Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Nico Rieckmann

Entwicklung einer Regelsoftware auf Linux-Basis für eine optische Batterie-Analyse

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

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

Nico Rieckmann

Entwicklung einer Regelsoftware auf Linux-Basis für eine optische Batterie-Analyse

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. Robert Heß

Abgegeben am 22. Juni 2016

Nico Rieckmann

Thema der Bachelorthesis

Entwicklung einer Regelsoftware auf Linux-Basis für eine optische Batterie-Analyse

Stichworte

Linux, Raspberry Pi, digitale Regelung, Unix-Sockets, modulare Software, digital geregelte Batterie-Zyklierung, optische und elektrische Messdatenaufnahme

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung und den Aufbau eines eingebetteten Messsystems für die Erforschung von Lithium-Ionen-Batterien. Deren Elektroden sollen mit einer Mikroskopkamera auf die Veränderung ihres optischen Spektrums während eines Lade- und Entladevorgangs untersucht werden. Diese Erkenntnisse ermöglichen genaue Aussagen über deren Ladezustand und Gesundheitszustand.

Nico Rieckmann

Title of the paper

Development of an Linux based regulation software for an optical battery measured station

Keywords

Linux, Raspberry Pi, digital controlling, Unix-Sockets, modular software, digital controlled battery-cycling, optical and electrical data acquisition

Abstract

This work includes the development and setup of an embedded measurement system to make a research of Lithium-ion batteries possible. The battery electrodes are to be examined with a microscope camera to a change in their optical spectrum during charging and discharging. These findings allow precise statements about the state of charge and state of health.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog Digital Converter
cC	constant Current charging mode
cCcV	constant Current const Voltage charging mode
cV	constant Voltage charging mode
DAC	Digital Analog Converter
FIFO	First in first out
FPU	Floating Point Unit
FSR	Full Scale Range
GHD	Gewerbe-Handel-Dienstleistungen
GUI	General user interface
IC	Integrated Circuit
IDE	Integrated development environment
IPC	Inter-process communication
ITO	Indiumzinnoxid
LAN	Local area network
LDO	Low Dropout Regulator
LDVR	Low Rrop Voltage Regulator
LED	Light-emitting diode
LSB	Least significant bit
MISO	Master input slave output
MOSI	Master output slave input
PWM	Pulsweitenmodulation

RGB	Red Green Blue
RPI2	Raspberry Pi 2
SMD	Surface Mounted Device
SNR	Signal Noise Ratio (Signal Stör Abstand)
SOA	Service-oriented architecture
SOC	State of charge
SOH	State of health
SPI	Serial Peripheral Interface
sudo	Superuser root
TWh	Terawattstunde
OOP	Objektorientierte Programmierung

Verwendete Formelzeichen

Hardwarespezifische Größen

U_{Ref} : Referenzspannung
U_{LSB} : Spannung des LSB
U_{out} : Ausgangsspannung
U_{in} : Eingangsspannung
G : Verstärkung
R : ohmscher Widerstand
C : Kapazität
D : Digitalwert
n : Anzahl Bits
f_g : Grenzfrequenz

Software Regelungs Größen

I_{heu} : heuristisch errechneter Strom
I_{max} : Maximalstrom
U_{Soll} : Soll-Spannung
U_{Ist} : Ist-Spannung
R_{ges} : Gesamtwiderstand
R_{mess} : Messwiderstand
R_{tol} : Toleranzwiderstand

Kennlinien Korrektur Größen

m : Steigung
b : Offset

1. Einleitung

1.1. Motivation

Aus den in der Natur vorkommenden Rohstoffen (Primärenergie) wird durch Umwandlungsprozesse die Endenergie in Form von Stein- u. Braunkohle, Mineralölprodukte, Gase, Strom, Fernwärme, erneuerbare Wärme und sonstige Energieträger erzeugt. Dabei emittieren immer auch Schadstoffe. Diese anthropogenen Schadstoffe stellen eine stetig wachsende Problematik für unsere Umwelt und Gesundheit dar.

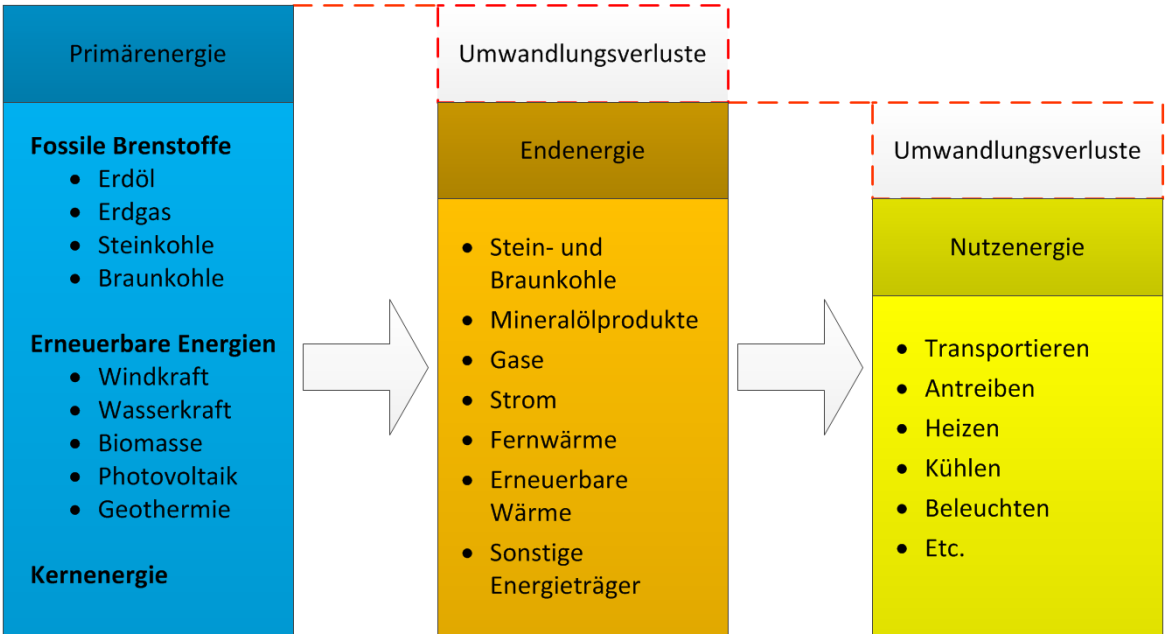

Abbildung 1.1.: Schematische Darstellung des Zusammenhangs von Primär-, End- und Nutzenergie sowie der zugehörigen Umwandlungsverluste [27]

Diese Endenergie wird durch weitere Umwandlungsprozesse genutzt, z.B. zum Transportieren, Antreiben, Heizen, Kühlen, Beleuchten usw.. Dabei entstehen neben der genutzten Energie viele unterschiedliche Schadstoffe.

