wird die MAC dem Ethernet Controller
114        // übergeben
115        pucMACArray[0] = ((ulUser0 >>  0) & 0xff);
116        pucMACArray[1] = ((ulUser0 >>  8) & 0xff);
117        pucMACArray[2] = ((ulUser0 >> 16) & 0xff);
118        pucMACArray[3] = ((ulUser1 >>  0) & 0xff);
119        pucMACArray[4] = ((ulUser1 >>  8) & 0xff);
120        pucMACArray[5] = ((ulUser1 >> 16) & 0xff);
121
122        // Initialisierung von lwIP, mit DHCP.
123        //lwIPInit(pucMACArray, 0, 0, 0, IPADDR_USE_DHCP);
124
```

ethernet.c

```
125         // Initialisierung von lwIP, mit statischer IP.
126         unsigned long lowerIP = GPIO_PORTE_DATA_R;
127         struct ip_addr local_addr;
128
129         IP4_ADDR(&local_addr,config.ip_d,config.ip_c,config.ip_b,config.ip_a);
130         lwIPInit(pucMACArray, local_addr.addr, 0xFFFFFF00, 0, IPADDR_USE_STATIC);
131
132         // Setup the device locator service
133         LocatorInit();
134         LocatorMACAddrSet(pucMACArray);
135         LocatorAppTitleSet("LM3S9D92 enet_lwip");
136
137         // ca. 30 ms delay for stabilize Ethernetconnection
138         ul_delay_count = 300000;
139         while (ul_delay_count) ul_delay_count--;
140         //--------------------------------
141
142         return 1;
143     }
144
145 /*
146  * Timer Interrupt für Ethernet spezifische Aktionen.
147  * Hier wird u.a. erkannt, wenn eine neue IP-Adresse zugewiesen
148  * wurde. Die Neue IP wird dann über UART ausgegeben.
149  */
150 void lwIPHostTimerHandler(void)
151 {
152     unsigned long ulIPAddress;
153
154     // Get the local IP address.
155     ulIPAddress = lwIPLocalIPAddrGet();
156
157     // See if an IP address has been assigned.
158     if(ulIPAddress == 0)
159     {
160         // Draw a spinning line to indicate that the IP address is being
161         // discoverd.
162
163         UARTprintf("\b%c", g_pcTwirl[g_ulTwirlPos]);
164
165         // Update the index into the twirl.
166         g_ulTwirlPos = (g_ulTwirlPos + 1) & 3;
167     }
168
169     // Check if IP address has changed, and display if it has
170     else if(ulIPAddress != g_ulLastIPAddr)
171     {
172         // Ausgabe der neuen MAC Adresse
173
174         UARTprintf("\n  MAC: %x:%x:%x:%x:%x:%x\n", pucMACArray[0],
175         pucMACArray[1], pucMACArray[2], pucMACArray[3],
176         pucMACArray[4], pucMACArray[5]);
177
178         //
179         // Ausgabe der neuen IP Adresse
180
181         UARTprintf("  IP: %d.%d.%d.%d\n", ulIPAddress & 0xff,
182         (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
183         (ulIPAddress >> 24) & 0xff);
184
185         // Speichern der neuen IP-Adresse
186         g_ulLastIPAddr = ulIPAddress;
```

ethernet.c

```
187
188         // Ausgabe der neuen Netzmaske
189         ulIPAddress = lwIPLocalNetMaskGet();
190
191         UARTprintf("  Netmask: %d.%d.%d.%d\n", ulIPAddress & 0xff,
192         (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
193         (ulIPAddress >> 24) & 0xff);
194
195         //
196         // Ausgabe der neuen Gateway Adresse
197         //
198
199         UARTprintf("  Gateway: %d.%d.%d.%d\n", ulIPAddress & 0xff,
200         (ulIPAddress >> 8) & 0xff, (ulIPAddress >> 16) & 0xff,
201         (ulIPAddress >> 24) & 0xff);
202
203         discover_init();  // Initialisierung des UDP Broadcast Dienstes
204         control_init();   // Initialisierung der TCP-Steuerung
205
206         UARTprintf("\nInitialization Ethernet...done\n\n");
207     }
208
209 }
210
211
```

led.h

```
 1 /*
 2  * led.h
 3  *
 4  *  Created on: 03.06.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef LED_H_
 9 #define LED_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void led_init(void);
17
18
19
20 #endif /* LED_H_ */
21
```

led.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           led.c

Auhtor:                         Thomas Wisnewski
Credits:

last modified:                  2013/05/27

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Initialize GPIOs for Frontpanel LEDs

*/

#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"

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


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

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

    UARTprintf("done\n");

}

```

lm3s9d92.cmd

```
/*****************************************************************************
 *
 * Default Linker Command file for the Texas Instruments LM3S9D92
 *
 * This is part of revision 8132 of the Stellaris Peripheral Driver Library.
 *
 *****************************************************************************/

--retain=g_pfnVectors

MEMORY
{
    FLASH (RX) : origin = 0x00000000, length = 0x00080000
    SRAM (RWX) : origin = 0x20000000, length = 0x00018000
}

/* The following command line options are set as part of the CCS project.    */
/* If you are building using the command line, or for some reason want to    */
/* define them here, you can uncomment and modify these lines as needed.     */
/* If you are using CCS for building, it is probably better to make any such */
/* modifications in your CCS project and leave this file alone.              */
/*                                                                           */
/* --heap_size=0                                                             */
/* --stack_size=256                                                          */
/* --library=rtsv7M3_T_le_eabi.lib                                           */

/* Section allocation in memory */

SECTIONS
{
    .intvecs:   > 0x00000000
    .text   :   > FLASH
    .const  :   > FLASH
    .cinit  :   > FLASH
    .pinit  :   > FLASH

    .vtable :   > 0x20000000
    .data   :   > SRAM
    .bss    :   > SRAM
    .sysmem :   > SRAM
    .stack  :   > SRAM
}

__STACK_TOP = __stack + 14336;

```

main.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           main.c

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

last modified:                  2013/05/26

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Main File for BATSEN Battery Cycling Machine
*/


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


/*****************************************************************************
*
* Own Includings
*
*****************************************************************************/
#include "utils/uartstdio.h"
#include "driverlib/rom_map.h"
#include "header/myuart.h"
#include "header/mysdcard.h"
#include "header/myssi.h"
#include "header/myadc.h"
#include "header/relais.h"
#include "header/config.h"
#include "header/mycmdline.h"
#include "header/clocktimer.h"
#include "header/temperature.h"
#include "header/ethernet.h"
```

main.c

```
 63 #include "header/control.h"
 64 #include "header/mypwm.h"
 65 #include "header/display.h"
 66 #include "header/rtc.h"
 67 #include "header/led.h"
 68
 69 /*******************************************************************************
 70 *
 71 * Globals
 72 *
 73 *******************************************************************************/
 74 extern volatile unsigned int cycle_active;
 75 extern volatile unsigned int power_relay_active;
 76
 77 // Instanz der Konfiguration
 78 config_t config;
 79
 80 // Define global location buffer
 81 char *g_cLocalBuf;
 82
 83 // Define Command line pointer
 84 tCmdLineEntry *g_psCmdTable;
 85
 86 // Define string for main location
 87 const char *g_cMainLocalBuf = "Main";
 88
 89 // define command set for main program
 90 tCmdLineEntry g_sMainCmdTable[] =
 91 {
 92     { "help",   Cmd_help,            "   : Display list of commands" },
 93     { "h",      Cmd_help,          "      : alias for help" },
 94     { "?",      Cmd_help,          "      : alias for help" },
 95     { "browser",my_start_cmd_line,      ": SD-Card browser" },
 96     { "config", config_cmd_line,       " : Configuration" },
 97     { "alive",  Cmd_alive,            "  : Check if Cycling is active" },
 98     { "start",  Cmd_start,            "  : Start measurement cycling" },
 99     { "stop",   Cmd_stop,            "   : Stop measurement cycling" },
100     { 0, 0, 0 }
101 };
102
103
104 void main(void) {
105
106     //
107     // Set the system clock to run at 50MHz from the PLL.
108     //
109     ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
110                        SYSCTL_XTAL_16MHZ);
111
112
113     //
114     // Enable processor interrupts.
115     //
116     IntMasterEnable();
117
118
119     //initializing program
120     myUARTinit();
121     buttons_init();
122     ssi1_init();
123     mysdcardinit();
124     init_config();
```

main.c

```
125     ssi1_init();
126     myadc_init();
127     init_ethernet();
128     init_relais();
129     init_clock_timer();
130     rtc_init();
131     init_temperature();
132     pwm_init();
133     display_init();
134     led_init();
135
136
137
138     //Set Priority of interrupts
139     ROM_IntPrioritySet(FAULT_SYSTICK,   0x00);  //1st priority
140     ROM_IntPrioritySet(INT_ADC0SS0,     0x20);  //2nd priority
141     ROM_IntPrioritySet(INT_ADC1SS1,     0x20);  //2nd priority
142     ROM_IntPrioritySet(INT_UART0,       0x20);  //2nd priority
143     ROM_IntPrioritySet(INT_TIMER0A,     0x40);  //3th priority
144     ROM_IntPrioritySet(INT_TIMER2A,     0x60);  //4th priority
145     ROM_IntPrioritySet(INT_TIMER1A,     0x60);  //4th priority
146     ROM_IntPrioritySet(INT_GPIOC,       0x80);  //5th priority
147     ROM_IntPrioritySet(INT_GPIOD,       0x80);  //5th priority
148     ROM_IntPrioritySet(INT_GPIOF,       0x80);  //5th priority
149     ROM_IntPrioritySet(INT_ETH,         0xE0);  //lowest priority
150     ROM_IntPrioritySet(INT_SYSCTL,      0xE0);  //lowest priority
151
152
153     // set Command line pointer to the beginning of the main command structure
154     g_psCmdTable = &g_sMainCmdTable[0];
155
156     // set location buffer to main
157     g_cLocalBuf = (char*)g_cMainLocalBuf;
158
159     //
160     // Print a prompt to the console.
161     //
162     UARTprintf("\n\n\n");
163     UARTprintf("***********************************************\n");
164     UARTprintf("***        Battery Cycle Machine          ****\n");
165     UARTprintf("***********************************************\n");
166
167     UARTprintf("\n Enter Command (type <help> to see list of commands):");
168     UARTprintf("\n%s: > ", g_cLocalBuf);
169
170     //calibrate_adc();
171
172     unsigned long ul_delay_count;
173
174     //
175     // Loop forever
176     //
177     while(1)
178     {
179         //Toggle LED in Cycling Mode
180         if(cycle_active)
181             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4, ~GPIOPinRead(GPIO_PORTJ_BASE,
    GPIO_PIN_4));
182         else
183             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);
184
185         //Activate LED "current" if one of powerrelays turned on
```

main.c

```
186         if(power_relay_active)
187             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  GPIO_PIN_7);
188         else
189             GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_7,  0x00);
190
191         // ca. 500 ms delay
192         ul_delay_count = 500000;
193         while (ul_delay_count) ul_delay_count--;
194         //--------------------------------
195
196     }
197 }
198
199
200
```

myadc.h

```
 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
```

myadc.c

```
 1 /*
 2 Project:                          battery_cycling_sw_v3
 3 File:                             myadc.c
 4
 5 Auhtor:                           Thomas Wisnewski
 6 Credits:                          Matthias Schneider
 7                                   Tobias Steinmann
 8                                   Fabian Schwartau
 9                                   Stellaris Ware
10
11 last modified:                   2013/05/28
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:                     Source Code for AD7798 usage
24
25 */
26
27
28
29
30 /*****************************************************************************
31 *
32 * Includings
33 *
34 *****************************************************************************/
35 #include "driverlib/uart.h"
36 #include "utils/uartstdio.h"
37 #include "driverlib/rom_map.h"
38 #include "inc/hw_ints.h"
39 #include "inc/hw_memmap.h"
40 #include "driverlib/rom.h"
41 #include "driverlib/gpio.h"
42 #include "driverlib/pin_map.h"
43 #include "driverlib/sysctl.h"
44 #include "driverlib/ssi.h"
45 #include "inc/hw_ssi.h"
46
47 /*****************************************************************************
48 *
49 * Own Includings
50 *
51 *****************************************************************************/
52 #include "header/myadc.h"
53 #include "header/config.h"
54 #include "header/relais.h"
55 #include "header/myssi.h"
56
57 //*****************************************************************************
58 //
59 // A global data buffer used for actual Cell voltage
60 //
61 //*****************************************************************************
62 volatile unsigned int g_iCellVoltages[12] = {0};
```

myadc.c

```
63 volatile unsigned int g_iCurrent = 0;
64
65 // AD7798 initialization and configuration
66 int myadc_init(void)
67 {
68
69     UARTprintf("Initialization of external ADC...");
70     unsigned long ul_delay_count;
71
72     // SSI1 Fifo löschen
73     adc_clear_fifo();
74
75     // ADC reset
76     spiChipSelect(ADC);
77     adc_write_read(0xFF);
78     adc_write_read(0xFF);
79     adc_write_read(0xFF);
80     adc_write_read(0xFF);
81     spiChipSelect(NONE);
82
83     // ca. 500 ms delay für ADC reset
84     ul_delay_count = 10000000;
85     while (ul_delay_count) ul_delay_count--;
86     //--------------------------------
87
88     // Setzen des "configuration register"
89     // unipolar mode, gain = 1, buffered, noref = 0x1010
90     //adc_set_config_reg(0x1010);
91     // bipolar mode, gain = 1, buffered, noref = 0x0010
92     //adc_set_config_reg(0x0010);
93     // bipolar mode, gain = 1, unbuffered, noref = 0x0000
94     //adc_set_config_reg(0x0000);
95     // unipolar mode, gain = 1, unbuffered, noref = 0x1000
96     adc_set_config_reg(0x1000);
97
98      // In den "power down" Modus gehen, um das "mode register" zu ändern
99     adc_set_mode_reg(0x600A);
100    adc_set_offset_reg(0);  // Initialisierung des "offset registers" mit 0
101    adc_set_fs_reg(0); // Initialisierung des "full scale registers" mit 0
102
103    // Prüfen, ob die Werte korrekt geschrieben wurden
104    if(adc_read_offset_reg() != 0)
105    {
106        UARTprintf("failed: offset calibration reset failed!\n");
107        return 0;
108    }
109    if(adc_read_fs_reg() != 0)
110    {
111        UARTprintf("failed: full scale calibration reset failed!\n");
112        return 0;
113    }
114
115    // Durchführung der internen Kalibrierung des ADCs
116    adc_set_mode_reg(0x800A); // Internal zero offset calibration
117    while((adc_status() & (1<<7)));
118    adc_set_mode_reg(0xA00A); // Internal full scale calibration
119    while((adc_status() & (1<<7)));
120
121    // Prüfen, ob die Kalibierung erfolgreich war
122    // (sich die Werte geändert haben)
123    if(adc_read_offset_reg() == 0)
124    {
```

myadc.c

```
125         UARTprintf("failed: offset calibration failed!\n");
126         return 0;
127     }
128     if(adc_read_fs_reg() == 0)
129     {
130         UARTprintf("failed: full scale calibration failed!\n");
131         return 0;
132     }
133 
134     //Calibrate HALL-Sensor to 0
135     config.zerocurrentch1 = (32768 - adc_get_single_value_ch2());
136     config.zerocurrentch2 = (32768 - adc_get_single_value_ch3());
137 
138     UARTprintf("done.\n");
139     return 1;
140 
141 
142 }
143 
144 // Get the voltage from the ADC
145 int adc_get_voltage(unsigned int choose)
146 {
147     unsigned long ul_delay_count;
148 
149     switch_relais(choose);
150 
151     // ca. 10 ms delay für ADC reset
152     ul_delay_count = 100000;
153     while (ul_delay_count) ul_delay_count--;
154     //--------------------------------
155 
156 
157     unsigned int voltage = (unsigned long long)adc_get_single_value() * config.gain[choose]
    / 1000000 + config.offset[choose];
158     switch_relais(0);
159 
160     if(voltage <= config.offset[choose] * 2){
161         voltage = 0;
162     }
163 
164     g_iCellVoltages[choose - 1] = voltage;
165 
166 return voltage;
167 }
168 
169 // Get the current from the ADC
170 int adc_get_current(void){
171 
172     long current = (unsigned long long)adc_get_single_value_ch2() * 76293945 / 1000000000 *
    3745  / 100 - 93625;
173 
174     if(abs(current) <= config.zerocurrentch1 * 2){
175         current = 0;
176     }
177 
178     if(current <= -75000 || current >= 75000){
179         current = (unsigned long long)adc_get_single_value_ch3() * 76293945 / 1000000000 *
    250 - 625000;
180 
181         if(abs(current) <= config.zerocurrentch2 * 2){
182             current = 0;
183         }
```

myadc.c

```
184     }
185
186     g_iCurrent = current;
187
188 return (int) current;
189 }
190
191
192
193 // Get the value of the ADC Ch1
194 unsigned short adc_get_single_value(void)
195 {
196     unsigned short data;
197
198     // Configure unipolar mode, gain = 1, unbuffered, noref , CH1 = 0x1000
199     adc_set_config_reg(0x1000);
200
201     // set ADC to single conversion mode with f_ADC = 123 Hz = 0x2003
202     // set ADC to single conversion mode with f_ADC = 470 Hz = 0x2001
203     adc_set_mode_reg(0x2003);
204
205     // wait for conversion to be done
206     while((adc_status() & (1<<7)));
207
208     data = (adc_read_data());
209
210     return data;
211 }
212
213 // Get the value of the ADC Ch2
214 unsigned short adc_get_single_value_ch2(void)
215 {
216     unsigned short data;
217
218     // Configure unipolar mode, gain = 1, unbuffered, noref , CH2 = 0x1001
219     adc_set_config_reg(0x1001);
220
221     // set ADC to single conversion mode with f_ADC = 123 Hz
222     adc_set_mode_reg(0x2003);
223
224     // wait for conversion to be done
225     while((adc_status() & (1<<7)));
226
227     data = (adc_read_data()) + config.zerocurrentch1;
228
229     return data;
230 }
231
232 // Get the value of the ADC Ch3
233 unsigned short adc_get_single_value_ch3(void)
234 {
235     unsigned short data;
236
237     // Configure unipolar mode, gain = 1, unbuffered, noref , CH3 = 0x1002
238     adc_set_config_reg(0x1002);
239
240     // set ADC to single conversion mode with f_ADC = 123 Hz
241     adc_set_mode_reg(0x2003);
242
243     // wait for conversion to be done
244     while((adc_status() & (1<<7)));
245
```

myadc.c

```
246     data = (adc_read_data())  + config.zerocurrentch2;
247
248     return data;
249 }
250
251 /*
252  * Auslesen des Konfigurations Registers des ADCs.
253  * Es werden hierfür zwei Bytes gelesen und anschließend
254  * zusammengesetzt.
255  */
256 unsigned short adc_read_config_reg(void)
257 {
258     unsigned short data;
259     spiChipSelect(ADC);
260     adc_write_read(0x50);
261     data = (adc_write_read(0x00))<<8;
262     data |= adc_write_read(0x00);
263     spiChipSelect(NONE);
264     return data;
265 }
266
267 /*
268  * Lesen des Status Registers vom ADC.
269  * Return: 1 Byte Status Register
270  */
271 unsigned char adc_status(void)
272 {
273     spiChipSelect(ADC);
274     adc_clear_fifo();
275     adc_write_read(0x40);
276     unsigned char status_reg = adc_write_read(0x00);
277
278     spiChipSelect(NONE);
279     return status_reg;
280 }
281
282 /*
283  * Schreib-, Leseoperation für den ADC über SSI
284  */
285 unsigned char adc_write_read(unsigned char data)
286 {
287
288     adc_clear_fifo();
289
290     unsigned long ans;
291     SSIDataPut(SSI1_BASE, data);
292     SSIDataGet(SSI1_BASE, &ans);
293
294     return (unsigned char)(ans&0xff);
295
296 }
297
298 /*
299  * Löschen des Hardware FIFOs vom SPI-Interface
300  */
301 void adc_clear_fifo(void)
302 {
303
304     unsigned long dummy;
305     while(SSIDataGetNonBlocking(SSI1_BASE, &dummy));  // clear the fifo
306
307 }
```

myadc.c

```
308
309 /*
310  * Auslesen des Fullscale Registers des ADC.
311  * Dieses ist zwei Bytes groß, die hier automatisch
312  * beide gelsen und zusammen gesetzt werden.
313  */
314 unsigned short adc_read_fs_reg(void)
315 {
316     unsigned short data;
317     spiChipSelect(ADC);
318     adc_write_read(0x78);
319     data = (adc_write_read(0x00))<<8;
320     data |= adc_write_read(0x00);
321     spiChipSelect(NONE);
322     return data;
323 }
324
325 /*
326  * Auslesen des Offsetregisters des ADC.
327  * Dieses ist zwei Bytes groß, die hier automatisch
328  * beide gelsen und zusammen gesetzt werden.
329  */
330 unsigned short adc_read_offset_reg(void)
331 {
332     unsigned short data;
333     spiChipSelect(ADC);
334     adc_write_read(0x70);
335     data = (adc_write_read(0x00))<<8;
336     data |= adc_write_read(0x00);
337     spiChipSelect(NONE);
338     return data;
339 }
340
341 /*
342  * Schreiben des Konfigurationsregsters
343  * Das Register ist 2 Bytes groß, es muss daher ein
344  * zwei Byte short Wert übergeben werden. Es erfolgt
345  * keine Prüfung.
346  */
347 void adc_set_config_reg(unsigned short value)
348 {
349     spiChipSelect(ADC);
350     adc_write_read(0x10);
351     adc_write_read((unsigned char)(value>>8));
352     adc_write_read((unsigned char)(value&0xff));
353     spiChipSelect(NONE);
354 }
355
356 /*
357 * Schreiben des Mode Registers.
358 * Das Register ist 2 Bytes groß, es muss daher ein
359 * zwei Byte short Wert übergeben werden. Es erfolgt
360 * keine Prüfung.
361  */
362 void adc_set_mode_reg(unsigned short value)
363 {
364     spiChipSelect(ADC);
365     adc_write_read(0x08);
366     adc_write_read((unsigned char)(value>>8));
367     adc_write_read((unsigned char)(value&0xff));
368     spiChipSelect(NONE);
369 }
```

myadc.c

```
370
371 /*
372  * Setzen des Fullscale Registers des ADC.
373  * Es erfolgt keine Prüfung!
374  */
375 void adc_set_fs_reg(unsigned short value)
376 {
377     spiChipSelect(ADC);
378     adc_write_read(0x38);
379     adc_write_read((unsigned char)(value>>8));
380     adc_write_read((unsigned char)(value));
381     spiChipSelect(NONE);
382 }
383
384 /*
385  * Setzen des Offsetregisters des ADC.
386  * Es erfolgt keine Prüfung!
387  */
388 void adc_set_offset_reg(unsigned short value)
389 {
390     spiChipSelect(ADC);
391     adc_write_read(0x30);
392     adc_write_read((unsigned char)(value>>8));
393     adc_write_read((unsigned char)(value));
394     spiChipSelect(NONE);
395 }
396
397 /*
398  * Auslesen der Daten vom ADC.
399  * Es werden hierfür zwei Bytes gelesen und anschließend
400  * zusammengesetzt. Diese Funktion wird nur intern
401  * verwendet!
402  */
403 unsigned short adc_read_data(void)
404 {
405     unsigned short data;
406     spiChipSelect(ADC);
407     adc_write_read(0x58);
408     data = (adc_write_read(0x00))<<8;
409     data |= adc_write_read(0x00);
410     spiChipSelect(NONE);
411     return data;
412 }
413
```

mycmdline.h

```
 1 /*
 2  * myadc.h
 3  *
 4  *  Created on: 20.02.2013
 5  *      Author: Thomas W./Stellarisware
 6  */
 7
 8
 9
10
11 //*****************************************************************************
12 //
13 // cmdline.h - Prototypes for command line processing functions.
14 //
15 // Copyright (c) 2007-2012 Texas Instruments Incorporated.  All rights reserved.
16 // Software License Agreement
17 //
18 // Texas Instruments (TI) is supplying this software for use solely and
19 // exclusively on TI's microcontroller products. The software is owned by
20 // TI and/or its suppliers, and is protected under applicable copyright
21 // laws. You may not combine this software with "viral" open-source
22 // software in order to form a larger program.
23 //
24 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
25 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
26 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
27 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
28 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
29 // DAMAGES, FOR ANY REASON WHATSOEVER.
30 //
31 // This is part of revision 8555 of the Stellaris Firmware Development Package.
32 //
33 //*****************************************************************************
34
35 #ifndef __CMDLINE_H__
36 #define __CMDLINE_H__
37
38 //*****************************************************************************
39 //
40 // If building with a C++ compiler, make all of the definitions in this header
41 // have a C binding.
42 //
43 //*****************************************************************************
44 #ifdef __cplusplus
45 extern "C"
46 {
47 #endif
48
49 //*****************************************************************************
50 //
51 //! \addtogroup cmdline_api
52 //! @{
53 //
54 //*****************************************************************************
55
56 //*****************************************************************************
57 //
58 //! Defines the value that is returned if the command is not found.
59 //
60 //*****************************************************************************
61 #define CMDLINE_BAD_CMD         (-1)
62
```

mycmdline.h

```
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 //*****************************************************************************
```

mycmdline.h

```
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
```

mycmdline.c

```
 1 /*
 2 Project:                          battery_cycling_sw_v3
 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
```

mycmdline.c

```
 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)
```

mycmdline.c

```
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                 // As long as the maximum number of arguments has not been
169                 // reached, then save the pointer to the start of this new arg
170                 // in the argv array, and increment the count of args, argc.
171                 //
172                 if(argc < CMDLINE_MAX_ARGS)
173                 {
174                     argv[argc] = pcChar;
175                     argc++;
176                     bFindArg = 0;
177                 }
178 
179                 //
180                 // The maximum number of arguments has been reached so return
181                 // the error.
182                 //
183                 else
184                 {
185                     return(CMDLINE_TOO_MANY_ARGS);
186                 }
```

mycmdline.c

```
187             }
188         }
189 
190         //
191         // Advance to the next character in the command line.
192         //
193         pcChar++;
194     }
195 
196     //
197     // If one or more arguments was found, then process the command.
198     //
199     if(argc)
200     {
201         //
202         // Start at the beginning of the command table, to look for a matching
203         // command.
204         //
205         pCmdEntry = g_psCmdTable;
206 
207         //
208         // Search through the command table until a null command string is
209         // found, which marks the end of the table.
210         //
211         while(pCmdEntry->pcCmd)
212         {
213             //
214             // If this command entry command string matches argv[0], then call
215             // the function for this command, passing the command line
216             // arguments.
217             //
218             if(!strcmp(argv[0], pCmdEntry->pcCmd))
219             {
220                 return(pCmdEntry->pfnCmd(argc, argv));
221             }
222 
223             //
224             // Not found, so advance to the next entry.
225             //
226             pCmdEntry++;
227         }
228     }
229 
230     //
231     // Fall through to here means that no matching command was found, so return
232     // an error.
233     //
234     return(CMDLINE_BAD_CMD);
235 }
236 
237 //*****************************************************************************
238 //
239 // Close the Doxygen group.
240 //! @}
241 //
242 //*****************************************************************************
243 
244 
245 
246 
247 //*****************************************************************************
248 //
```

mycmdline.c

```
249 // Function to interprete command. Check if command is valid and execute.
250 //
251 //*****************************************************************************
252 void Cmd_interprete(char *g_cCmdBuf){
253
254     int nStatus;
255
256     //
257     // Echo received command
258     //
259     UARTprintf("%s", g_cCmdBuf);
260     sprintf(g_cAnsBuf, "%s\n", g_cCmdBuf);
261     telnet_write(g_cAnsBuf);
262
263     //
264     // Pass the line from the user to the command processor.
265     // It will be parsed and valid commands executed.
266     //
267     nStatus = CmdLineProcess(g_cCmdBuf);
268
269     //
270     // Handle the case of bad command.
271     //
272     if(nStatus == CMDLINE_BAD_CMD)
273     {
274         UARTprintf("\n Bad command!");
275         if(control_connection){
276         sprintf(g_cAnsBuf, " Bad command!\n");
277         telnet_write(g_cAnsBuf);
278         }
279     }
280
281     //
282     // Handle the case of too many arguments.
283     //
284     else if(nStatus == CMDLINE_TOO_MANY_ARGS)
285     {
286         UARTprintf("\n Too many arguments for command processor!");
287         if(control_connection){
288         sprintf(g_cAnsBuf,  " Too many arguments for command processor!\n");
289         telnet_write(g_cAnsBuf);
290         }
291     }
292
293     if(control_connection){
294     telnet_write(" Enter Command (type <help> to see list of commands):\n");
295     sprintf(g_cAnsBuf, "%s: > \n", g_cLocalBuf);
296     telnet_write(g_cAnsBuf);
297     }
298
299     //
300     // Print a prompt to the console.
301     //
302     UARTprintf("\n Enter Command (type <help> to see list of commands):");
303     UARTprintf("\n%s: > ", g_cLocalBuf);
304
305
306 }
307
308 //*****************************************************************************
309 //
310 // Command to check whetet Cycling mode is current active. Returns "YES" or "NO"
```

mycmdline.c

```
311 //
312 //*****************************************************************************
313 int
314 Cmd_alive(int argc, char *argv[])
315 {
316     if(cycle_active){
317         UARTprintf("\nYES");
318         if(control_connection)
319             telnet_write("YES\n");
320         }
321     else{
322         UARTprintf("\nNO");
323         if(control_connection)
324             telnet_write("NO\n");
325     }
326     //
327     // Return success.
328     //
329     return(0);
330 }
331
332
333
```

mypwm.h

```
 1 /*
 2  * mypwm.h
 3  *
 4  *  Created on: 22.03.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef MYPWM_H_
 9 #define MYPWM_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 void pwm_init(void);
17
18 #endif /* MYPWM_H_ */
19
```

mypwm.c

```
1 /*
2 Project:                          battery_cycling_sw_v3
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
```

mypwm.c

```
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
```

mysdcard.h

```
 1 /*
 2  * mysdcard.h
 3  *
 4  *  Created on: 21.01.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef MYSDCARD_H_
 9 #define MYSDCARD_H_
10
11 /*****************************************************************************
12 *
13 * Function Declarations
14 *
15 *****************************************************************************/
16 int Cmd_cat(int argc, char *argv[]);
17 int Cmd_cd(int argc, char *argv[]);
18 int Cmd_help(int argc, char *argv[]);
19 int Cmd_ls(int argc, char *argv[]);
20 int Cmd_pwd(int argc, char *argv[]);
21 const char * StringFromFresult(FRESULT);
22 int Cmd_sdcard_exit(int argc, char *argv[]);
23
24
25 int add_to_file(const char *filename, const char *write_Buffer);
26 int delete_file(const char *filename);
27 int read_file(const char *filename);
28 int read_into_buffer(const char *filename, char *buffer);
29 int write_to_file(const char *filename, const char *write_Buffer);
30 int create_file(const char *filename);
31 void do_measure();
32
33 void SystickHandler(void);
34
35
36 void mysdcardinit(void);
37 int my_start_cmd_line(int argc, char *argv[]);
38 int Cmd_del(int argc, char *argv[]);
39 int Cmd_cre(int argc, char *argv[]);
40
41
42 #endif /* MYSDCARD_H_ */
43
```

mysdcard.c

```
 1 /*
 2 Project:                          battery_cycling_sw_v3
 3 File:                             mysdcard.c
 4
 5 Auhtor:                           Thomas Wisniewski
 6 Credits:                          Matthias Schneider
 7                                   Stellaris Ware
 8
 9 last modified:                    2013/03/22
10
11 Project Status                    Under Construction
12 Status:                           running
13
14 CCS:                              5.5.1.00031
15 Stellarisware:                    8555
16
17 Hardware:                         Stellaris EKS-LM3S9B92 on Extension Board with
18                                   16 Bit ADC AD7798, SD-Card, Reed-Relais Matrix,
19                                   NTC-Connectors, MAX3232 and Suplly Circuits
20
21 Description:                      Source Code for SDCard at SPI used with FatFs
22
23                                   Mainly copied from ti stellarisware example:
24                                   stellarisware/boards/ek-lm329b96/sdcard
25 */
26
27 /*****************************************************************************
28 *
29 * Original Stellaris Ware Includings
30 *
31 *****************************************************************************/
32
33 #include <string.h>
34 #include "inc/hw_memmap.h"
35 #include "inc/hw_types.h"
36 #include "inc/hw_ssi.h"
37 #include <inc/lm3s9b92.h>
38 #include "driverlib/gpio.h"
39 #include "driverlib/interrupt.h"
40 #include "driverlib/ssi.h"
41 #include "driverlib/sysctl.h"
42 #include "driverlib/systick.h"
43 #include <driverlib/gpio.h>
44 #include "driverlib/rom.h"
45 #include "header/mycmdline.h"
46 #include "utils/uartstdio.h"
47 #include "utils/ustdlib.h"
48 #include "third_party/fatfs/src/ff.h"
49 #include "third_party/fatfs/src/diskio.h"
50
51 /*****************************************************************************
52 *
53 * Own Includings
54 *
55 *****************************************************************************/
56
57 #include "header/mysdcard.h"
58 #include "header/myadc.h"
59 #include "header/temperature.h"
60 #include "header/clocktimer.h"
61 #include "header/config.h"
62 #include "header/control.h"
```

mysdcard.c