In Abbildung 1.2 wird der Endenergieverbrauch in die vier Sektoren Industrie, Haushalte, Gewerbe-Handel-Dienstleistungen (GHD) und Verkehr unterteilt.

Endenergieverbrauch 2014 nach Sektoren und Energieträgern

Endenergieverbrauch	Industrie	Haushalte	GHD	Verkehr	Gesamt	Gesamt Anteile
Stein- und Braunkohle	114 TWh	6 TWh	0 TWh	0 TWh	120 TWh	5,0 %
Mineralöl-Produkte	21 TWh	144 TWh	79 TWh	684 TWh	928 TWh	38,6 %
Gase	239 TWh	218 TWh	104 TWh	2 TWh	563 TWh	23,4 %
Strom(inkl. Eneuerbare Energien)	224 TWh	130 TWh	143 TWh	12 TWh	509 TWh	21,2 %
Fernwärme	52 TWh	42 TWh	15 TWh	0 TWh	108 TWh	4,5 %
Erneuerbare Wärme	30 TWh	75 TWh	20 TWh	32 TWh	157 TWh	6,6 %
Sonst. Energieträger	17 TWh	0 TWh	0 TWh	0 TWh	17 TWh	0,7 %
Summe	697 TWh	615 TWh	361 TWh	730 TWh	2.402 TWh	100,0 %

Abbildung 1.2.: Endenergieverbrauch 2014 nach Sektoren und Energieträgern [12]

Betrachtet man die Spalte „Gesamt", so fällt auf, dass die Mineralölprodukte mit 928 TWh den größten Anteil am Endenergieverbrauch ausmachen. Bei weiterer Betrachtung wird ersichtlich, dass der Sektor „Verkehr" mit 684 TWh am Verbrauch der Mineralölprodukte beteiligt ist. Das entspricht einem Anteil von 73,7%. Mit steigendem Wirtschaftswachstum steigt auch die Zahl der am Verkehr beteiligten Fahrzeuge. Am 01.01.2016 emittierten allein aus den in Deutschland zugelassenen (54.576.939)[1] Kraftfahrzeugen neben Lärm auch viele Schadstoffe [18] durch Verbrennungsmotoren. Darunter sind Schadstoffe wie Benzol, Dieselrußpartikel, Stickstoffoxide, Kohlenwasserstoffe, Kohlenmonoxid und Kohlendioxid [26]. Diese anthropogenen Schadstoffe können von der Natur nicht so schnell abgebaut, bzw. umgewandelt werden, wie sie entstehen. Kohlendioxid hat z.B. eine durchschnittliche Verweildauer von 120 Jahren [25]. Das bedeutet eine stetig wachsende Zunahme der anthropogenen Schadstoffe. Soll eine für uns Menschen gesunde Umwelt erreicht und erhalten werden, so muss die Entstehung anthropogener Stoffe drastisch verringert werden.
Dies versucht die Bundesregierung im Rahmen des „Nationalen Entwicklungsplans Elektromobilität" anzugehen. Darin heißt es unter anderem „Elektrofahrzeuge können die Städte von Schadstoffen, Feinstaub und Lärm befreien und so die Lebensqualität steigern"; und weiter heißt es „Ziel des Nationalen Entwicklungsplanes Elektromobilität ist es, die Forschung und Entwicklung, die Marktvorbereitung und die Markteinführung von batterieelektrisch betriebenen Fahrzeugen in Deutschland voranzubringen" [13].

[1] 54.602.441 Kfz minus 25.502 Elektro-Pkw = 54.576.939 Schadstoff emittierende Kfz

Diese Bachelorthesis soll im Rahmen der Forschungsaktivitäten der Arbeitsgruppe „Batteriesensorik" an der HAW Hamburg, die Analyse und die daraus resultierende Weiterentwicklung von Akkumulatoren vorantreiben. Dadurch sollen Nachteile, wie hohe Kosten und geringe Reichweite, der Elektrofahrzeuge minimiert werden. Wenn die elektrische Energie der Akkumulatoren auch noch von erneuerbaren Energieträgern stammt, verspricht dieser Weg eine nachhaltige und saubere Umwelt.

1.2. Einarbeitung in die BATSEN Gruppe

Um Fahrzeugbatterien zu verbessern und sie im Hinblick auf Wirtschaftlichkeit, Verfügbarkeit und Sicherheit konkurrenzfähiger gegenüber Verbrennungsmotoren werden zu lassen, wurde hierzu an der HAW das Forschungsprojekt „Drahtlose Zellensensoren für Fahrzeugbatterien BATSEN" eingerichtet.
Im Rahmen dieses durch das Bundesministerium für Bildung und Forschung geförderten Forschungsprojektes sind bereits verschiedene drahtlose Zellensensoren zur elektrischen Überwachung einzelner Zellen einer Batterie entstanden [22]. Auch Elektroden von $LiFePO_4$-Zellen wurden bereits optisch betrachtet und der Nachweis, dass sich optische Veränderungen am Elektrodenmaterial messtechnisch erfassen lassen, eruiert [23]. Es folgten umfangreiche Untersuchungen [15][21] der Zellchemie, der Materialien für Batteriekomponenten, Vergleiche des Materials für Kathoden und Anoden sowie deren optischen Effekte. Entwurf und Aufbau eines Messplatzes mit Kamera, Spektrometer sowie der modular gestalteten Steuer- und Analysesoftware für die optische Zustandsbetrachtung von Lithiumzellen war Gegenstand der Bachelorthesis [15] von Herrn Griessbach. Weitere Studienarbeiten folgten und führten auch immer zur Weiterentwicklung und Optimierung des Messplatzes. So zum Beispiel die Entwicklung von Verfahren mittels UNIX-Filtern, um die Verarbeitung großer Datenmengen zu ermöglichen und damit den Zusammenhang zwischen optischen und elektrischen Messwerten belegen zu können [21]. Durch die Bachelorarbeit von Herrn Geist erhielt der Messplatz ein Steuersystem auf Basis einer entwickelten Messplatine und einem Einplatinen-Computer [14]. Damit sollen alle nötigen Prozesse wie die Bildverarbeitung, die Erfassung von Spannung, Strom und Temperatur sowie der Prozess zur Zyklier-Ablaufsteuerung erfaßt werden. Dieses Steuersystem wurde in den Messplatz eingebettet und ersetzt die bisherigen PCs.