```
 63 #include "header/display.h"
 64 #include "stdio.h"
 65 #include "utils/lwiplib.h"
 66 #include "header/myssi.h"
 67
 68
 69 //*****************************************************************************
 70 //Extern variables for Clock
 71 //*****************************************************************************
 72 extern volatile unsigned long clock_msec;
 73 extern volatile unsigned long clock_sec;
 74 extern volatile unsigned long clock_min;
 75 extern volatile unsigned long clock_hour;
 76 extern volatile unsigned long clock_day;
 77 extern volatile unsigned long clock_month;
 78 extern volatile unsigned long clock_year;
 79
 80 //*****************************************************************************
 81 //Extern variable for current active power relay
 82 //*****************************************************************************
 83 extern volatile unsigned int power_relay_active;
 84
 85 //*****************************************************************************
 86 //Extern variable for current
 87 //*****************************************************************************
 88 extern volatile unsigned int g_iCurrent;
 89
 90 //*****************************************************************************
 91 //Variable for filename for measurment saving
 92 //*****************************************************************************
 93 char MEAS_FILE[64];
 94
 95 //*****************************************************************************
 96 //
 97 // Defines the size of the buffers that hold the path, or temporary
 98 // data from the SD card.  There are two buffers allocated of this size.
 99 // The buffer size must be large enough to hold the longest expected
100 // full path name, including the file name, and a trailing null character.
101 //
102 //*****************************************************************************
103 #define PATH_BUF_SIZE   80
104
105
106
107 //*****************************************************************************
108 //
109 // This buffer holds the full path to the current working directory.
110 // Initially it is root ("/").
111 //
112 //*****************************************************************************
113 static char g_cCwdBuf[PATH_BUF_SIZE] = "/";
114
115 //*****************************************************************************
116 //
117 // A temporary data buffer used when manipulating file paths, or reading data
118 // from the SD card.
119 //
120 //*****************************************************************************
121 static char g_cTmpBuf[PATH_BUF_SIZE];
122
123
124 //*****************************************************************************
```

mysdcard.c

```
125 //
126 // The following are data structures used by FatFs.
127 //
128 //*****************************************************************************
129 static FATFS g_sFatFs;
130 static DIR g_sDirObject;
131 static FILINFO g_sFileInfo;
132 static FIL g_sFileObject;
133
134
135 //*****************************************************************************
136 //
137 // A structure that holds a mapping between an FRESULT numerical code,
138 // and a string represenation.  FRESULT codes are returned from the FatFs
139 // FAT file system driver.
140 //
141 //*****************************************************************************
142 typedef struct
143 {
144     FRESULT fresult;
145     char *pcResultStr;
146 }
147 tFresultString;
148
149 //*****************************************************************************
150 //
151 // A macro to make it easy to add result codes to the table.
152 //
153 //*****************************************************************************
154 #define FRESULT_ENTRY(f)        { (f), (#f) }
155
156 //*****************************************************************************
157 //
158 // A table that holds a mapping between the numerical FRESULT code and
159 // it's name as a string.  This is used for looking up error codes for
160 // printing to the console.
161 //
162 //*****************************************************************************
163 tFresultString g_sFresultStrings[] =
164 {
165     FRESULT_ENTRY(FR_OK),
166     FRESULT_ENTRY(FR_NOT_READY),
167     FRESULT_ENTRY(FR_NO_FILE),
168     FRESULT_ENTRY(FR_NO_PATH),
169     FRESULT_ENTRY(FR_INVALID_NAME),
170     FRESULT_ENTRY(FR_INVALID_DRIVE),
171     FRESULT_ENTRY(FR_DENIED),
172     FRESULT_ENTRY(FR_EXIST),
173     FRESULT_ENTRY(FR_RW_ERROR),
174     FRESULT_ENTRY(FR_WRITE_PROTECTED),
175     FRESULT_ENTRY(FR_NOT_ENABLED),
176     FRESULT_ENTRY(FR_NO_FILESYSTEM),
177     FRESULT_ENTRY(FR_INVALID_OBJECT),
178     FRESULT_ENTRY(FR_MKFS_ABORTED)
179 };
180
181 //*****************************************************************************
182 //
183 // A macro that holds the number of result codes.
184 //
185 //*****************************************************************************
186 #define NUM_FRESULT_CODES (sizeof(g_sFresultStrings) / sizeof(tFresultString))
```

mysdcard.c

```
187
188 //*****************************************************************************
189 //
190 // This function returns a string representation of an error code
191 // that was returned from a function call to FatFs.  It can be used
192 // for printing human readable error messages.
193 //
194 //*****************************************************************************
195 const char *
196 StringFromFresult(FRESULT fresult)
197 {
198     unsigned int uIdx;
199
200     //
201     // Enter a loop to search the error code table for a matching
202     // error code.
203     //
204     for(uIdx = 0; uIdx < NUM_FRESULT_CODES; uIdx++)
205     {
206         //
207         // If a match is found, then return the string name of the
208         // error code.
209         //
210         if(g_sFresultStrings[uIdx].fresult == fresult)
211         {
212             return(g_sFresultStrings[uIdx].pcResultStr);
213         }
214     }
215
216     //
217     // At this point no matching code was found, so return a
218     // string indicating unknown error.
219     //
220     return("UNKNOWN ERROR CODE");
221 }
222
223
224 //*****************************************************************************
225 //
226 // This is the table that holds the command names, implementing functions,
227 // and brief description.
228 //
229 //*****************************************************************************
230 tCmdLineEntry g_sBrowserCmdTable[] =
231 {
232     { "help",   Cmd_help,      " : Display list of commands" },
233     { "h",      Cmd_help,    "    : alias for help" },
234     { "?",      Cmd_help,    "    : alias for help" },
235     { "ls",     Cmd_ls,       "   : Display list of files" },
236     { "chdir",  Cmd_cd,         ": Change directory" },
237     { "cd",     Cmd_cd,       "   : alias for chdir" },
238     { "pwd",    Cmd_pwd,      "   : Show current working directory" },
239     { "cat",    Cmd_cat,      "   : Show contents of a text file (use only in Idle Mode)"
    },
240     { "cre",    Cmd_cre,      "   : Create a file" },
241     { "del",    Cmd_del,      "   : Delete a file" },
242     { "exit",   Cmd_sdcard_exit,      " : Exit SD-Card browser" },
243     { 0, 0, 0 }
244 };
245
246 extern tCmdLineEntry g_sMainCmdTable;
247 extern tCmdLineEntry *g_psCmdTable;
```

mysdcard.c

```
248
249 // string for main location
250 extern const char *g_cMainLocalBuf;
251
252 // Global location buffer
253 extern char *g_cLocalBuf;
254
255 // TCP-Socket für die aktuelle Steuerungs-Verbindung
256 extern struct tcp_pcb* control_connection;
257
258 // String buffer for print operations to UART and Ethernet
259 char print_buffer[80];
260
261 //define string for SD-Card location
262 char *g_cSdLocalBuf  = "SD-Card";
263
264
265 //*****************************************************************************
266 //
267 // This is the handler for this SysTick interrupt.  FatFs requires a
268 // timer tick every 10 ms for internal timing purposes.
269 //
270 //*****************************************************************************
271 void
272 SysTickHandler(void)
273 {
274     //
275     // Call the FatFs tick timer.
276     //
277     TimerClock();
278     disk_timerproc();
279     lwIPTimer(1);
280
281 }
282
283 //*****************************************************************************
284 //
285 // This function implements the "ls" command.  It opens the current
286 // directory and enumerates through the contents, and prints a line for
287 // each item it finds.  It shows details such as file attributes, time and
288 // date, and the file size, along with the name.  It shows a summary of
289 // file sizes at the end along with free space.
290 //
291 //*****************************************************************************
292 int
293 Cmd_ls(int argc, char *argv[])
294 {
295
296     unsigned long ulTotalSize;
297     unsigned long ulFileCount;
298     unsigned long ulDirCount;
299     FRESULT fresult;
300     FATFS *pFatFs;
301
302     //
303     // Open the current directory for access.
304     //
305     fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
306
307     //
308     // Check for error and return if there is a problem.
309     //
```

mysdcard.c

```
310     if(fresult != FR_OK)
311     {
312         return(fresult);
313     }
314
315     ulTotalSize = 0;
316     ulFileCount = 0;
317     ulDirCount = 0;
318
319     //
320     // Give an extra blank line before the listing.
321     //
322     UARTprintf("\n");
323
324     //
325     // Enter loop to enumerate through all directory entries.
326     //
327     for(;;)
328     {
329         //
330         // Read an entry from the directory.
331         //
332         fresult = f_readdir(&g_sDirObject, &g_sFileInfo);
333
334         //
335         // Check for error and return if there is a problem.
336         //
337         if(fresult != FR_OK)
338         {
339             sprintf(print_buffer, "Error at reading File System Entry ");
340
341             UARTprintf(print_buffer);               // debug information
342             if(control_connection){
343                 telnet_write(print_buffer);
344             }
345             return(fresult);
346         }
347
348         //
349         // If the file name is blank, then this is the end of the
350         // listing.
351         //
352         if(!g_sFileInfo.fname[0])
353         {
354             break;
355         }
356
357         //
358         // If the attribue is directory, then increment the directory count.
359         //
360         if(g_sFileInfo.fattrib & AM_DIR)
361         {
362             ulDirCount++;
363         }
364
365         //
366         // Otherwise, it is a file.  Increment the file count, and
367         // add in the file size to the total.
368         //
369         else
370         {
371             ulFileCount++;
```

mysdcard.c

```
372             ulTotalSize += g_sFileInfo.fsize;
373         }
374 
375         //
376         // Print the entry information on a single line with formatting
377         // to show the attributes, date, time, size, and name.
378         //
379         sprintf(print_buffer, "%c%c%c%c%c %u/%02u/%02u %02u:%02u %9u  %s\n",
380                               (g_sFileInfo.fattrib & AM_DIR) ? 'D' : '-',
381                               (g_sFileInfo.fattrib & AM_RDO) ? 'R' : '-',
382                               (g_sFileInfo.fattrib & AM_HID) ? 'H' : '-',
383                               (g_sFileInfo.fattrib & AM_SYS) ? 'S' : '-',
384                               (g_sFileInfo.fattrib & AM_ARC) ? 'A' : '-',
385                               (g_sFileInfo.fdate >> 9) + 1980,
386                               (g_sFileInfo.fdate >> 5) & 15,
387                                g_sFileInfo.fdate & 31,
388                               (g_sFileInfo.ftime >> 11),
389                               (g_sFileInfo.ftime >> 5) & 63,
390                                g_sFileInfo.fsize,
391                                g_sFileInfo.fname);
392 
393         UARTprintf(print_buffer);
394         if(control_connection){
395             telnet_write(print_buffer);
396         }
397     }   // endfor
398 
399     //
400     // Print summary lines showing the file, dir, and size totals.
401     //
402     sprintf(print_buffer, "\n%4u File(s),%10u bytes total\n%4u Dir(s)",
403             ulFileCount, ulTotalSize, ulDirCount);
404     UARTprintf(print_buffer);
405     if(control_connection){
406         telnet_write(print_buffer);
407     }
408     //
409     // Get the free space.
410     //
411     fresult = f_getfree("/", &ulTotalSize, &pFatFs);
412 
413     //
414     // Check for error and return if there is a problem.
415     //
416     if(fresult != FR_OK)
417     {
418         //UARTprintf(" Error at the end of the line ");     // debug information
419         return(fresult);
420     }
421 
422     //
423     // Display the amount of free space that was calculated.
424     //
425     sprintf(print_buffer, ", %10uK bytes free\n", ulTotalSize * pFatFs->sects_clust / 2);
426 
427     UARTprintf(print_buffer);
428     if(control_connection){
429         telnet_write(print_buffer);
430     }
431 
432 
433     //
```

mysdcard.c

```
434     // Made it to here, return with no errors.
435     //
436     return(0);
437 }
438
439 //*****************************************************************************
440 //
441 // This function implements the "cd" command.  It takes an argument
442 // that specifes the directory to make the current working directory.
443 // Path separators must use a forward slash "/".  The argument to cd
444 // can be one of the following:
445 // * root ("/")
446 // * a fully specified path ("/my/path/to/mydir")
447 // * a single directory name that is in the current directory ("mydir")
448 // * parent directory ("..")
449 //
450 // It does not understand relative paths, so dont try something like this:
451 // ("../my/new/path")
452 //
453 // Once the new directory is specified, it attempts to open the directory
454 // to make sure it exists.  If the new path is opened successfully, then
455 // the current working directory (cwd) is changed to the new path.
456 //
457 //*****************************************************************************
458 int
459 Cmd_cd(int argc, char *argv[])
460 {
461     unsigned int uIdx;
462     FRESULT fresult;
463
464     //
465     // Copy the current working path into a temporary buffer so
466     // it can be manipulated.
467     //
468     strcpy(g_cTmpBuf, g_cCwdBuf);
469
470     //
471     // If the first character is /, then this is a fully specified
472     // path, and it should just be used as-is.
473     //
474     if(argv[1][0] == '/')
475     {
476         //
477         // Make sure the new path is not bigger than the cwd buffer.
478         //
479         if(strlen(argv[1]) + 1 > sizeof(g_cCwdBuf))
480         {
481             sprintf(print_buffer, "Resulting path name is too long\n");
482
483             UARTprintf(print_buffer);
484             if(control_connection){
485                 telnet_write(print_buffer);
486             }
487             return(0);
488         }
489
490         //
491         // If the new path name (in argv[1])  is not too long, then
492         // copy it into the temporary buffer so it can be checked.
493         //
494         else
495         {
```

mysdcard.c

```
496             strncpy(g_cTmpBuf, argv[1], sizeof(g_cTmpBuf));
497         }
498     }
499
500     //
501     // If the argument is .. then attempt to remove the lowest level
502     // on the CWD.
503     //
504     else if(!strcmp(argv[1], ".."))
505     {
506         //
507         // Get the index to the last character in the current path.
508         //
509         uIdx = strlen(g_cTmpBuf) - 1;
510
511         //
512         // Back up from the end of the path name until a separator (/)
513         // is found, or until we bump up to the start of the path.
514         //
515         while((g_cTmpBuf[uIdx] != '/') && (uIdx > 1))
516         {
517             //
518             // Back up one character.
519             //
520             uIdx--;
521         }
522
523         //
524         // Now we are either at the lowest level separator in the
525         // current path, or at the beginning of the string (root).
526         // So set the new end of string here, effectively removing
527         // that last part of the path.
528         //
529         g_cTmpBuf[uIdx] = 0;
530     }
531
532     //
533     // Otherwise this is just a normal path name from the current
534     // directory, and it needs to be appended to the current path.
535     //
536     else
537     {
538         //
539         // Test to make sure that when the new additional path is
540         // added on to the current path, there is room in the buffer
541         // for the full new path.  It needs to include a new separator,
542         // and a trailing null character.
543         //
544         if(strlen(g_cTmpBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cCwdBuf))
545         {
546             sprintf(print_buffer, "Resulting path name is too long\n");
547
548             UARTprintf(print_buffer);
549
550             if(control_connection){
551                 telnet_write(print_buffer);
552             }
553             return(0);
554         }
555
556         //
557         // The new path is okay, so add the separator and then append
```

mysdcard.c

```
558         // the new directory to the path.
559         //
560         else
561         {
562             //
563             // If not already at the root level, then append a /
564             //
565             if(strcmp(g_cTmpBuf, "/"))
566             {
567                 strcat(g_cTmpBuf, "/");
568             }
569
570             //
571             // Append the new directory to the path.
572             //
573             strcat(g_cTmpBuf, argv[1]);
574         }
575     }
576
577     //
578     // At this point, a candidate new directory path is in chTmpBuf.
579     // Try to open it to make sure it is valid.
580     //
581     fresult = f_opendir(&g_sDirObject, g_cTmpBuf);
582
583     //
584     // If it cant be opened, then it is a bad path.  Inform
585     // user and return.
586     //
587     if(fresult != FR_OK)
588     {
589         sprintf(print_buffer, "cd: %s\n", g_cTmpBuf);
590
591         UARTprintf(print_buffer);
592
593         if(control_connection){
594             telnet_write(print_buffer);
595         }
596         return(fresult);
597     }
598
599     //
600     // Otherwise, it is a valid new path, so copy it into the CWD.
601     //
602     else
603     {
604         strncpy(g_cCwdBuf, g_cTmpBuf, sizeof(g_cCwdBuf));
605     }
606
607
608     //
609     // Return success.
610     //
611     return(0);
612 }
613
614 //*****************************************************************************
615 //
616 // This function implements the "pwd" command.  It simply prints the
617 // current working directory.
618 //
619 //*****************************************************************************
```

mysdcard.c