2. Analyse des bestehenden Zykliersystems

Das aus vorheriger Arbeit von Herrn Geist [14] entwickelte Zykliersystem besteht aus einer Mess-und Regel-Software mit zugehöriger Hardware aus Mess- und Stellgliedern. Die Plattform auf der die Regel-Software ausgeführt wird ist ein Raspberry Pi 2 (RPI2). Es ist ein Einplatinen-Computer auf dem das Betriebssystem Raspbian aufgesetzt ist, welches ein Derivat der Linux-Distribution Debian ist.
Die Aufgabe des gesamten Systems ist es, eine 3,3V Lithium-Eisenphosphat ($LiFePO_4$) Forschungszelle über längeren Zeitraum zyklisch zu laden und zu entladen. Permanent wird die Zelle überwacht und gemessen. Dabei interessieren elektrische Werte, Temperatur-, wie auch optische Spektralwerte, die danach aus den Fotos einer Mikroskopkamera gerechnet werden. Diese fotografiert alle 10 Sekunden das Elektrodenfenster der Forschungszelle.
Ziel ist es, der Forschung ein Messsystem bereit zu stellen, mit dem ein korrelativer Zusammenhang von RGB-Farbwerten, elektrischen Werten und dem Ladezustand (SOC) sowie dem Gesundheitszustand (SOH) der Zelle untersucht werden kann.

Eine schematische Übersicht des Systems ist in Abildung 2.1 veranschaulicht.

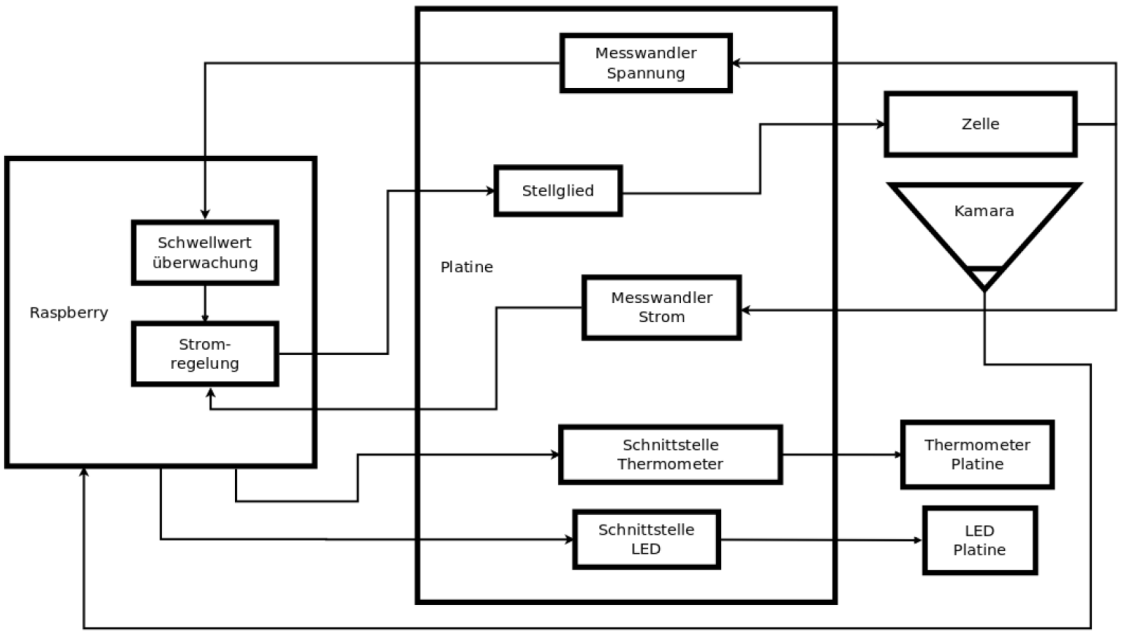

Abbildung 2.1.: Übersicht des Zykliersystem-Aufbaus nach [14]

Die Anforderungen an die Lade-/Entladeregelung des Zykliersystems sind in Tabelle 2.1 mit nachfolgenden Zusätzen aufgelistet.

Parameter	**Symbol**	**Wert**
Maximaler Lade-/Entladestrom	$I_{Z\,max}$	1 mA
Kleinste Zellspannug	$U_{Z\,min}$	0 V
Höchste Zellspannung	$U_{Z\,max}$	4,5 V
Schrittweite Stromvorgabe	$\Delta I_{Z\,step}$	1 μA

Tabelle 2.1.: Anforderungen an das Zykliersystem nach [14]

Zusätzlich muss das System auch folgende Aufgaben erfüllen:

- Überwachung der Zellspannung sowie des Lade-/Entladestroms.
- Überwachung der nahen Umgebungstemperatur der Zelle.
- Steuerung einer RGB-LED, welche zur Beleuchtung des Objekts für die Mikroskopkamerabilder dient.

Der Anschlussplan von den Pins der Messplatine zum RP2 Pinheader ist aus den Abbildungen 2.2 und 2.3 folgender Darstellung zu sehen.

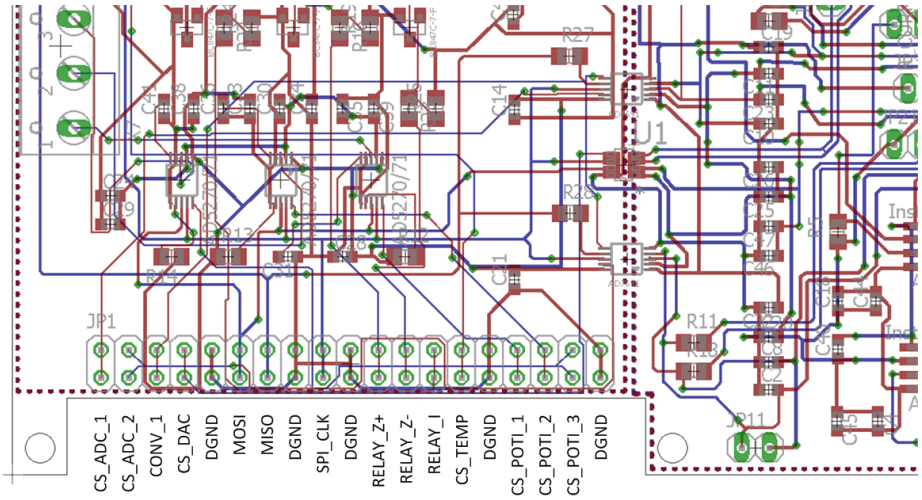

Abbildung 2.2.: Pinheader der Messplatine

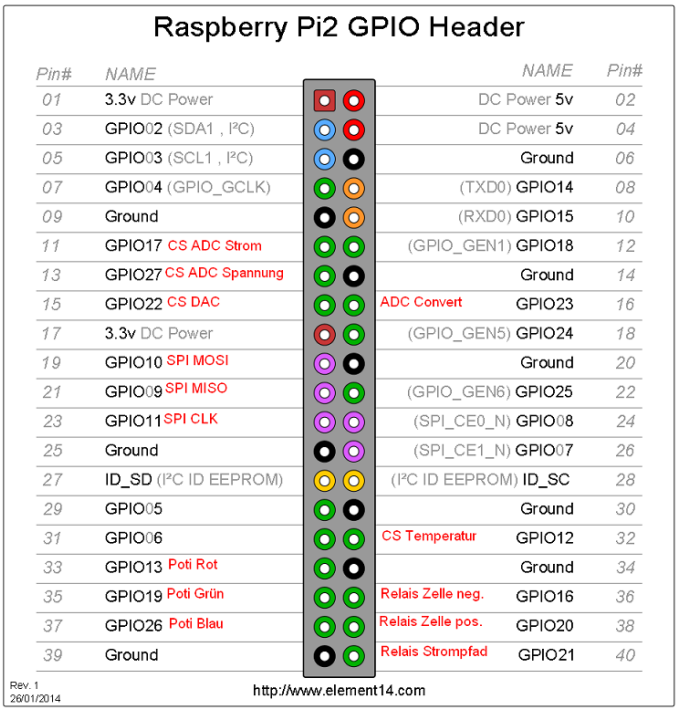

Abbildung 2.3.: Pinheader des Raspberry Pi 2 nach [14]

2.1. Bestandsaufnahme der Hardware

Die von Herrn Geist entwickelte Messplatine [14] macht den umfangreichsten Teil seiner Arbeit aus. Es ist eine Analog- sowie Digitalschaltung in SMD-Technologie auf ihr verlötet. Realisiert ist ein Lade-/Entladekreis mit einem DAC als Stellglied und zwei ADC als Messglieder, ein Interface für eine externe Temperaturmessung und noch eine LED-Steuerung über ein 8 bzw. 10 Bit Digitalpotentiometer. Ihre Maße betragen 100x80mm.

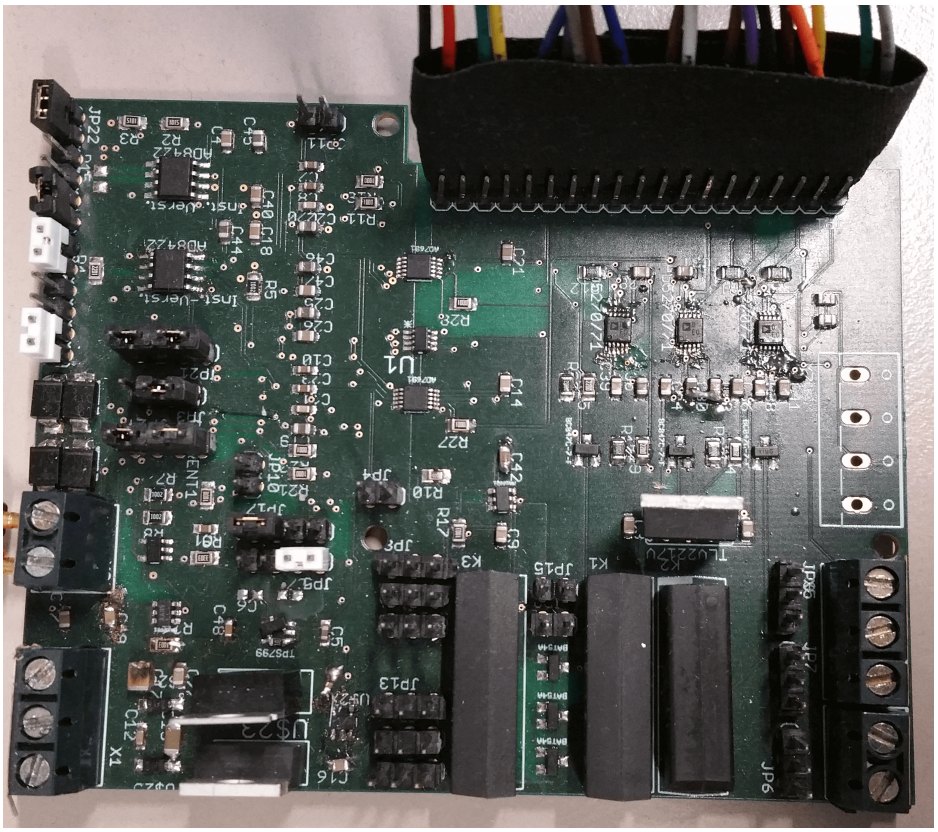

Abbildung 2.4.: Messplatine von Torsten Geist

Ein vereinfachtes Schaltbild, auf dem nur die wichtigsten Stell- und Messglieder abgebildet sind, ist auf folgendem Bild 2.5 zu sehen.