```
620 int
621 Cmd_pwd(int argc, char *argv[])
622 {
623
624     //
625     // Print the CWD to the console.
626     //
627     sprintf(print_buffer, "%s\n", g_cCwdBuf);
628
629     UARTprintf(print_buffer);
630
631     if(control_connection){
632         telnet_write(print_buffer);
633     }
634
635
636     //
637     // Return success.
638     //
639     return(0);
640 }
641
642 //*****************************************************************************
643 //
644 // This function implements the "cat" command.  It reads the contents of
645 // a file and prints it to the console.  This should only be used on
646 // text files.  If it is used on a binary file, then a bunch of garbage
647 // is likely to printed on the console.
648 //
649 //*****************************************************************************
650 int
651 Cmd_cat(int argc, char *argv[])
652 {
653
654     FRESULT fresult;
655     unsigned short usBytesRead;
656
657     //
658     // First, check to make sure that the current path (CWD), plus
659     // the file name, plus a separator and trailing null, will all
660     // fit in the temporary buffer that will be used to hold the
661     // file name.  The file name must be fully specified, with path,
662     // to FatFs.
663     //
664     if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
665     {
666         sprintf(print_buffer, "Resulting path name is too long\n");
667
668         UARTprintf(print_buffer);
669
670         if(control_connection){
671             telnet_write(print_buffer);
672         }
673         return(0);
674     }
675
676     //
677     // Copy the current path to the temporary buffer so it can be manipulated.
678     //
679     strcpy(g_cTmpBuf, g_cCwdBuf);
680
681     //
```

mysdcard.c

```
682     // If not already at the root level, then append a separator.
683     //
684     if(strcmp("/", g_cCwdBuf))
685     {
686         strcat(g_cTmpBuf, "/");
687     }
688
689     //
690     // Now finally, append the file name to result in a fully specified file.
691     //
692     strcat(g_cTmpBuf, argv[1]);
693
694     //
695     // Open the file for reading.
696     //
697     fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
698
699     //
700     // If there was some problem opening the file, then return
701     // an error.
702     //
703     if(fresult != FR_OK)
704     {
705         return(fresult);
706     }
707
708
709     //
710     // Enter a loop to repeatedly read data from the file and display it,
711     // until the end of the file is reached.
712     //
713     do
714     {
715         //
716         // Read a block of data from the file.  Read as much as can fit
717         // in the temporary buffer, including a space for the trailing null.
718         //
719         fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf) - 1,
720                          &usBytesRead);
721
722         //
723         // If there was an error reading, then print a newline and
724         // return the error to the user.
725         //
726         if(fresult != FR_OK)
727         {
728             sprintf(print_buffer, "\n");
729             UARTprintf(print_buffer);
730             if(control_connection){
731                 telnet_write(print_buffer);
732             }
733             return(fresult);
734         }
735
736         //
737         // Null terminate the last block that was read to make it a
738         // null terminated string that can be used with printf.
739         //
740         g_cTmpBuf[usBytesRead] = 0;
741
742         //
743         // Print the last chunk of the file that was received.
```

mysdcard.c

```
744         //
745         sprintf(print_buffer, "%s", g_cTmpBuf);
746 
747         UARTprintf(print_buffer);
748 
749         if(control_connection){
750             telnet_write(print_buffer);
751         }
752 
753     //
754     // Continue reading until less than the full number of bytes are
755     // read.  That means the end of the buffer was reached.
756     //
757     }
758     while(usBytesRead == sizeof(g_cTmpBuf) - 1);
759 
760 
761     //
762     // Return success.
763     //
764     return(0);
765 }
766 
767 //*****************************************************************************
768 //
769 // This function implements the "del" command.  It deletes a file
770 //
771 //*****************************************************************************
772 int
773 Cmd_del(int argc, char *argv[])
774 {
775 
776     FRESULT fresult;
777 
778     //
779     // First, check to make sure that the current path (CWD), plus
780     // the file name, plus a separator and trailing null, will all
781     // fit in the temporary buffer that will be used to hold the
782     // file name.  The file name must be fully specified, with path,
783     // to FatFs.
784     //
785     if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
786     {
787         sprintf(print_buffer, "Resulting path name is too long\n");
788 
789         UARTprintf(print_buffer);
790 
791         if(control_connection){
792             telnet_write(print_buffer);
793         }
794         return(0);
795     }
796 
797     //
798     // Copy the current path to the temporary buffer so it can be manipulated.
799     //
800     strcpy(g_cTmpBuf, g_cCwdBuf);
801 
802     //
803     // If not already at the root level, then append a separator.
804     //
805     if(strcmp("/", g_cCwdBuf))
```

mysdcard.c

```
806     {
807         strcat(g_cTmpBuf, "/");
808     }
809 
810     //
811     // Now finally, append the file name to result in a fully specified file.
812     //
813     strcat(g_cTmpBuf, argv[1]);
814 
815     //
816         // Delete File
817         //
818           fresult = f_unlink(g_cTmpBuf);
819 
820           //
821           // Check if operation succeeded
822           //
823 
824           if(fresult != FR_OK)
825           {
826               return(fresult);
827           }
828 
829 
830 
831     //
832     // Return success.
833     //
834     return(0);
835 
836 }
837 
838 //*****************************************************************************
839 //
840 // This function implements the "cre" command.  It creates a file
841 //
842 //*****************************************************************************
843 int
844 Cmd_cre(int argc, char *argv[])
845 {
846 
847     FRESULT fresult;
848     FIL fnew;       /* new file object */
849 
850     //
851     // First, check to make sure that the current path (CWD), plus
852     // the file name, plus a separator and trailing null, will all
853     // fit in the temporary buffer that will be used to hold the
854     // file name.  The file name must be fully specified, with path,
855     // to FatFs.
856     //
857     if(strlen(g_cCwdBuf) + strlen(argv[1]) + 1 + 1 > sizeof(g_cTmpBuf))
858     {
859         sprintf(print_buffer, "Resulting path name is too long\n");
860 
861         UARTprintf(print_buffer);
862 
863         if(control_connection){
864             telnet_write(print_buffer);
865         }
866         return(0);
867     }
```

mysdcard.c

```
868
869     //
870     // Copy the current path to the temporary buffer so it can be manipulated.
871     //
872     strcpy(g_cTmpBuf, g_cCwdBuf);
873
874     //
875     // If not already at the root level, then append a separator.
876     //
877     if(strcmp("/", g_cCwdBuf))
878     {
879         strcat(g_cTmpBuf, "/");
880     }
881
882     //
883     // Now finally, append the file name to result in a fully specified file.
884     //
885     strcat(g_cTmpBuf, argv[1]);
886
887     //
888     // Create the File
889     //
890     fresult = f_open(&fnew, argv[1], FA_CREATE_ALWAYS | FA_WRITE );
891
892     //
893     // Check if creation succeeded
894     //
895
896     if(fresult != FR_OK)
897     {
898         return(fresult);
899     }
900
901     /* Close opened files */
902     f_close(&fnew);
903
904
905     //
906     // Return success.
907     //
908     return(0);
909
910 }
911
912 //*****************************************************************************
913 //
914 // This function implements the "help" command.  It prints a simple list
915 // of the available commands with a brief description.
916 //
917 //*****************************************************************************
918 int
919 Cmd_help(int argc, char *argv[])
920 {
921
922     tCmdLineEntry *pEntry;
923
924     //
925     // Print some header text.
926     //
927
928     UARTprintf("\nAvailable commands\n");
929     UARTprintf("------------------\n");
```

mysdcard.c

```
930
931     if(control_connection){
932         telnet_write("Available commands\n");
933         telnet_write("------------------\n");
934     }
935
936     //
937     // Point at the beginning of the command table.
938     //
939     pEntry = g_psCmdTable;
940
941     //
942     // Enter a loop to read each entry from the command table.  The
943     // end of the table has been reached when the command name is NULL.
944     //
945     while(pEntry->pcCmd)
946     {
947         //
948         // Print the command name and the brief description.
949         //
950         sprintf(print_buffer, "%s%s\n", pEntry->pcCmd, pEntry->pcHelp);
951
952         UARTprintf(print_buffer);
953         if(control_connection){
954             telnet_write(print_buffer);
955         }
956
957         //
958         // Advance to the next entry in the table.
959         //
960         pEntry++;
961     }
962
963
964     //
965     // Return success.
966     //
967     return(0);
968 }
969
970 int Cmd_sdcard_exit(int argc, char *argv[])
971 {
972
973     // set Command line pointer to the beginning of the main command structure
974     g_psCmdTable = &g_sMainCmdTable;
975
976     // set location buffer to main
977     g_cLocalBuf = (char*)g_cMainLocalBuf;
978
979     return(0);
980 }
981
982 //*****************************************************************************
983 //
984 // This is the table that holds the command names, implementing functions,
985 // and brief description.
986 //
987 //*****************************************************************************
988 //extern tCmdLineEntry g_sCmdTable[];
989
990
991 //*****************************************************************************
```

mysdcard.c

```
 992 //
 993 // Own Initialization of SSI0, SysTick, FatFs and SDcard
 994 //
 995 //*****************************************************************************
 996
 997 void mysdcardinit(void)
 998 {
 999
1000     // variable for FatFs results
1001     FRESULT fresult = FR_NOT_READY;
1002
1003     UARTprintf("SD card initialization...");
1004
1005     // Configure SysTick for a 100Hz interrupt.  The FatFs driver
1006     // wants a 10 ms tick.
1007
1008     SysTickPeriodSet(SysCtlClockGet() / 100);
1009     SysTickEnable();
1010     SysTickIntEnable();
1011
1012     fresult = f_mount(0, &g_sFatFs);
1013
1014     if(fresult != FR_OK)
1015     {
1016         UARTprintf("f_mount error: %s\n", StringFromFresult(fresult));
1017     }
1018     // debugging stuff
1019     //else
1020     //{
1021     //    UARTprintf(" f_mount successful\n");
1022     //}
1023
1024     // reset status flag
1025     fresult = FR_NOT_READY;
1026
1027     // Open the current directory for access.
1028     fresult = f_opendir(&g_sDirObject, g_cCwdBuf);
1029
1030     // Check for error and return if there is a problem.
1031     if(fresult != FR_OK)
1032     {
1033         UARTprintf(" f_opendir error: %s\n", StringFromFresult(fresult));
1034     }
1035     else
1036     {
1037         UARTprintf("done.\n");
1038     }
1039     // debugging stuff
1040     //else
1041     //{
1042     //    UARTprintf(" f_opendir successful\n");
1043     //}
1044 }
1045
1046
1047
1048
1049
1050 //*****************************************************************************
1051 //
1052 // This function implements the "read_File" command.  It reads the contents of
1053 // a file and prints it to the console.  This should only be used on
```

mysdcard.c

```
1054 // text files.  If it is used on a binary file, then a bunch of garbage
1055 // is likely to printed on the console.
1056 //
1057 //*****************************************************************************
1058 int read_file(const char *filename)
1059 {
1060
1061     FRESULT fresult;
1062     unsigned short usBytesRead;
1063
1064     //
1065     // First, check to make sure that the current path (CWD), plus
1066     // the file name, plus a separator and trailing null, will all
1067     // fit in the temporary buffer that will be used to hold the
1068     // file name.  The file name must be fully specified, with path,
1069     // to FatFs.
1070     //
1071     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1072     {
1073         UARTprintf("Resulting path name is too long\n");
1074         return(0);
1075     }
1076
1077     //
1078     // Copy the current path to the temporary buffer so it can be manipulated.
1079     //
1080     strcpy(g_cTmpBuf, g_cCwdBuf);
1081
1082     //
1083     // If not already at the root level, then append a separator.
1084     //
1085     if(strcmp("/", g_cCwdBuf))
1086     {
1087         strcat(g_cTmpBuf, "/");
1088     }
1089
1090     //
1091     // Now finally, append the file name to result in a fully specified file.
1092     //
1093     strcat(g_cTmpBuf, filename);
1094
1095     //
1096     // Open the file for reading.
1097     //
1098     fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1099
1100     //
1101     // If there was some problem opening the file, then return
1102     // an error.
1103     //
1104     if(fresult != FR_OK)
1105     {
1106         return(fresult);
1107     }
1108
1109     //
1110     // Enter a loop to repeatedly read data from the file and display it,
1111     // until the end of the file is reached.
1112     //
1113     do
1114     {
1115         //
```

mysdcard.c

```
1116        // Read a block of data from the file.  Read as much as can fit
1117        // in the temporary buffer, including a space for the trailing null.
1118        //
1119        fresult = f_read(&g_sFileObject, g_cTmpBuf, sizeof(g_cTmpBuf) - 1,
1120                         &usBytesRead);
1121
1122        //
1123        // If there was an error reading, then print a newline and
1124        // return the error to the user.
1125        //
1126        if(fresult != FR_OK)
1127        {
1128            UARTprintf("\n");
1129            return(fresult);
1130        }
1131
1132        //
1133        // Null terminate the last block that was read to make it a
1134        // null terminated string that can be used with printf.
1135        //
1136        g_cTmpBuf[usBytesRead] = 0;
1137
1138        //
1139        // Print the last chunk of the file that was received.
1140        //
1141
1142        UARTprintf("%s", g_cTmpBuf);
1143
1144
1145        //
1146        // Wait for the UART transmit buffer to empty.
1147        //
1148
1149        #if defined(UART_BUFFERED)
1150            UARTFlushTx(false);
1151        #endif
1152
1153    //
1154    // Continue reading until less than the full number of bytes are
1155    // read.  That means the end of the buffer was reached.
1156    //
1157    }
1158    while(usBytesRead == sizeof(g_cTmpBuf) - 1);
1159
1160    /* Close opened files */
1161    f_close(&g_sFileObject);
1162
1163
1164    //
1165    // Return success.
1166    //
1167    return(0);
1168 }
1169
1170 //*****************************************************************************
1171 //
1172 // This function implements the "read_config" command.  It reads the contents of
1173 // a file and prints it to the console.  This should only be used on
1174 // text files.  If it is used on a binary file, then a bunch of garbage
1175 // is likely to printed on the console.
1176 //
1177 //*****************************************************************************
```

mysdcard.c

```
1178 int read_into_buffer(const char *filename, char *buffer)
1179 {
1180
1181     FRESULT fresult;
1182     unsigned short usBytesRead;
1183
1184     //
1185     // First, check to make sure that the current path (CWD), plus
1186     // the file name, plus a separator and trailing null, will all
1187     // fit in the temporary buffer that will be used to hold the
1188     // file name.  The file name must be fully specified, with path,
1189     // to FatFs.
1190     //
1191     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1192     {
1193         UARTprintf("Resulting path name is too long\n");
1194         return(0);
1195     }
1196
1197     //
1198     // Copy the current path to the temporary buffer so it can be manipulated.
1199     //
1200     strcpy(g_cTmpBuf, g_cCwdBuf);
1201
1202     //
1203     // If not already at the root level, then append a separator.
1204     //
1205     if(strcmp("/", g_cCwdBuf))
1206     {
1207         strcat(g_cTmpBuf, "/");
1208     }
1209
1210     //
1211     // Now finally, append the file name to result in a fully specified file.
1212     //
1213     strcat(g_cTmpBuf, filename);
1214
1215     //
1216     // Open the file for reading.
1217     //
1218     fresult = f_open(&g_sFileObject, g_cTmpBuf, FA_READ);
1219
1220     //
1221     // If there was some problem opening the file, then return
1222     // an error.
1223     //
1224     if(fresult != FR_OK)
1225     {
1226         return(fresult);
1227     }
1228
1229     //
1230     // Enter a loop to repeatedly read data from the file and display it,
1231     // until the end of the file is reached.
1232     //
1233     char *p = buffer;
1234     do
1235     {
1236         //
1237         // Read a block of data from the file.  Read as much as can fit
1238         // in the temporary buffer, including a space for the trailing null.
1239         //
```

mysdcard.c