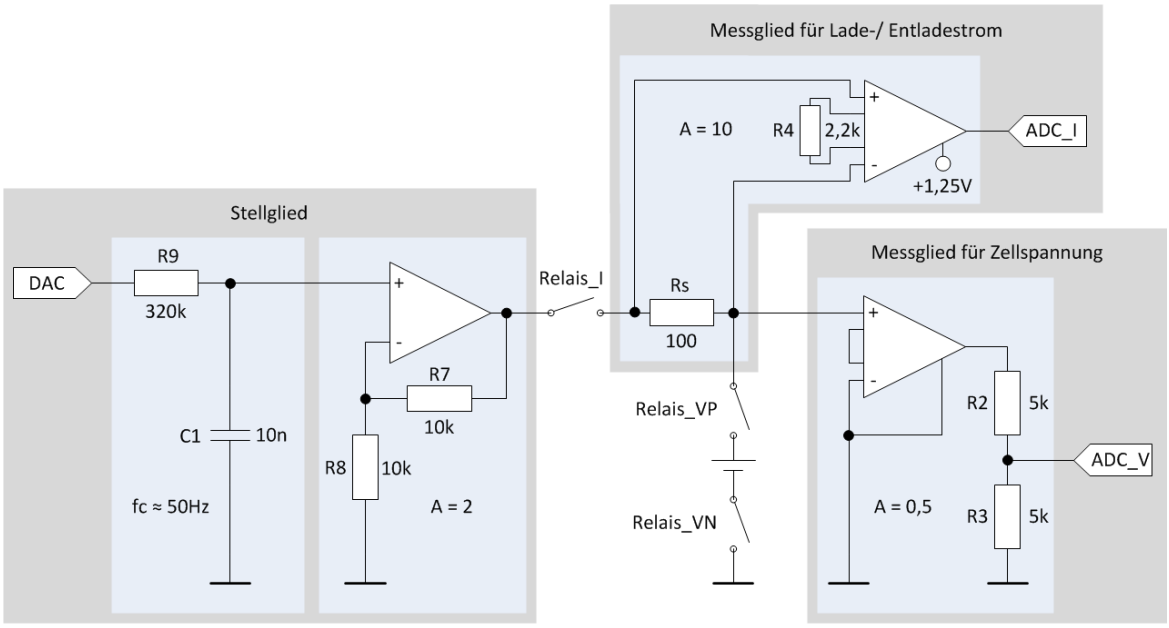

Abbildung 2.5.: Übersicht Zykliersystem

Die zu messende Batteriezelle wird mittels Relais zweipolig zugeschaltet. Beim Lade-/Entladevorgang gibt das Stellglied eine Spannung vor, auf die sich die Zelle asymptotisch anpassen wird. Durch den dadurch entstehenden Lade-/Entladestrom entsteht ein Spannungsabfall an dem 100Ω Messwiderstand R_S. Dieser ist zwecks Strommessung vorgesehen.

Dabei ist der analoge Teil vom digitalen Teil schaltungs- sowie flächentechnisch getrennt. Das verhindert hochfrequente Einstreuungen, die von digitalen Schaltungselementen in die empfindlicheren analogen Messschaltungsbereiche gelangen. Diese könnten zu mehr Ungenauigkeiten oder gar funktionalen Störungen führen.

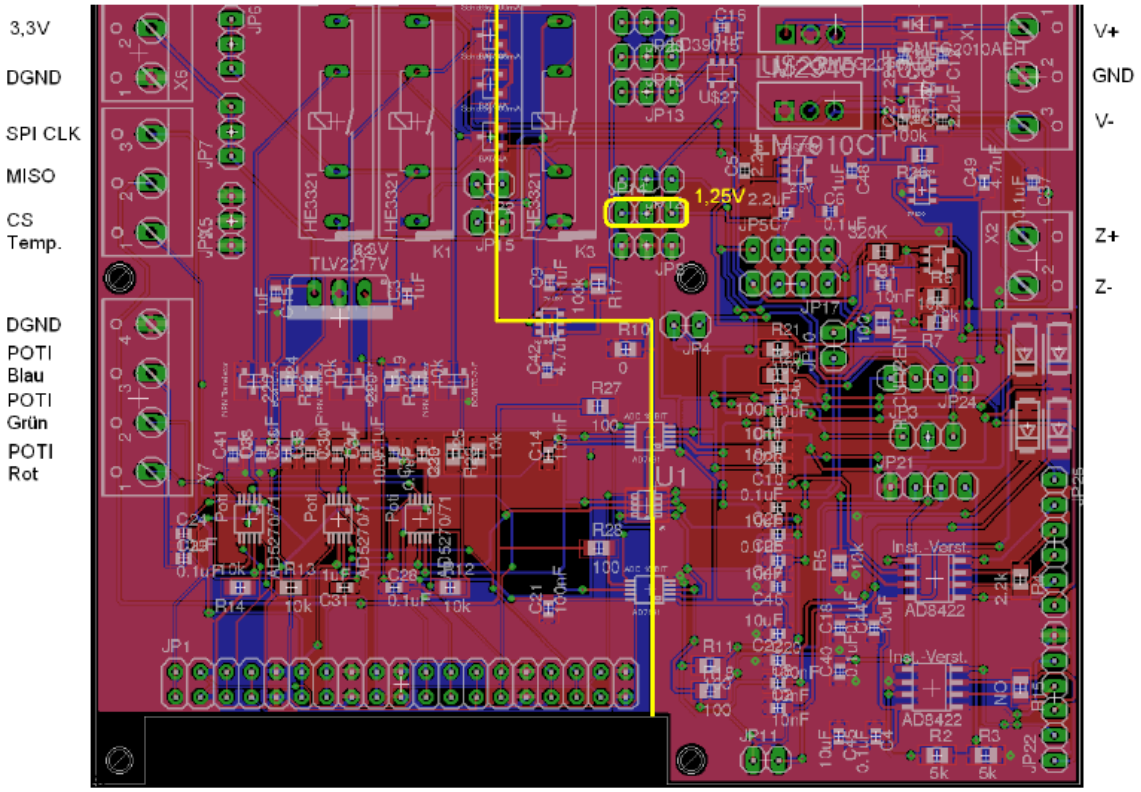

Abbildung 2.6.: Layout der Messplatine nach [14]

Analogschaltungen sind empfindlicher, weil ihr kontinuierlicher Wertebereich theoretisch unendlich viele Werte annehmen kann. Das führt bei einem geringen Signal-Stör-Abstand (SNR) zu messbaren Verfälschungen. Digitale Komponenten hingegen arbeiten unter den gleichen Bedingungen unbeeinflusst weiter, da sie eine Änderung des Eingangssignals erst ab einem bestimmten Schwellwert als high bzw. low interpretieren. Die analoge und die digitale Masse sind an einer Stelle der Platine sternförmig verbunden.

2.1.1. Steuerbare Spannungsquelle als Stellglied

Das Stellglied setzt sich zusammen aus einem DAC, einem nachgeschalteten Tiefpassfilter und einem nichtinvertierenden Verstärker. Der DAC „AD5680" [4], von dem Hersteller Analog Devices, ist für eine Aussteuerung von $0\,\mathrm{V}$ bis $U_{Ref} - U_{LSB}$ mit einer Auflösung von 16 Bit ausgelegt. Davon sind 12 Bit Genauigkeit garantiert. Auf der Messplatine wird die analoge Seite des DAC mit einer Betriebsspannung von 5V gegen Masse und die digitale

Seite des SPI-Interfaces mit 3,3V gegen Masse versorgt. Seine Referenzspannung bekommt er von dem rauscharmen 2,5V LDO Regler TPS799 [11] von Texas Instruments. Die DAC Ausgangsspannung berechnet sich wie folgt:

$$U_{out} = U_{ref} * \left(\frac{D}{2^n}\right) = 2,5V * \left(\frac{D}{2^{16}}\right) \tag{2.1}$$

Um Rauschen des DAC Ausgangssignals zu reduzieren, ist der RC-Tiefpassfilter mit ca. 50Hz Grenzfrequenz in Serie geschaltet. Dieser Tiefpass besteht aus einem $R_9 = 320k\Omega$ SMD Widerstand und einem $C_1 = 10nF$ SMD Kondensator. Seine Grenzfrequenz bei -3dB ergibt sich in Formel 2.2.

$$f_g = \frac{1}{2\pi * R_9 * C_1} = \frac{1}{2\pi * 320k\Omega * 10nF} = 49,736Hz \tag{2.2}$$

Der nichtinvertierende Verstärker besteht aus einem Rail-To-Rail OPA743 [6] Operationsverstärker von dem Hersteller Texas Instruments. Dieser verstärkt das DAC-Ausgangssignal um den Faktor $G_{OP} = 2$ und dient als Lade- und Entlade-Stromtreiber der jeweils angeschlossenen Batteriezelle. In der obigen Schaltung der Messplatine wird er mit 10V gegen Masse betrieben. Seine Verstärkung berechnet sich wie folgt:

$$G_{OP} = \frac{U_{out}}{U_{in}} = 1 + \frac{R_7}{R_8} = 1 + \frac{10k\Omega}{10k\Omega} = 2 \tag{2.3}$$

2.1.2. Strommessung

Um den sehr kleinen Lade-/Entladestrom von 1μA bis 1mA messen zu können, ist etwas mehr Hardware-Aufwand notwendig. Gängig für eine Strommessung ist es, an einem niederohmigen Shuntwiderstand eine geringe Spannung abfallen zu lassen und diese ggf. zu verstärken, zu messen und in einen Stromwert umzurechnen. Da die Ströme in diesem Fall so klein sind, dass eine abfallende Spannung ansonsten zu stark verstärkt werden müsste, was mit Verstärker-Rauschen einhergehen würde, ist der Shuntwiderstand 100Ω groß. Das erlaubt dem Rail-To-Rail Instrumentenverstärker AD8422 [9], von dem Hersteller Analog Devices, mit einer Verstärkung von nur $G = 10$ zu arbeiten, indem seine Gainpins mit einem $2,2k\Omega$ Widerstand beschaltet sind. Dies lässt sich wie folgt berechnen [9]:

$$R_G = \frac{19,8k\Omega}{G-1} = \frac{19,8k\Omega}{10-1} = 2,2k\Omega \tag{2.4}$$

Der 18 Bit Strom-ADC AD7691[8] von Analog Devices wird analogseitig mit 5V und digitalseitig mit 3,3V gespeist. Seine Referenzspannung bekommt er vom selben 2,5V LDO-Regler wie der DAC. Da der Strom-ADC zwar differentielle Eingänge hat, aber schaltungstechnisch unipolar betrieben wird, ist das Bit für das Vorzeichen obsolet. Dem Instrumentenverstärker wird durch einen 1,25V LDO-Regler ein Offset gespeist, der den Stromnullpunkt im Wertebereich des ADC auf die halbe Fullscale anhebt.

2.1.3. Spannungsmessung

Für die Messung der Zellspannung werden ein gleicher Instrumentenverstärker und gleicher ADC wie bei der Strommessung verwendet. Allerdings sind die Gainpins des Instrumentenverstärkers mit keinem Widertand gebrückt, sodass die Spannung mit einer Verstärkung von $G = 1$ am Verstärkerausgang anliegt. Dies berechnet sich wie 2.4 schon mit:

$$G = 1 + \frac{19.8k\Omega}{R_G} = 1 + \frac{19.8k\Omega}{\infty} = 1 \tag{2.5}$$

Der nach dem Ausgang liegende Spannungsteiler mit einem Widerstandsverhältnis von 1:2 halbiert die Ausgangsspannung des Instrumentenverstärkers mit:

$$U_{out} = \frac{U_{in} * R_3}{R_2 + R_3} = \frac{U_{in} * 5k\Omega}{10k\Omega} = \frac{U_{in}}{2} \tag{2.6}$$

Der Spannungs-ADC wird betriebs- und referenzspannungsmäßig gleich betrieben wie der Strom-ADC. Damit hat auch dieser einen Eingangsspannungsbereich von 0V bis $U_{Ref} - U_{LSB}$. Auch hier beträgt $U_{Ref} = 2,5V$.

Eine Anmerkung zum Stellglied ist noch, dass über eine Jumperkonfiguration die Referenzspannung des DAC von 2,5V auf 5V verdoppelt werden kann. Das schafft die Möglichkeit, eine Zelle mit noch höheren Strömen als 1mA zu laden. Jedoch müssten die Referenzspannungen der ADC auch verdoppelt werden. Dies ist aber über Jumper nicht möglich. Deswegen wird diese Option nicht verwendet.

2.1.4. Temperaturmessung

Die Temperaturmessung wird von dem Sensor-IC LM74 [10] von Texas Instruments übernommen. Dieser wird sehr nahe an die zu messende Zelle befestigt. Der IC hat ein Gehäuse in SOIC-8 SMD-Bauform und wird von einem 3,3V LDO versorgt. Über auf der Messplatine vorgesehene SPI-Bus Leitungen und Klemmen, wird der Sensor mit der Messplatine und letztendlich dann mit dem Raspberry Pi 2 verbunden. Mit einer Auflösung von 13 Bit inklusive Vorzeichen Bit entspricht das LSB einer Temperatur von 0,0625°C. Ein Aufbau mit dem Temperatursensor gab es zu dem Zeitpunkt der Projektübergabe noch nicht.