```
1240        fresult = f_read(&g_sFileObject, p, sizeof(p) - 1,
1241                         &usBytesRead);
1242
1243        //
1244        // If there was an error reading, then print a newline and
1245        // return the error to the user.
1246        //
1247        if(fresult != FR_OK)
1248        {
1249            UARTprintf("\n");
1250            return(fresult);
1251        }
1252
1253        //
1254        // Null terminate the last block that was read to make it a
1255        // null terminated string that can be used with printf.
1256        //
1257        p[usBytesRead] = 0;
1258
1259        //
1260        // Print the last chunk of the file that was received.
1261        //
1262        //UARTprintf("%s", p);
1263        p += usBytesRead;
1264
1265        //
1266        // Wait for the UART transmit buffer to empty.
1267        //
1268        #if defined(UART_BUFFERED)
1269            UARTFlushTx(false);
1270        #endif
1271
1272    //
1273    // Continue reading until less than the full number of bytes are
1274    // read.  That means the end of the buffer was reached.
1275    //
1276    }
1277    while(usBytesRead == sizeof(p) - 1);
1278
1279    /* Close opened files */
1280    fresult = f_close(&g_sFileObject);
1281
1282    if(fresult != FR_OK)
1283    {
1284        return(fresult);
1285    }
1286
1287
1288    //
1289    // Return success.
1290    //
1291    return (0);
1292 }
1293
1294
1295
1296 //
1297 // This function implements the "delete_File" command.
1298 // It simply delete a file on the SD Card, selected with its filename
1299 //
1300 int delete_file(const char *filename){
1301
```

mysdcard.c

```
1302
1303        FRESULT fresult;
1304
1305        //
1306        // First, check to make sure that the current path (CWD), plus
1307        // the file name, plus a separator and trailing null, will all
1308        // fit in the temporary buffer that will be used to hold the
1309        // file name.  The file name must be fully specified, with path,
1310        // to FatFs.
1311        //
1312        if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1313        {
1314            UARTprintf("Resulting path name is too long\n");
1315            return(0);
1316        }
1317
1318        //
1319        // Copy the current path to the temporary buffer so it can be manipulated.
1320        //
1321        strcpy(g_cTmpBuf, g_cCwdBuf);
1322
1323        //
1324        // If not already at the root level, then append a separator.
1325        //
1326        if(strcmp("/", g_cCwdBuf))
1327        {
1328            strcat(g_cTmpBuf, "/");
1329        }
1330
1331        //
1332        // Now finally, append the file name to result in a fully specified file.
1333        //
1334        strcat(g_cTmpBuf, filename);
1335
1336            //
1337            // Wait for the UART transmit buffer to empty.
1338            //
1339        #if defined(UART_BUFFERED)
1340            UARTFlushTx(false);
1341        #endif
1342
1343
1344
1345    //
1346    // Delete File
1347    //
1348      fresult = f_unlink(g_cTmpBuf);
1349
1350      //
1351      // Check if operation succeeded
1352      //
1353
1354      if(fresult != FR_OK)
1355      {
1356          return(fresult);
1357      }
1358
1359
1360      //
1361      // Return success.
1362      //
1363      return(0);
```

mysdcard.c

```
1364
1365 }
1366
1367
1368 //*****************************************************************************
1369 //
1370 // This function implements the "write_to_file" command.  It overrides the content of
1371 // a file with new input.
1372 //
1373 //*****************************************************************************
1374
1375 int write_to_file(const char *filename, const char *write_Buffer){
1376
1377
1378         FIL fnew;      /* new file object */
1379         FRESULT fresult;
1380         unsigned short bw = 0;
1381
1382
1383         //
1384         // First, check to make sure that the current path (CWD), plus
1385         // the file name, plus a separator and trailing null, will all
1386         // fit in the temporary buffer that will be used to hold the
1387         // file name.  The file name must be fully specified, with path,
1388         // to FatFs.
1389         //
1390         if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1391         {
1392             UARTprintf("Resulting path name is too long\n");
1393             return(0);
1394         }
1395
1396         //
1397         // Copy the current path to the temporary buffer so it can be manipulated.
1398         //
1399         strcpy(g_cTmpBuf, g_cCwdBuf);
1400
1401         //
1402         // If not already at the root level, then append a separator.
1403         //
1404         if(strcmp("/", g_cCwdBuf))
1405         {
1406             strcat(g_cTmpBuf, "/");
1407         }
1408
1409         //
1410         // Now finally, append the file name to result in a fully specified file.
1411         //
1412         strcat(g_cTmpBuf, filename);
1413
1414             //
1415             // Wait for the UART transmit buffer to empty.
1416             //
1417             #if defined(UART_BUFFERED)
1418                 UARTFlushTx(false);
1419             #endif
1420
1421
1422     //
1423     // Create File
1424     //
1425
```

mysdcard.c

```
1426        fresult = f_open(&fnew, g_cTmpBuf, FA_CREATE_ALWAYS | FA_WRITE | FA_READ);
1427
1428
1429
1430        if(fresult != FR_OK)
1431        {
1432            return(fresult);
1433        }
1434
1435
1436
1437        fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1438
1439        if(fresult != FR_OK)
1440        {
1441            return(fresult);
1442        }
1443
1444
1445        /* Close opened files */
1446        f_close(&fnew);
1447
1448
1449        //
1450        // Return success.
1451        //
1452        return(0);
1453
1454 }
1455
1456
1457 //*****************************************************************************
1458 //
1459 // This function implements the "add_to_file" command.  It appends new input to
1460 // an existing file, selected by its filename. If file not exists, file is being created.
1461 //
1462 //*****************************************************************************
1463
1464 int add_to_file(const char *filename, const char *write_Buffer){
1465
1466
1467     FIL fnew;      /* new file object */
1468     FRESULT fresult;
1469     unsigned short bw = 0;
1470
1471
1472     //
1473     // First, check to make sure that the current path (CWD), plus
1474     // the file name, plus a separator and trailing null, will all
1475     // fit in the temporary buffer that will be used to hold the
1476     // file name.  The file name must be fully specified, with path,
1477     // to FatFs.
1478     //
1479     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1480     {
1481         UARTprintf("Resulting path name is too long\n");
1482         return(0);
1483     }
1484
1485     //
1486     // Copy the current path to the temporary buffer so it can be manipulated.
1487     //
```

mysdcard.c

```
1488    strcpy(g_cTmpBuf, g_cCwdBuf);
1489
1490    //
1491    // If not already at the root level, then append a separator.
1492    //
1493    if(strcmp("/", g_cCwdBuf))
1494    {
1495        strcat(g_cTmpBuf, "/");
1496    }
1497
1498    //
1499    // Now finally, append the file name to result in a fully specified file.
1500    //
1501    strcat(g_cTmpBuf, filename);
1502
1503    //
1504    // Wait for the UART transmit buffer to empty.
1505    //
1506    #if defined(UART_BUFFERED)
1507        UARTFlushTx(false);
1508    #endif
1509
1510
1511    //
1512    // Open/Create File
1513    //
1514
1515  fresult = f_open(&fnew, g_cTmpBuf, FA_OPEN_ALWAYS | FA_WRITE |FA_READ);
1516
1517  if(fresult != FR_OK)
1518  {
1519      return(fresult);
1520  }
1521
1522  // Set Pointer to end of File
1523  f_lseek(&fnew,fnew.fsize);
1524
1525  // Write String to end of File
1526  fresult = f_write(&fnew, write_Buffer, strlen(write_Buffer), &bw );
1527  if(fresult != FR_OK)
1528  {
1529      return(fresult);
1530  }
1531
1532  /* Close opened files */
1533  f_close(&fnew);
1534
1535
1536  //
1537  // Return success.
1538  //
1539  return(0);
1540
1541 }
1542
1543
1544 //*****************************************************************************
1545 //
1546 // This function implements the "create_file" command.  It creates a file on
1547 // the current working directory with specified filename.
1548 //
1549 //*****************************************************************************
```

mysdcard.c

```
1550
1551 int create_file(const char *filename)
1552 {
1553
1554
1555     FIL fnew;      /* new file object */
1556     FRESULT fresult;
1557
1558     //
1559     // First, check to make sure that the current path (CWD), plus
1560     // the file name, plus a separator and trailing null, will all
1561     // fit in the temporary buffer that will be used to hold the
1562     // file name.  The file name must be fully specified, with path,
1563     // to FatFs.
1564     //
1565     if(strlen(g_cCwdBuf) + strlen(filename) + 1 + 1 > sizeof(g_cTmpBuf))
1566     {
1567         UARTprintf("Resulting path name is too long\n");
1568         return(0);
1569     }
1570
1571     //
1572     // Copy the current path to the temporary buffer so it can be manipulated.
1573     //
1574     strcpy(g_cTmpBuf, g_cCwdBuf);
1575
1576     //
1577     // If not already at the root level, then append a separator.
1578     //
1579     if(strcmp("/", g_cCwdBuf))
1580     {
1581         strcat(g_cTmpBuf, "/");
1582     }
1583
1584     //
1585     // Now finally, append the file name to result in a fully specified file.
1586     //
1587     strcat(g_cTmpBuf, filename);
1588
1589         //
1590         // Wait for the UART transmit buffer to empty.
1591         //
1592         #if defined(UART_BUFFERED)
1593             UARTFlushTx(false);
1594         #endif
1595
1596
1597 //
1598 // Create the File
1599 //
1600   fresult = f_open(&fnew, filename, FA_CREATE_ALWAYS | FA_WRITE );
1601
1602   //
1603   // Check if creation succeeded
1604   //
1605
1606   if(fresult != FR_OK)
1607   {
1608       return(fresult);
1609   }
1610
1611   /* Close opened files */
```

mysdcard.c

```
1612   f_close(&fnew);
1613
1614
1615       //
1616       // Return success.
1617       //
1618       return(0);
1619 }
1620
1621
1622
1623
1624 //*****************************************************************************
1625 //
1626 // Own variation of sdcard.c main function from stellarisware to implement an
1627 // command line based file explorer for browsing the SD Card
1628 //
1629 //*****************************************************************************
1630
1631 int my_start_cmd_line(int argc, char *argv[])
1632 {
1633     // set Command line pointer to the beginning of the SD-Card/Browser command structure
1634     g_psCmdTable = &g_sBrowserCmdTable[0];
1635
1636     sprintf(g_cSdLocalBuf, "SD-Card: %s>", g_cCwdBuf);
1637
1638     // set location buffer to Browser
1639     g_cLocalBuf = (char*)g_cSdLocalBuf;
1640
1641     return(0);
1642 }
1643
1644 //*****************************************************************************
1645 //
1646 // Main function for measurement of voltages, current and temperature. Multiplex
1647 // all relevant cells (depended on configuration) and write data to LogFile on
1648 // SD-Card.
1649 //
1650 //*****************************************************************************
1651 void do_measure(){
1652
1653 //*****************************************************************************
1654 // Use this commented version if current is needed measured with every cellvoltage
1655 //*****************************************************************************
1656 //  unsigned int i = 0;
1657 //  char buf[64];
1658 //  //UARTprintf("\n");
1659 //  for (i=1; i <= config.quantity_cells; i++){
1660 //      sprintf(buf, "%02d,%04d,%02d,%02d,%02d,%02d,%02d,uV,%07d,C,%03d,mA,%07d,%01d\n"
1661 //                          , i, clock_year, clock_month, clock_day, clock_hour,
    clock_min, clock_sec
1662 //                          ,adc_get_voltage(i), get_temperature(i), adc_get_current(),
    power_relay_active);
1663 //
1664 //      //UARTprintf(buf);
1665 //      add_to_file(MEAS_FILE, buf);
1666 //
1667 ////        if(control_connection){
1668 ////            telnet_write(buf);
1669 ////        }
1670 //
1671 //  }
```

mysdcard.c

```
1672 //*****************************************************************************
1673
1674         unsigned int i = 0;
1675         char buf[64];
1676         UARTprintf("\n");
1677         //Turn on Measure-LED
1678         GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  GPIO_PIN_5);
1679
1680         adc_get_current();
1681         for (i=1; i <= config.quantity_cells; i++){
1682             sprintf(buf, "%02d,%04d,%02d,%02d,%02d,%02d,%02d,%07d,%03d,%07d,%01d\n"
1683                                 , i, clock_year, clock_month, clock_day, clock_hour,
     clock_min, clock_sec
1684                                 ,adc_get_voltage(i), get_temperature(i), g_iCurrent,
     power_relay_active);
1685
1686             UARTprintf(buf);
1687             add_to_file(MEAS_FILE, buf);
1688
1689             if(control_connection){
1690                 telnet_write(buf);
1691             }
1692
1693         }
1694         //Turn off Measure-LED
1695         GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_5,  0x00);
1696
1697 }
1698
1699
1700
```

myssi.h

```
 1 /*
 2  * myssi.h
 3  *
 4  *  Created on: 26.05.2013
 5  *      Author: Thomas W.
 6  */
 7
 8 #ifndef MYSSI_H_
 9 #define MYSSI_H_
10
11
12
13 // SPI chip selects
14 typedef enum
15     {
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
```

myssi.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           my_ssi.c

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

last modified:                  2013/05/26

Project Status                  Under Construction
Status:                         not working...

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code for ssi1 initalization for use with SD Card
  and 16-bit ADC

*/


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


#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ssi.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/ssi.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "header/myssi.h"
#include "third_party/fatfs/src/diskio.h"



//*****************************************************************************
//
// Initialize SSI1-Interface
//
//*****************************************************************************
void ssi1_init(void)
{
```

myssi.c

```
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:
```

myssi.c

```
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
```

myuart.h

```
 1 /*
 2  * timer.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
```

myuart.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           myuart.c

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

last modified:                  2013/05/26

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code for UART0 Initialization for use with
                                UARTStdio.h and UARTprintf()

*/


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

#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "header/myuart.h"
#include "utils/uartstdio.h"
#include "driverlib/rom_map.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_uart.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "header/mycmdline.h"
#include "string.h"


//*****************************************************************************
//
// Defines the size of the buffer that holds the command line.
//
//*****************************************************************************
#define UART_BUF_SIZE    64

//*****************************************************************************
// Boolean for fully received uart commands
//*****************************************************************************
tBoolean received = false;
```

myuart.c

```
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     //
```

myuart.c

```
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
```

relais.h

```
 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
```

relais.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           relais.c

Auhtor:                         Thomas Wisnewski
Credits:                        Stellaris Ware

last modified:                  2013/05/27

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code for controlling installed relais

*/
/*****************************************************************************
*
*   Includings
*
*****************************************************************************/
#include <string.h>
#include <stdarg.h>
#include <stdio.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"
#include <inc/hw_ssi.h>
#include "inc/hw_gpio.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include <driverlib/ssi.h>
#include <driverlib/gpio.h>
#include <driverlib/sysctl.h>

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

extern volatile unsigned int power_relay_active;

// TCP-Socket für die aktuelle Steuerungs-Verbindung
```

relais.c

```
63 extern struct tcp_pcb* control_connection;
64
65 //*****************************************************************************
66 //
67 // Defines the size of the buffer that holds the input data.
68 //
69 //*****************************************************************************
70 #define RELAIS_INPUT_DATA_SIZE    64
71
72 //*****************************************************************************
73 //
74 // A temporary data buffer used for input data
75 //
76 //*****************************************************************************
77 static char g_cTmpInpData[RELAIS_INPUT_DATA_SIZE];
78
79
80 //*****************************************************************************
81 //
82 // Initialize GPIO ports for relay-mux and powerrelays
83 //
84 //*****************************************************************************
85 void init_relais(void)
86 {
87
88     UARTprintf("Initializing Relais...");
89
90     //Enable Peripheral for Relais-Mux
91     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);   // enable peripheral
92     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);   // enable peripheral
93
94     //Enable Peripheral for Relais on power section
95     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);   // enable peripheral
96     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);   // enable peripheral
97
98 //  //Unlock PB7 for changing from NMI-Mode
99 //
100 //  //
101 //  // Unlock access to the commit register
102 //  //
103 //  HWREG(GPIO_PORTB_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY_DD;
104 //
105 //  //
106 //  // Set the commit register for PB7 to allow changing the function
107 //  //
108 //  HWREG(GPIO_PORTB_BASE + GPIO_O_CR) = 0x80;
109 //
110 //  //
111 //  // Enable the alternate function for PB7 (NMI)
112 //  //
113 //  HWREG(GPIO_PORTB_BASE + GPIO_O_AFSEL) |= 0x80;
114 //
115 //  //
116 //  // Turn on the digital enable for PB7
117 //  //
118 //  HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= 0x80;
119
120
121     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3 ); // set PB2 and PB3 as
    digital output
122     GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7); // set
    PH3, PH4 and PH7 as digital output
```