2.1.5. Steuerung der RGB-LED

Um Zusammenhänge von dem optischen Farbspektrum der Photozelle mit ihrem Ladezustand (SOC) und ihrem Gesundheitszustand (SOH) beobachten zu können, wird die Photozelle mit einer Mikroskopkamera fotografiert. Die Reflexionen an der Zelloberfläche werden von einer RGB-LED Beleuchtung, die ringförmig um die Kameralinse montiert wurde, verursacht. Mit Matlab werden anschließend die Rot-, Grün- und Blauanteile im Farbspektrum herausgerechnet und geplottet.

Aus historisch gewachsenen Gründen werden die RGB-LED nicht direkt von den PWM-Pins des Raspberry Pi 2 angesteuert, weil sie bereits mit einem IC als PWM-Stufe ausgestattet sind. Diese wandeln linear ihre Eingangsspannung in einen PWM-Tastgrad um und treiben die RGB-LED. Die Eingangsspannung liegt jeweils im Bereich von 0V bis fast 5V und wird von drei digitalen Potentiometern auf der Messplatine eingestellt. Durch diesen einstellbaren Spannungsteiler wird die LED-Eingangsspannung jedoch nicht linear angesteuert.

$$U_{LED} = \frac{R_{in} * U_{ges}}{R_{in} + R_{Poti}} \qquad \text{mit } R_{Poti} \in \{0..R_{WA}\} \tag{2.7}$$

Die digitalen Potentiometer AD5270 [3] bzw. AD5271 [3] von dem Hersteller Analog Devices haben eine Auflösung von 8 Bit und 10 Bit und sind im MSOP-10 Gehäuse. Ihr Maximalwiderstand R_{WA} lässt sich über SPI-Bus einstellen in $20k\Omega$, $50k\Omega$ und $100k\Omega$, der dann in 256, bzw. 1024 einzelne Schleiferpositionen unterteilt wird.

AD5270:

$$R_{WA}(D) = \frac{D}{256} * R_{WA} \tag{2.8}$$

AD5271:

$$R_{WA}(D) = \frac{D}{1024} * R_{WA} \tag{2.9}$$

2.1.6. Spannungsversorgung

Auf der Platine sind mehrere Spannungsebenen vorhanden, die zur Versorgung einzelner IC und Bauteile sowie zur Referenzvorgabe gemessener physikalischer Größen dienen. Diese sind mittels kaskadierten Spannungsreglern, sogenannte LDO, realisiert.

Die Platine benötigt eine bipolare Eingangsspannung zwischen 12V und 26V und für die negative Versorgung -12V bis -28V. Diese wird von einem LM2940T-10.0 [5] und einem LM7910CT [1] LDO zur Erzeugung der 10V und -10V Ebene benötigt. Die -10V Ebene versorgt die negativen Betriebsspannungen der zwei Instrumentenverstärker. Der 10V Regler versorgt die positive Betriebsspannungen der beiden Instrumentenverstärker, einen Operationsverstärker und zwei 5V Regler für den analogen und digitalen Schaltungsteil. Diese sind wie unter Kapitel 2.1 beschrieben an ihren Massen nur an einem Punkt miteinander verbunden, damit der empfindliche Analogteil nicht von hochfrequenten digitalen Signalen beeinflusst wird. So versorgt der 5V Regler die analoge Seite der Mess- und Stellinstrumente, einen 2,5V und einen 1,25V Regler, die jeweils als Referenzspannungsquellen dienen. Der 5V Regler für den digitalen Teil versorgt die Relais, die abfallende Spannung an den Potentiometern und die Digitalseite aller SPI-Busteilnehmer.

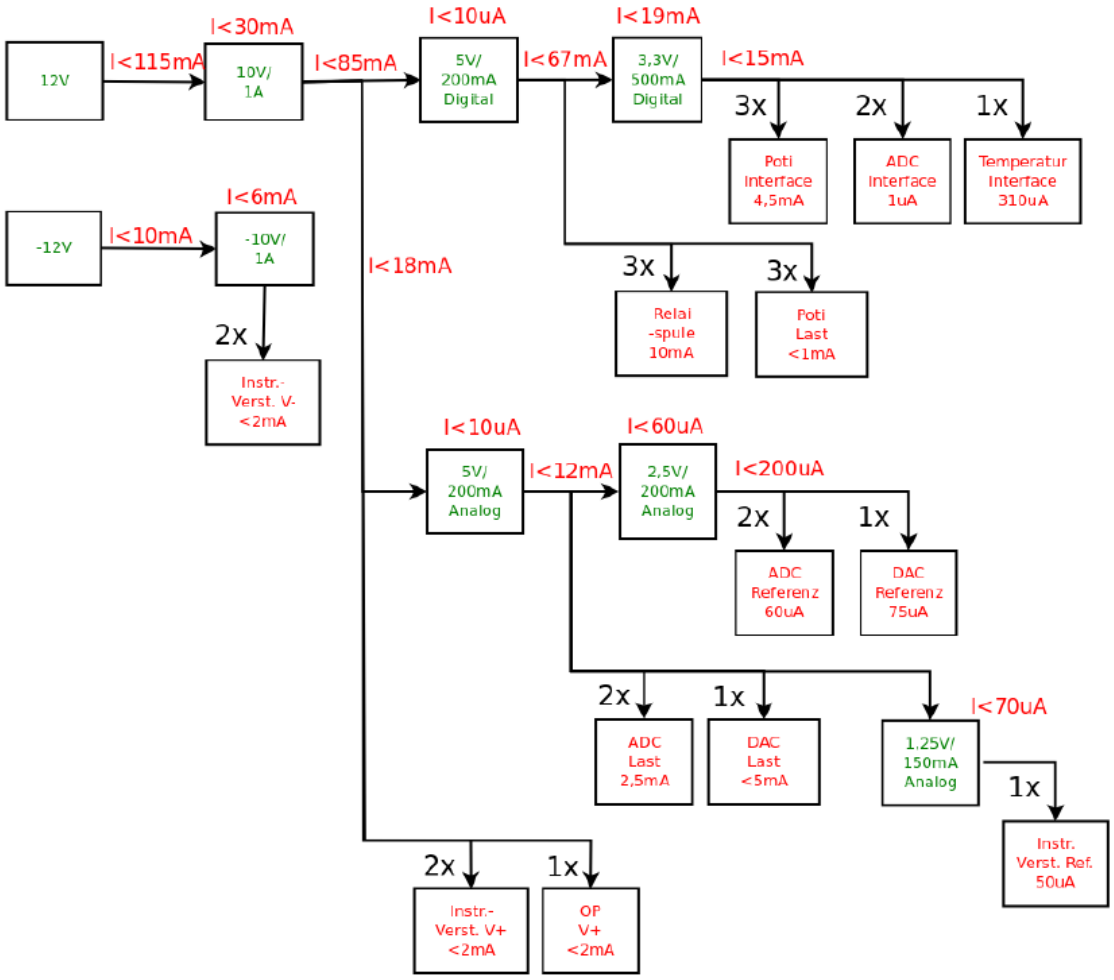

Abbildung 2.7.: Spannungsebenen der Messplatine nach [14]

Die Wärmeberechnung wird hier nicht durchgeführt, da sie bereits in der Thesis von Herrn Geist in [14] ab Kapitel 3.3 „Spannungsversorgung” beschrieben ist.