relais.c

```
123     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7); // set
    PC5, PC6 and PC7 as digital output
124     GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_7); // set PG7 as digital output
125
126     GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2 | GPIO_PIN_3, 0x00); // set PB2 and PB3 to
    logic "0"
127     GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_7, 0x00); // set PH3,
    PH4 and PH7 to logic "0"
128
129     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, 0x00); // set PC5,
    PC6 and PC7 to logic "0"
130     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7,  GPIO_PIN_7); // set PG7 to logic "1"
    "Strobe"
131
132     switch_relais(0);
133     switch_power_relais(0);
134
135     UARTprintf("done\n");
136 }
137
138 //*****************************************************************************
139 //
140 // Function for switching relays-mux
141 //
142 //*****************************************************************************
143 void switch_relais(int choose)
144 {
145
146         GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PB7 (Enable) to logic
    "1"
147
148
149         switch(choose){
150
151         case 0:
152
153             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
154             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
155             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
156             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
157
158         break;
159
160
161
162         case 1:
163
164             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
165             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
166             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
167             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
168             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
169
170         break;
171
172         case 2:
173
174             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
175             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
176             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
177             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
```

relais.c

```
178             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
179
180         break;
181
182         case 3:
183
184             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
185             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
186             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
187             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
188             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
189
190         break;
191
192         case 4:
193
194             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
195             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
196             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
197             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
198             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
199
200         break;
201
202         case 5:
203
204             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
205             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
206             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
207             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
208             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
209
210         break;
211
212         case 6:
213
214             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
215             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
216             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
217             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
218             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
219
220         break;
221
222         case 7:
223
224             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
225             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
226             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
227             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
228             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
229
230         break;
231
232         case 8:
233
```

relais.c

```
234             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
235             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
236             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, GPIO_PIN_3);
237             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
238             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
239
240         break;
241
242         case 9:
243
244             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
245             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
246             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
247             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
248             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
249
250         break;
251
252         case 10:
253
254             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
255             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
256             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
257             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
258             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
259
260         break;
261
262         case 11:
263
264             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
265             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
266             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
267             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
268             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
269
270         break;
271
272         case 12:
273
274             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, GPIO_PIN_2);
275             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3);
276             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
277             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, GPIO_PIN_4);
278             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_7, 0x00); // set PB7 (Enable) to logic
    "0"
279
280         break;
281
282         default:
283
284             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, 0x00);
285             GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0x00);
286             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3, 0x00);
287             GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_4, 0x00);
288
289
290         break;
```

relais.c

```
291     }
292
293
294
295
296 }
297
298
299 //*****************************************************************************
300 //
301 // Function for switching powerrelay
302 //
303 //*****************************************************************************
304 int switch_power_relais(int choose)
305 {
306     int active;
307     unsigned long ul_delay_count;
308
309         switch(choose){
310
311         case 0:
312
313             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
314             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
315             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
316             active = 0;
317
318         break;
319
320         case 1:
321
322             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
323             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
324             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
325             active = 1;
326         break;
327
328         case 2:
329
330             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
331             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
332             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
333             active = 2;
334         break;
335
336         case 3:
337
338             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
339             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
340             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, GPIO_PIN_7);
341             active = 3;
342         break;
343
344         case 4:
345
346             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, GPIO_PIN_5);
347             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
348             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
349             active = 4;
350         break;
351
352         default:
```

relais.c

```
353
354             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0x00);
355             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0x00);
356             GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, 0x00);
357             active = 0;
358         break;
359
360         }
361
362         GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, 0x00); // set PG7 (Strobe) to logic "0"
363         // ca. 10 ms delay
364         ul_delay_count = 100000;
365         while (ul_delay_count) ul_delay_count--;
366         //--------------------------------
367         GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_7, GPIO_PIN_7); // set PG7 (Strobe) to logic
    "1"
368         // ca. 10 ms delay
369         ul_delay_count = 100000;
370         while (ul_delay_count) ul_delay_count--;
371         //--------------------------------
372
373         return active;
374 }
375
376 //*****************************************************************************
377 //
378 // This function implements the "Relais" command. It activates ONE chosen power relais
379 //
380 //*****************************************************************************
381 int
382 Cmd_relais(int argc, char *argv[])
383 {
384     char buf[32];
385
386     int relais, active;
387     //
388     // Copy the first input data into buffer.
389     //
390     strcpy(g_cTmpInpData, argv[1]);
391
392     if(g_cTmpInpData[1] != 0 && g_cTmpInpData[2] == 0){
393         relais = 10 * (g_cTmpInpData[0] - '0') + (g_cTmpInpData[1] - '0');
394     }
395     else if(g_cTmpInpData[0] != 0 && g_cTmpInpData[1] == 0){
396         relais = (g_cTmpInpData[0] - '0');
397     }
398     else if(g_cTmpInpData[0] == 0){
399         relais = 0;
400     }
401     else{
402         UARTprintf("\nBAD COMMAND!");
403         sprintf(buf, "BAD COMMAND!\n");
404         if(control_connection){
405             telnet_write(buf);
406         }
407         return(0);
408     }
409
410     active = switch_power_relais(relais);
411     sprintf(buf, "Relais %02d activated\n", active);
412     UARTprintf("\nRelais %02d activated", active);
413     if(control_connection){
```

relais.c

```
414         telnet_write(buf);
415     }
416     power_relay_active = active;
417     //
418     // Return success.
419     //
420     return(0);
421 }
422
```

rtc.h

```c
/*
 * rtc.h
 *
 *  Created on: 03.06.2013
 *      Author: Thomas W.
 */

#ifndef RTC_H_
#define RTC_H_

/*****************************************************************************
*
* Function Declarations
*
*****************************************************************************/
void rtc_init(void);
void set_clock_values(void);
void set_rtc_values(unsigned int year, unsigned int month, unsigned int day, unsigned int
  hour, unsigned int min, unsigned int sec );

#endif /* RTC_H_ */

```

rtc.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           rtc.c

Auhtor:                         Thomas Wisnewski
Credits:                        Stellaris Ware

last modified:                  2013/06/03

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Source Code extern RTC-Clock

*/
/*******************************************************************************
*
*   Includings
*
*******************************************************************************/
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "inc/hw_i2c.h"
#include "driverlib/i2c.h"
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "header/rtc.h"


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

//*****************************************************************************
//
// Initialize RTC-Clock and setup systemclok by values from RTC
//
//*****************************************************************************
void rtc_init(void){


    UARTprintf("Initializing RTC...");

    //Enable peripherals
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
```

rtc.c

```
63     ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
64
65     GPIOPinConfigure(GPIO_PA6_I2C1SCL);
66     GPIOPinConfigure(GPIO_PA7_I2C1SDA);
67     ROM_GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_6 | GPIO_PIN_7);
68
69     ROM_I2CMasterInitExpClk(I2C1_MASTER_BASE,SysCtlClockGet(),false);
   //InitializeMasterandSlave
70
71     set_clock_values();
72
73
74
75     UARTprintf("done\n");
76
77 }
78
79
80 //
81 // Function for setting clock values from rtc. Called once while Initializing on startup.
82 //
83 void set_clock_values(void){
84
85
86     unsigned char adc_dat;
87
88
89     //Set year
90     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
91
92     I2CMasterDataPut(I2C1_MASTER_BASE,0x06); // location address
93     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
94     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
95
96     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
97     while(I2CMasterBusy(I2C1_MASTER_BASE));
98
99     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
100    while(I2CMasterBusy(I2C1_MASTER_BASE));
101
102    adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
103    while(I2CMasterBusy(I2C1_MASTER_BASE));
104
105    clock_year = ((adc_dat >> 4) & 0x0f) * 10 + (adc_dat & 0x0f) + 2000;
106
107
108    //Set month
109    I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
110
111    I2CMasterDataPut(I2C1_MASTER_BASE,0x05); // location address
112    I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
113    while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
114
115    I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
116    while(I2CMasterBusy(I2C1_MASTER_BASE));
117
118    I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
119    while(I2CMasterBusy(I2C1_MASTER_BASE));
120
121    adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
122    while(I2CMasterBusy(I2C1_MASTER_BASE));
123
```

rtc.c

```
124     clock_month = ((adc_dat >> 4) & 0x03) * 10 + (adc_dat & 0x0f);
125
126     //Set date
127     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
128
129     I2CMasterDataPut(I2C1_MASTER_BASE,0x04); // location address
130     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
131     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
132
133     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
134     while(I2CMasterBusy(I2C1_MASTER_BASE));
135
136     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
137     while(I2CMasterBusy(I2C1_MASTER_BASE));
138
139     adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
140     while(I2CMasterBusy(I2C1_MASTER_BASE));
141
142     clock_day = ((adc_dat >> 4) & 0x03) * 10 + (adc_dat & 0x0f);
143
144     //Set hour
145     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
146
147     I2CMasterDataPut(I2C1_MASTER_BASE,0x02); // location address
148     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
149     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
150
151     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
152     while(I2CMasterBusy(I2C1_MASTER_BASE));
153
154     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
155     while(I2CMasterBusy(I2C1_MASTER_BASE));
156
157     adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
158     while(I2CMasterBusy(I2C1_MASTER_BASE));
159
160     clock_hour = ((adc_dat >> 4) & 0x03) * 10 + (adc_dat & 0x0f);
161
162     //Set minute
163     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
164
165     I2CMasterDataPut(I2C1_MASTER_BASE,0x01); // location address
166     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
167     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
168
169     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
170     while(I2CMasterBusy(I2C1_MASTER_BASE));
171
172     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
173     while(I2CMasterBusy(I2C1_MASTER_BASE));
174
175     adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
176     while(I2CMasterBusy(I2C1_MASTER_BASE));
177
178     clock_min = ((adc_dat >> 4) & 0x07) * 10 + (adc_dat & 0x0f);
179
180     //Set second
181     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
182
183     I2CMasterDataPut(I2C1_MASTER_BASE,0x00); // location address
184     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_SEND);
185     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
```

rtc.c

```
186
187     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, true);
188     while(I2CMasterBusy(I2C1_MASTER_BASE));
189
190     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);
191     while(I2CMasterBusy(I2C1_MASTER_BASE));
192
193     adc_dat=I2CMasterDataGet(I2C1_MASTER_BASE);
194     while(I2CMasterBusy(I2C1_MASTER_BASE));
195
196     clock_sec = ((adc_dat >> 4) & 0x07) * 10 + (adc_dat & 0x0f);
197
198
199 }
200
201 //
202 // Function for setting values in rtc. Not needed for run time functions.
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     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
207
208     I2CMasterDataPut(I2C1_MASTER_BASE,0x06); // location address
209     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
210     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
211
212     I2CMasterDataPut(I2C1_MASTER_BASE,year); // location address
213     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
214     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
215
216     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
217
218     I2CMasterDataPut(I2C1_MASTER_BASE,0x05); // location address
219     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
220     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
221
222     I2CMasterDataPut(I2C1_MASTER_BASE,month); // location address
223     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
224     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
225
226     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
227
228     I2CMasterDataPut(I2C1_MASTER_BASE,0x04); // location address
229     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
230     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
231
232     I2CMasterDataPut(I2C1_MASTER_BASE,day); // location address
233     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
234     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
235
236     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
237
238     I2CMasterDataPut(I2C1_MASTER_BASE,0x02); // location address
239     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
240     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
241
242     I2CMasterDataPut(I2C1_MASTER_BASE,hour); // location address
243     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
244     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
245
246     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
```

rtc.c

```
247
248     I2CMasterDataPut(I2C1_MASTER_BASE,0x01); // location address
249     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
250     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
251
252     I2CMasterDataPut(I2C1_MASTER_BASE,min); // location address
253     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
254     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
255
256     I2CMasterSlaveAddrSet(I2C1_MASTER_BASE, 0x68, false);
257
258     I2CMasterDataPut(I2C1_MASTER_BASE,0x00); // location address
259     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_START);
260     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
261
262     I2CMasterDataPut(I2C1_MASTER_BASE,sec); // location address
263     I2CMasterControl(I2C1_MASTER_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
264     while(I2CMasterBusy(I2C1_MASTER_BASE)); // wait till transfering
265
266 }
267
268
269
```

startup_ccs.c

```
 1 /*
 2 Project:                          battery_cycling_sw_v3
 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
29 //*****************************************************************************
30 //
31 // startup_ccs.c - Startup code for use with TI's Code Composer Studio.
32 //
33 // Copyright (c) 2009-2012 Texas Instruments Incorporated.  All rights reserved.
34 // Software License Agreement
35 //
36 // Texas Instruments (TI) is supplying this software for use solely and
37 // exclusively on TI's microcontroller products. The software is owned by
38 // TI and/or its suppliers, and is protected under applicable copyright
39 // laws. You may not combine this software with "viral" open-source
40 // software in order to form a larger program.
41 //
42 // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
43 // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
44 // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
45 // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
46 // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
47 // DAMAGES, FOR ANY REASON WHATSOEVER.
48 //
49 // This is part of revision 8555 of the EK-LM3S9B92 Firmware Package.
50 //
51 //*****************************************************************************
52
53 //*****************************************************************************
54 //
55 // Forward declaration of the default fault handlers.
56 //
57 //*****************************************************************************
58 void ResetISR(void);
59 static void NmiSR(void);
60 static void FaultISR(void);
61 static void IntDefaultHandler(void);
62 extern void UARTStdioIntHandler(void);
```

startup_ccs.c

```
63 extern void SysTickHandler(void);
64
65
66 //*****************************************************************************
67 //
68 // External declaration for the reset handler that is to be called when the
69 // processor is started
70 //
71 //*****************************************************************************
72 extern void _c_int00(void);
73
74 //*****************************************************************************
75 //
76 // Linker variable that marks the top of the stack.
77 //
78 //*****************************************************************************
79 extern unsigned long __STACK_TOP;
80
81 //*****************************************************************************
82 //
83 // External declaration for the interrupt handler used by the application.
84 //
85 //*****************************************************************************
86 extern void UARTIntHandler(void);
87 extern void Timer0IntHandler(void);
88 extern void Timer1IntHandler(void);
89 extern void Timer2IntHandler(void);
90 extern void ADC0Handler(void);
91 extern void ADC1Handler(void);
92 extern void lwIPEthernetIntHandler(void);
93 extern void UART0IntHandler(void);
94
95 extern void GPIOD01Handler(void);
96 extern void GPIOF00Handler(void);
97 extern void GPIOC04Handler(void);
98 extern void GPIOH05Handler(void);
99
100 //*****************************************************************************
101 //
102 // The vector table.  Note that the proper constructs must be placed on this to
103 // ensure that it ends up at physical address 0x0000.0000 or at the start of
104 // the program if located at a start address other than 0.
105 //
106 //*****************************************************************************
107 #pragma DATA_SECTION(g_pfnVectors, ".intvecs")
108 void (* const g_pfnVectors[])(void) =
109 {
110     (void (*)(void))((unsigned long)&__STACK_TOP),
111                                             // The initial stack pointer
112     ResetISR,                               // The reset handler
113     NmiSR,                                  // The NMI handler
114     FaultISR,                               // The hard fault handler
115     IntDefaultHandler,                      // The MPU fault handler
116     IntDefaultHandler,                      // The bus fault handler
117     IntDefaultHandler,                      // The usage fault handler
118     0,                                      // Reserved
119     0,                                      // Reserved
120     0,                                      // Reserved
121     0,                                      // Reserved
122     IntDefaultHandler,                      // SVCall handler
123     IntDefaultHandler,                      // Debug monitor handler
124     0,                                      // Reserved
```

startup_ccs.c

```
125     IntDefaultHandler,                      // The PendSV handler
126     SysTickHandler,                         // The SysTick handler
127     IntDefaultHandler,                      // GPIO Port A
128     IntDefaultHandler,                      // GPIO Port B
129     GPIOC04Handler,                         // GPIO Port C
130     GPIOD01Handler,                         // GPIO Port D
131     IntDefaultHandler,                      // GPIO Port E
132     UART0IntHandler,                        // UART0 Rx and Tx
133     //UARTStdioIntHandler,                  // UART0 Rx and Tx
134     IntDefaultHandler,                      // UART1 Rx and Tx
135     IntDefaultHandler,                      // SSI0 Rx and Tx
136     IntDefaultHandler,                      // I2C0 Master and Slave
137     IntDefaultHandler,                      // PWM Fault
138     IntDefaultHandler,                      // PWM Generator 0
139     IntDefaultHandler,                      // PWM Generator 1
140     IntDefaultHandler,                      // PWM Generator 2
141     IntDefaultHandler,                      // Quadrature Encoder 0
142     ADC0Handler,                            // ADC Sequence 0
143     IntDefaultHandler,                      // ADC Sequence 1
144     IntDefaultHandler,                      // ADC Sequence 2
145     IntDefaultHandler,                      // ADC Sequence 3
146     IntDefaultHandler,                      // Watchdog timer
147     Timer0IntHandler,                       // Timer 0 subtimer A
148     IntDefaultHandler,                      // Timer 0 subtimer B
149     Timer1IntHandler,                       // Timer 1 subtimer A
150     IntDefaultHandler,                      // Timer 1 subtimer B
151     Timer2IntHandler,                      // Timer 2 subtimer A
152     IntDefaultHandler,                      // Timer 2 subtimer B
153     IntDefaultHandler,                      // Analog Comparator 0
154     IntDefaultHandler,                      // Analog Comparator 1
155     IntDefaultHandler,                      // Analog Comparator 2
156     IntDefaultHandler,                      // System Control (PLL, OSC, BO)
157     IntDefaultHandler,                      // FLASH Control
158     GPIOF00Handler,                         // GPIO Port F
159     IntDefaultHandler,                      // GPIO Port G
160     GPIOH05Handler,                         // GPIO Port H
161     IntDefaultHandler,                      // UART2 Rx and Tx
162     IntDefaultHandler,                      // SSI1 Rx and Tx
163     IntDefaultHandler,                      // Timer 3 subtimer A
164     IntDefaultHandler,                      // Timer 3 subtimer B
165     IntDefaultHandler,                      // I2C1 Master and Slave
166     IntDefaultHandler,                      // Quadrature Encoder 1
167     IntDefaultHandler,                      // CAN0
168     IntDefaultHandler,                      // CAN1
169     IntDefaultHandler,                      // CAN2
170     lwIPEthernetIntHandler,                 // Ethernet
171     IntDefaultHandler,                      // Hibernate
172     IntDefaultHandler,                      // USB0
173     IntDefaultHandler,                      // PWM Generator 3
174     IntDefaultHandler,                      // uDMA Software Transfer
175     IntDefaultHandler,                      // uDMA Error
176     IntDefaultHandler,                      // ADC1 Sequence 0
177     ADC1Handler,                            // ADC1 Sequence 1
178     IntDefaultHandler,                      // ADC1 Sequence 2
179     IntDefaultHandler,                      // ADC1 Sequence 3
180     IntDefaultHandler,                      // I2S0
181     IntDefaultHandler,                      // External Bus Interface 0
182     IntDefaultHandler                       // GPIO Port J
183 };
184
185 //*****************************************************************************
186 //
```

startup_ccs.c

```
187 // This is the code that gets called when the processor first starts execution
188 // following a reset event.  Only the absolutely necessary set is performed,
189 // after which the application supplied entry() routine is called.  Any fancy
190 // actions (such as making decisions based on the reset cause register, and
191 // resetting the bits in that register) are left solely in the hands of the
192 // application.
193 //
194 //*****************************************************************************
195 void
196 ResetISR(void)
197 {
198     //
199     // Jump to the CCS C initialization routine.
200     //
201     __asm("    .global _c_int00\n"
202           "    b.w     _c_int00");
203 }
204
205 //*****************************************************************************
206 //
207 // This is the code that gets called when the processor receives a NMI.  This
208 // simply enters an infinite loop, preserving the system state for examination
209 // by a debugger.
210 //
211 //*****************************************************************************
212 static void
213 NmiSR(void)
214 {
215     //
216     // Enter an infinite loop.
217     //
218     while(1)
219     {
220     }
221 }
222
223 //*****************************************************************************
224 //
225 // This is the code that gets called when the processor receives a fault
226 // interrupt.  This simply enters an infinite loop, preserving the system state
227 // for examination by a debugger.
228 //
229 //*****************************************************************************
230 static void
231 FaultISR(void)
232 {
233     unsigned long ul_delay_count;
234     GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_4,  0x00);
235
236     Cmd_interprete("stop");
237     //
238     // Enter an infinite loop.
239     //
240     while(1)
241     {
242
243         GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_6, ~GPIOPinRead(GPIO_PORTJ_BASE,
    GPIO_PIN_6));
244
245         // ca. 0,5 s delay
246         ul_delay_count = 5000000;
247         while (ul_delay_count) ul_delay_count--;
```

startup_ccs.c

```
248         //-------------------------------
249
250     }
251 }
252
253 //*****************************************************************************
254 //
255 // This is the code that gets called when the processor receives an unexpected
256 // interrupt.  This simply enters an infinite loop, preserving the system state
257 // for examination by a debugger.
258 //
259 //*****************************************************************************
260 static void
261 IntDefaultHandler(void)
262 {
263     //
264     // Go into an infinite loop.
265     //
266     while(1)
267     {
268     }
269 }
270
```

temperature.h

```
 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
```

temperature.c

```
/*
Project:                        battery_cycling_sw_v3
File:                           main.c

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

last modified:                  2013/05/28

Project Status                  Under Construction
Status:                         running

CCS:                            5.5.1.00031
Stellarisware:                  8555

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

Description:                    Code for initializing intern ADCs for temperature
 measurement
*/

/*****************************************************************************
*
*   Includings
*
*****************************************************************************/
#include "driverlib/rom.h"
#include "third_party/fatfs/src/ff.h"
#include "third_party/fatfs/src/diskio.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include <stdio.h>
#include "utils/uartstdio.h"
#include "math.h"



//*****************************************************************************
//
// A global data buffer used for actual Cell temperature
//
//*****************************************************************************
volatile int g_iCellTemperature[12] = {0};

//Buffer for ADC-Samples
unsigned long g_pulData0[8];
unsigned long g_pulData1[4];
```

temperature.c

```
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
```

temperature.c

```
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         unsigned int temperature;
148         if((choose >= 0) && (choose <= 8)){
149             temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData0[choose -
    1] / 10000 - 1 ) * 298) - 273) * 10;
150         }
151
152         if((choose >= 9) && (choose <= 12)){
153             temperature = (3988 * 298 / ( 3988 + log(10000.0 * 1024.0 / g_pulData1[choose -
    9] / 10000 - 1 ) * 298) - 273) * 10;
154         }
155
156         if(temperature == (-1u)){
157         g_iCellTemperature[choose-1] = 0;
158         temperature = 0;
159         }
160
161         else
162         g_iCellTemperature[choose-1] = temperature;
163
164
165 return temperature;
166 }
167
168
169 //
170 // The ADC unit 0 interrupt hander
171 //
172 void
173 ADC0Handler(void)
174 {
175     ROM_ADCIntClear(ADC0_BASE, 0);
176     //write adc-values to global buffer
177     ROM_ADCSequenceDataGet(ADC0_BASE, 0, g_pulData0);
178 }
179
180
181 //
182 // The ADC unit 1 interrupt handler
```

temperature.c

```
183 //
184 void
185 ADC1Handler(void)
186 {
187     ROM_ADCIntClear(ADC1_BASE, 1);
188     //write adc-values to global buffer
189     ROM_ADCSequenceDataGet(ADC1_BASE, 1, g_pulData1);
190 }
191
192
193
```

E. MatLab-Skripte

connect.m

```
1  % Aufbau der Verbindungen zur BCMV2.
2  % Es wird eine TCP-Verbindung mit der
3  % IP-Adresse zum Port 56936 hergestellt.
4  % Es wird ausserdem der Timeout der Verbindungen
5  % auf eine Sekunde gesetzt, da die spaeter
6  % benoetigte Funktion fread blockend ist und
7  % fuer den Fall, dass keine Daten ausgelesen
8  % werden koennen nicht zu lange wartet.
9  function [connection failed] = connect(address)
10 failed = 0;
11
12
13 connection = tcpip(address, 56936);
14 connection.OutputBufferSize = 15000;
15 connection.InputBufferSize = 15000;
16
17 connection.Timeout = 1;
18 set(connection, 'ReadAsyncMode', 'continuous');
19 set(connection, 'BytesAvailableFcn', {'receive_cmd'});
20 fopen(connection);
21
22 if(length(connection.status) ~= 4)
23 fprintf('Unable to connect to %s\n', address);
24 failed = 1;
25 end
26
27 %fread(connection, 1); % Puffer leeren
28
29
30 end
```

main.m

```
%% Establish Connection
% Set the ip of the Controller
ip = '192.168.0.128';

% Connect to the Controller
cons = connect(ip);

%% Order help
send_cmd(cons, sprintf('help\n'));

%% Enter browser
send_cmd(cons, sprintf('browser\n'));

%% Enter config
send_cmd(cons, sprintf('config\n'));

%% Start measurement cycling
send_cmd(cons, sprintf('start\n'));

%% Stop measurement cycling
send_cmd(cons, sprintf('stop\n'));

%% Exit to root
send_cmd(cons, sprintf('exit\n'));
```

plotdata_4cells.m

```
cell_1 = zeros(fix(length(data)/4.1),11);
cell_2 = zeros(fix(length(data)/4.1),11);
cell_3 = zeros(fix(length(data)/4.1),11);
cell_4 = zeros(fix(length(data)/4.1),11);
time   = zeros(fix(length(data)/4.1),1);

for count_line = 1:4:length(data)
    cell_1((count_line-1)/4+1,1:11) = data(count_line,1:11);
end


for count_line = 2:4:length(data)
    cell_2((count_line-2)/4+1,1:11) = data(count_line,1:11);
end
```

```
15
16
17 for count_line = 3:4:length(data)
18     cell_3((count_line-3)/4+1,1:11) = data(count_line,1:11);
19 end
20
21
22 for count_line = 4:4:length(data)
23     cell_4((count_line-4)/4+1,1:11) = data(count_line,1:11);
24 end
25
26 for k = 1:4:length(data)
27
28   time((k-1)/4+1) = datenum(data(k,2), data(k,3), data(k,4), data(
        k,5), data(k,6), data(k,7));
29
30 end
31
32 %%
33 figure(1);
34
35 subplot(3,1,1)
36 plot(smooth(cell_1(:,8)./1e6));
37 hold on
38 plot(smooth(cell_2(:,8)./1e6),'r');
39 plot(smooth(cell_3(:,8)./1e6),'k');
40 plot(smooth(cell_4(:,8)./1e6),'g');
41 axis([0 69119 0 4.5]);
42 labels = datestr(time, 'dd.mm HH:MM');
43 set(gca,'Xtick',1:(length(labels)/30):length(labels));
44 set(gca, 'XTickLabel', labels(1:(length(labels)/30):length(labels)
       ,1:11));
45 rotateticklabel(gca ,45);
46 ylabel('Voltage in V');
47 hold off
48 grid on
49
50
51 subplot(3,1,2)
52 plot(smooth(cell_1(:,9)./1e1));
53 hold on
54 plot(smooth(cell_2(:,9)./1e1),'r');
55 plot(smooth(cell_3(:,9)./1e1),'k');
56 plot(smooth(cell_4(:,9)./1e1),'g');
```

```
set(gca,'Xtick',1:(length(labels)/30):length(labels));
set(gca, 'XTickLabel', labels(1:(length(labels)/30):length(labels)
   ,1:11));
rotateticklabel(gca ,45);
ylabel('Temperature in C');
hold off
grid on

subplot(3,1,2)
plot(smooth(cell_1(:,10)./1));
hold on
plot(smooth(cell_2(:,10)./1),'r');
plot(smooth(cell_3(:,10)./1),'k');
plot(smooth(cell_4(:,10)./1),'g');
set(gca,'Xtick',1:(length(labels)/30):length(labels));
set(gca, 'XTickLabel', labels(1:(length(labels)/30):length(labels)
   ,1:11));
rotateticklabel(gca ,45);
ylabel('Current in mA');
hold off
grid on
```

receive_cmd.m

```
% Empfangen einer Sequenz von der BCMV2.
% Param: cons: Die geoeffneten TCP-Verbindungen
% event: Art
% Wiedergbabe der empfangene Sequenz
% 0=Fehler, keine Bestaetigung empfangen
function error = receive_cmd(cons, event)

        count = fscanf(cons);
    if(count < 1)
        fprintf('Error: Unable to receive sequence');
        fclose(cons);
        error = 0;
    else
        error = 1;
    end
    fprintf('%s', count);

end
```

rotateticklabel.m

```
1  function th=rotateticklabel(h,rot,demo)
2  %ROTATETICKLABEL rotates tick labels
3  %   TH=ROTATETICKLABEL(H,ROT) is the calling form where H is a handle to
4  %   the axis that contains the XTickLabels that are to be rotated. ROT is
5  %   an optional parameter that specifies the angle of rotation. The default
6  %   angle is 90. TH is a handle to the text objects created. For long
7  %   strings such as those produced by datetick, you may have to adjust the
8  %   position of the axes so the labels don't get cut off.
9  %
10 %   Of course, GCA can be substituted for H if desired.
11 %
12 %   TH=ROTATETICKLABEL([],[],'demo') shows a demo figure.
13 %
14 %   Known deficiencies: if tick labels are raised to a power, the power
15 %   will be lost after rotation.
16 %
17 %   See also datetick.
18
19 %   Written Oct 14, 2005 by Andy Bliss
20 %   Copyright 2005 by Andy Bliss
21
22 %DEMO:
23 if nargin==3
24     x=[now-.7 now-.3 now];
25     y=[20 35 15];
26     figure
27     plot(x,y,'.-')
28     datetick('x',0,'keepticks')
29     h=gca;
30     set(h,'position',[0.13 0.35 0.775 0.55])
31     rot=90;
32 end
33
34 %set the default rotation if user doesn't specify
35 if nargin==1
```

```
36      rot=90;
37  end
38  %make sure the rotation is in the range 0:360 (brute force method)
39  while rot>360
40      rot=rot-360;
41  end
42  while rot<0
43      rot=rot+360;
44  end
45  %get current tick labels
46  a=get(h,'XTickLabel');
47  %erase current tick labels from figure
48  set(h,'XTickLabel',[]);
49  %get tick label positions
50  b=get(h,'XTick');
51  c=get(h,'YTick');
52  %make new tick labels
53  if rot<180
54      th=text(b,repmat(c(1)-.1*(c(2)-c(1)),length(b),1),a,'
        HorizontalAlignment','right','rotation',rot);
55  else
56      th=text(b,repmat(c(1)-.1*(c(2)-c(1)),length(b),1),a,'
        HorizontalAlignment','left','rotation',rot);
57  end
```

send_cmd.m

```
1  % Senden eines Befehls an dire BCMV2.
2  % Param: cons: Die geoeffneten TCP-Verbindungen
3  % cmd: Befehl mit Zeilenumbruch am Ende
4  function error = send_cmd(cons, cmd)
5
6          fwrite(cons, cmd);
7          error = 0;
8  end
```

F. Berechnung Kalibrierwerte

CELL1

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	96
			Gain:	76308362

LOW

3931	300060	3931
3931		
3932		
3931		
3931		
3932		
3929		
3932		
3930		
3928		
3930		
3932		
3930		
3931		
3932		
3933		
3932		
3932		
3930		
3930		

HIGH	3176870	41631
41632		
41631		
41630		
41631		
41630		
41629		
41630		
41631		
41630		
41631		
41631		
41631		
41631		
41631		
41630		
41631		
41632		
41631		
41631		
41631		

CELL2

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	125
			Gain:	76307211

LOW

3931	300050	3931
3929		
3929		
3931		
3932		
3930		
3930		
3930		
3931		
3928		
3930		
3932		
3930		
3933		
3932		
3931		
3931		
3929		
3931		
3930		

HIGH	3176870	41631
41631		
41632		
41634		
41631		
41632		
41630		
41630		
41631		
41632		
41631		
41631		
41632		
41629		
41630		
41630		
41631		
41630		
41632		
41632		
41629		

CELL3

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	129
			Gain:	76305996

LOW

3931	300050	3931
3929		
3929		
3930		
3929		
3931		
3930		
3931		
3931		
3931		
3930		
3931		
3931		
3932		
3930		
3932		
3932		
3931		
3929		
3930		

HIGH

	3176870	41632
41631		
41631		
41631		
41631		
41631		
41633		
41631		
41633		
41630		
41633		
41630		
41632		
41635		
41632		
41629		
41632		
41632		
41632		
41631		
41632		

CELL4

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	58
			Gain:	76312575

LOW

3932	300050	3931
3931		
3932		
3933		
3930		
3929		
3932		
3932		
3930		
3934		
3931		
3933		
3931		
3930		
3932		
3930		
3928		
3931		
3931		
3930		

HIGH	3176870	41629
41624		
41630		
41628		
41625		
41629		
41627		
41630		
41629		
41627		
41629		
41631		
41630		
41629		
41628		
41631		
41628		
41629		
41632		
41631		
41632		

CELL5

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	116
			Gain:	76310449

LOW			
3929	300050	3930	
3929			
3929			
3930			
3930			
3933			
3932			
3930			
3931			
3929			
3932			
3930			
3928			
3932			
3931			
3930			
3932			
3931			
3931			
3930			

HIGH	3176870	41629	
41627			
41630			
41632			
41628			
41626			
41628			
41631			
41628			
41627			
41626			
41629			
41629			
41629			
41631			
41630			
41632			
41629			
41632			
41631			
41632			

CELL6

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	153
			Gain:	76305895

LOW

3932	300050	3930
3929		
3932		
3930		
3929		
3932		
3927		
3931		
3929		
3930		
3930		
3929		
3928		
3933		
3929		
3933		
3929		
3931		
3931		
3930		

HIGH	3176870	41631
41632		
41631		
41632		
41630		
41633		
41630		
41631		
41629		
41632		
41633		
41630		
41633		
41629		
41630		
41632		
41632		
41633		
41632		
41631		
41632		

CELL7

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	112
			Gain:	76306869

LOW

3931	300040	3931
3932		
3932		
3931		
3930		
3930		
3929		
3932		
3931		
3930		
3932		
3930		
3932		
3927		
3931		
3930		
3929		
3931		
3931		
3930		

HIGH 3176870 41631

41631
41632
41629
41631
41631
41629
41632
41633
41633
41631
41631
41633
41630
41632
41633
41631
41630
41632
41631
41632

CELL8

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	125
			Gain:	76307375

LOW

3929	300040	3930
3931		
3930		
3931		
3931		
3931		
3930		
3932		
3931		
3932		
3929		
3929		
3928		
3930		
3930		
3932		
3929		
3931		
3931		
3930		

HIGH	3176870	41631
41630		
41630		
41632		
41630		
41634		
41631		
41631		
41633		
41629		
41632		
41630		
41632		
41631		
41631		
41630		
41631		
41629		
41629		
41631		
41632		

CELL9

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	176
			Gain:	76305148

LOW

3928	300040	3930
3929		
3931		
3932		
3930		
3930		
3928		
3927		
3933		
3929		
3930		
3930		
3930		
3929		
3929		
3928		
3930		
3931		
3931		
3931		

HIGH	3176870	41631
41632		
41631		
41628		
41631		
41631		
41634		
41633		
41633		
41632		
41626		
41632		
41630		
41631		
41633		
41630		
41633		
41632		
41633		
41632		
41632		

CELL10

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	114
			Gain:	76307375

LOW

3931	300040	3931
3931		
3930		
3931		
3932		
3929		
3932		
3930		
3930		
3928		
3931		
3931		
3931		
3932		
3930		
3930		
3928		
3931		
3931		
3931		

HIGH	3176870	41631
41632		
41630		
41630		
41630		
41631		
41632		
41631		
41634		
41631		
41631		
41632		
41631		
41630		
41632		
41629		
41630		
41631		
41632		
41630		
41632		

CELL11

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	181
			Gain:	76310840

LOW

3932	300250	3932
3933		
3932		
3932		
3932		
3932		
3932		
3932		
3932		
3932		
3931		
3933		
3932		
3932		
3933		
3932		
3932		
3932		
3933		
3933		

HIGH	3177100	41631
41632		
41632		
41631		
41632		
41631		
41632		
41632		
41631		
41630		
41632		
41629		
41632		
41631		
41632		
41632		
41630		
41632		
41631		
41631		
41631		

CELL12

Messwerte	Soll-Wert	Ist-Wert (Mittel)	Offset:	321
			Gain:	76308108

LOW

3931	300250	3931
3931		
3930		
3931		
3932		
3929		
3932		
3930		
3930		
3928		
3931		
3931		
3931		
3932		
3930		
3930		
3928		
3931		
3931		
3931		

HIGH

	3177100	41631
41632		
41630		
41630		
41630		
41631		
41632		
41631		
41632		
41631		
41631		
41632		
41631		
41630		
41632		
41629		
41630		
41631		
41632		
41630		
41632		

G. Konfigurationsdatei

G.1. Versuch 1

D:\config.txt **Sonntag, 23. Juni 2013 15:06**

```
Quantity Cells:  04
Cyclestep (sec): 0005

ADC Gain:        76308362 76307211 76305996 76312575 76310449 76305895 76306869 76307375
76305148 76307375 76310840 76308108
ADC Offset:      000096 000125 000129 000058 000116 000153 000112 000125 000176 000114 000181
000321

IP Address:      192.168.000.128

Measuretime (min):   05760

Relay Quantity:      22

Relaytime (min)      Relayselect
00001                01
00030                02
00060                00
00360                01
00390                02
00420                00
01000                01
01030                00
01200                02
01230                00
02500                01
02530                02
02560                00
03000                01
03005                00
03060                01
03085                00
03300                02
03330                00
04600                01
04630                02
04660                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
```

00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00
00000	00

G.2. Versuch 2

```
Quantity Cells:  04
Cyclestep (sec): 0002

ADC Gain:        76308362 76307211 76305996 76312575 76310449 76305895 76306869 76307375
76305148 76307375 76310840 76308108
ADC Offset:      000096 000125 000129 000058 000116 000153 000112 000125 000176 000114
000181 000321

IP Address:      192.168.000.128

Measuretime (min):   02400

Relay Quantity:      08

Relaytime (min)      Relayselect
00360                01
00390                00
00780                02
00795                00
01560                01
01575                00
01980                02
01995                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
```

```
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
00000                00
```

H. Kurzanleitung

Kurzanleitung

für den

Batterie Zyklierprüfstand

Inhaltsverzeichnis

1 Inbetriebnahme und Bedienoberfläche

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

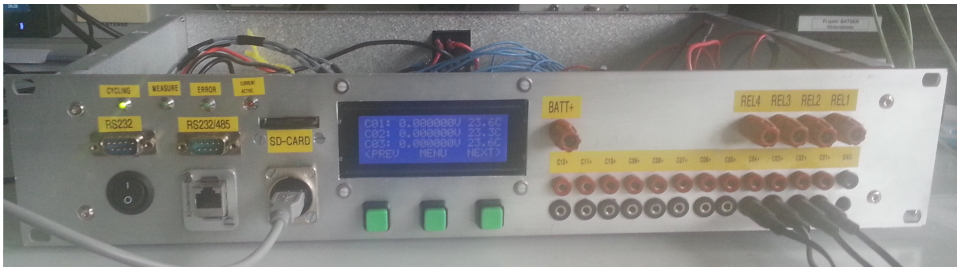


Abbildung 1: Ansicht Frontpanel

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. 3) abgelegt sein muss. Die Messdaten werden auf diese SD-Karte gespeichert. Die maximale Größe der SD-Karte ist auf 4 GB beschränkt. Die Speicherkarte muss im FAT16 oder FAT32 formatiert sein.

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

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

Tabelle 1: LEDs am Frontpanel

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. 4).

8.) Über ein ETHERNET-Anschluss kann über LAN eine Verbindung aufgebaut werden (Kap. 5)

Abbildung 3: Mittlerer Teil des Frontpanels

9.) Über ein LED-Display (Abb. 3 erfolgt eine Informationsausgabe. Das Display kann über 3 Taster bedient werden (Kap. 6).

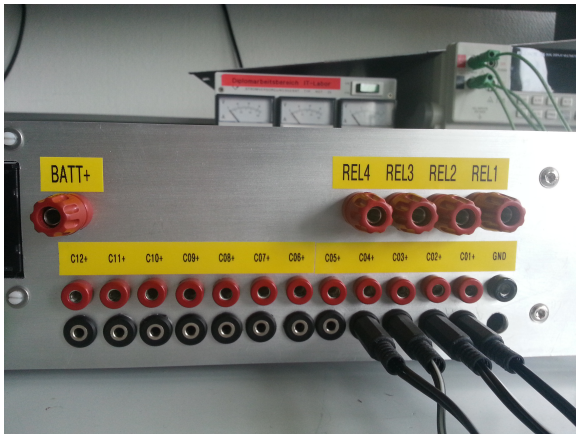


Abbildung 4: Rechter Teil des Frontpanels

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

2 Anschlussvorgaben Batterieprüfling

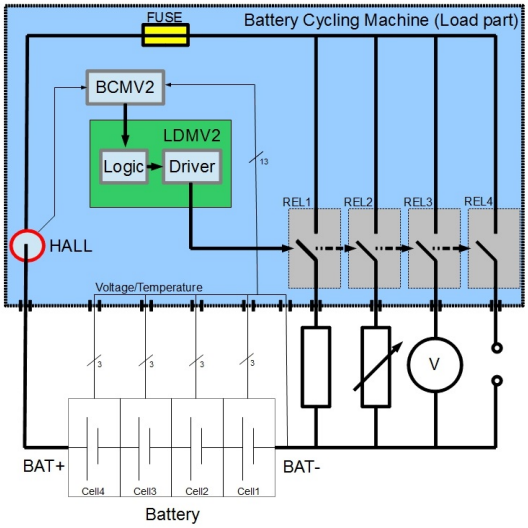


Abbildung 5: Interne Beschaltung der Lastrelais am Bsp. mit 4 Zellen

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 vom Ground der Batterie aus. Ground des Batterieprüflings wird als Null-Potential am GND-Messkontakt (schwarzen Buchse GND) angeschlossen.

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, falls vorhanden, dem positiven Pol angeschlossen.

5.) Ground von Batterieprüfling und externer Geräte werden außerhalb des Zyklierprüfplatzes zusammengeschaltet. Der Zyklierprüfplatz schaltet mit den Lastrelais das obere Spannungspotential.

6.) Der interne Lastkreis des Prüfstandes ist auf max. 20 A ausgelegt. Eine 20 A Sicherung ist im internen Lastkreis verbaut.

7.) Die NTC-Tempereturensensoren werden über einen Klinkenanschluss je Zelle angeschlossen und direkt an die jeweilige Batteriezelle montiert.

3 Konfigurationsdatei

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

2.) Ist keine Konfigurationsdatei vorhanden, oder kann diese nicht geöffnet werden, so werden für alle Parameter, bis auf die Kalibrierdaten, Nullwerte ins Programm geladen. Die Kalibrierdaten des ADC sind im Programm hinterlegt.

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

1. Die Anzahl der zu messenden Zellen (Quantity of Cells, dezimal zweistellig)
2. Zeitabstand zwischen den Messvorgängen in Sekunden (Cyclestep, vierstellig)
3. ADC Verstärkungsfaktor je Kanal (ADC Gain in $\frac{pV}{bit}$, achtstellig)
4. ADC Offsetwert je Kanal (ADC Offset in μV, sechsstellig)
5. Die IP-Adresse für die Ethernet Schnittstelle (IP Address, jedes Oktet dreistellig)
6. Die gesamte Dauer der Messung in Minuten (Measuretime, fünfstellig)
7. Anzahl der Zeitumschaltpunkte der Lastrelais (Relay Quantity, zweistellig)
8. Lastrelaisumschaltzeitpunkt in Minuten nach Programmstart (Relaytime, fünfstellig)
9. Zu schaltendes Lastrelais zur Umschaltzeitpunkt (Relayselect, zweistellig)

4.) Die Formatierung der Konfigurationsdatei muss eingehalten werden (Abb 6. Es dürfen keine Zeichen, Leerzeichen oder Tabs hinzugefügt oder entfernt werden!

5.) Liegt keine Konfigurationsdatei vor, kann vom Zyklierprüfplatz eine neue blanke Konfigurationsdatei erzeugt werden (z.B. über Bedienoberfläche UART, Kap. 4). Darauf gilt es die Konfigurationsdatei durch ersetzten der 0-Werte anzupassen.

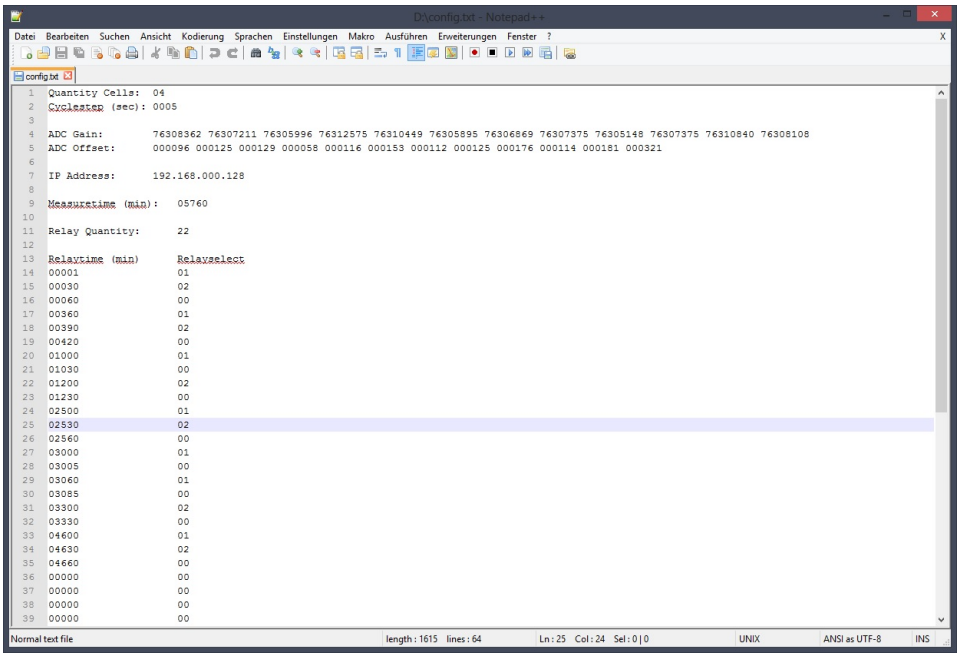


Abbildung 6: Format der Konfigurationsdatei

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 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.

4 Anbindung UART - hTerm

1.) Über die 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:

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

Tabelle 2: Einstellungen UART-Schnittstelle

2.) Es werden USB-Treiber für den Zyklierprüfstand benötigt. Die Treiber können auf der Homepge 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. 7). 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 7: Terminalausgabe im Menü ’Main’

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

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

Tabelle 3: Befehlssatz im Kontextmenü ’Main’

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.

Befehl	**Beschreibung**
help	Listet alle möglichen Befehle auf
h	Abkürzung für ’help’
?	Abkürzung für ’help’
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’

Tabelle 4: Befehlssatz im Kontextmenü ’SD-Card’

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

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 (01 - 12)
cyclestep	Setzte Zeitabstand zwischen zyklischer Messung in Sekunden (0000-9999)
gain	Setze kanalabhängig Verstärkungsfaktor für ADC (z.B. ’gain 01 76273500’)
offset	Setze kanalabhängig Offset für ADC (z.B. ’offset 01 000120’)
ip	Setze IP-Adresse (’192.168.000.128’) ⇒ Neustart erforderlich
relais	Manuel Lastrelais aktivieren (z.B. ’relais 1’)
relsel	Wähle für die Ablaufsteuerung an entsprechender Stelle und Zeitpunkt ein Lastrelais(z.B. ’relsel 00 00001 1’)
relque	Setze die Anzahl der Lastrelaisumschaltzeiten (01 - 99)
time	Setze Laufzeit der gesamten Messung in Minuten (0-99999)
save	Speichere aktuelle Konfiguration auf die SD-Karte (überschreibt config.txt)
exit	Zurück zum Kontextmenü ’Main’

Tabelle 5: Befehlssatz im Kontextmenü ’Config’

9.) Über die Befehle geänderten Konfigurationsparameter werden nicht automatisch auf der SD-Karte gespeichert.

10.) 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.

5 Anbindung Ethernet - MatLab

1.) Über MatLab kann eine TCP/IP Verbindung mit dem Controller 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 192.168.0.128

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 "$send\_cmd(cons, sprintf('cmd\backslash n'))$" können Befehle (cmd) an die Zykleirmaschine über Ethernet geschickt werden. Dabei können die selben Befehler verwendet werden wie über UART (Kap. 4).

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.

6 LED-Display und Funktion

(a) LED-Zustand: START

(b) LED-Zustand: CLOCK

(c) LED-Zustand: MEAS

(d) LED-Zustand: CELLA

(e) LED-Zustand: CURRENT

(f) LED-Zustand: IP

(g) LED-Zustand: CONF

Abbildung 8: Zustands-bedingte Ausgabe LED-Display

1.) Über das LED-Display des Zyklierprüfstandes können Informationen 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 - 8g geschaltet werden.

4.) Im Menü 'Measurment Cycling' (Abb. 8c) kann über den linken Taster "START" die zyklische Messdatenerfassung direkt gestartet werden.

5.) Im Menü 'Measurment Cycling' (Abb. 8c) 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.

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, 4. Juli 2013	
Ort, Datum	Unterschrift