2.2. Bestandsaufnahme der Software

Die Messwerterfassung, die Stromregelung während der Zellzyklierung, die Verwaltung der Messdaten sowie die Automatisierung eines Zyklierablaufplans wird von Software übernommen. Wie schon erwähnt, operiert diese auf der Open Source Plattform Raspbian, ein Betriebssystem des Raspberry Pi 2. Wie dieses System neu aufgesetzt und eingerichtet wird, ist in der Abschlussarbeit von Torsten Geist [14] unter Kapitel 4.1 „Konfiguration des Linux-Betriebssystems" ausführlich beschrieben und nicht Teil dieser Arbeit.

Es wird ein entwickeltes C-Programm sowie Open Source Software verwendet. Die Regelung und Überwachung der elektrischen Batterieparameter werden von dem Mess- und Regelprogramm namens „Steuerung" übernommen. Die Ansteuerung der Mikroskopkamera wird von dem Open Source Programm „fswebcam" übernommen. Beide Programme werden innerhalb eines Shell Skripts zum Starten des gesamten Zykliersystems aufgerufen.

2.2.1. Shell-Skript

Das Shell-Skript in Listing 4.1 legt noch nicht angelegte Ordner für die Bilder der Mikroskopkamera auf einer USB-Festplatte an. Es startet das Programm „Steuerung" im Hintergrund. Anschließend wird das Programm „DMM4020_reader_v2" von Herrn Grießbach [15] gestartet, um die Stromwerte der Zelle mit denen der Messplatine zu überprüfen. Als letztes wird dann „fswebcam" für die Mikroskopkamera aufgerufen, um alle zehn Sekunden das Elektrodenfenster der Zelle zu fotografieren.

```
1  #!/bin/bash
2
3  MESSUNG="Zelle_Bemerkung_Datum_"    # HIER DIE MESSUNG BENENNEN
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
4
5  STROM=$MESSUNG'Strom.txt'
6  SPANNUNG=$MESSUNG'Spannung.txt'
7  TEMPERATUR=$MESSUNG'Temperatur.txt'
8  CELL="DMM"
9
10 # Ordner Ã¼fr die Daten der aktuellen Messung
11 ORDNER="/media/65b55418-38c6-4990-b8ba-a1cf2dee2462/PI/MitPlatine/20150907"   # HIER
       VERZEICHNIS AKTUALLISIEREN !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
12
13
14 # Ordner Ã¼fr die Programme
15 STEUERUNG="/home/pi/Programmierung/Steuerung/"
16 DMMORT="/home/pi/Desktop/Messung"
17 # Name des Zyklierplans
18 PLANORT="/home/pi/Programmierung/Steuerung"
19 PLANNAME="steuerfile"
20
```

```
21 #mkdir $ORDNER      # Wenn man die Ordner nicht manuell erstellt Ã¶gehren die root!
22 mkdir $ORDNER/Bilder
23
24 # Start der Zellzyklierung
25 sudo $STEUERUNG/steuerung -i $ORDNER/$STROM -v $ORDNER/$SPANNUNG  -t $ORDNER/$TEMPERATUR -c
       $PLANORT/$PLANNAME -l 500 &
26 PID_PLATINE=$!
27
28 # Start der Spannungsmessung mit dem DMM4020
29 #$DMMORT/DMM4020_reader_v2/DMM4020_reader_v2 -d /dev/ttyUSB0 -f /media/Extern1/PI/20150618/
       $CELL"_spannung_".txt -i 10 -m 1 &
30 #PID_SPANNUNG=$!
31
32 # Start der Strommessung mit dem DMM4020
33 $DMMORT/DMM4020_reader_v2/DMM4020_reader_v2 -d /dev/ttyUSB0 -f $ORDNER/$CELL"_strom_".txt -
       i 10  -m 2 &
34 PID_STROM=$!
35
36 #fswebcam --set "White Balance Temperature, Auto=False"
37 fswebcam -d /dev/video0 --loop 10 -F5  --png 0 --scale 1600x1200 --set "White Balance
       Temperature, Auto=False" --save $ORDNER/Bilder/$CELL'%s'.png &
38 PID_CAMERA=$!
39
40
41 read -n1 -s
42
43 kill -SIGINT $PID_PLATINE
44 #kill -SIGTERM $PID_SPANNUNG
45 kill -SIGTERM $PID_STROM
46 kill -SIGTERM $PID_CAMERA
```

Listing 2.1: Programmstart.sh

2.2.2. Steuersoftware des Zykliersystems

Das von Herrn Geist entwickelte C-Programm „Steuerung" ist auf die Sensorik und Aktorik der Messplatine zugeschnitten. Es kontrolliert und arbeitet folgende Aspekte ab.

- Ist-Spannung der Zelle permanent mit der Schwellenspannung überwachen
- Lade-/Entladestrom permanent konstant regeln
- Einstellen der RGB-LED als Kamerabeleuchtung
- Vorgegebene Ruhezeit zwischen Ladung und Entladung abwarten
- Messdaten in Dateien abspeichern
- Den Lade-/Entladezyklus n-mal wiederholen

Aus Listing 4.1 in Zeile 25 haben die Übergabeparameter des Programms „Steuerung" folgende Bedeutung.

Konsolenaufruf:

→ steuerung -i CurrentFile -v VoltageFile -t TemperatureFile -c ControlFile -l Loops luvcview

Parameter	Bedeutung
-i: current	Pfad der Datei für Strommesswerte
-v: voltage	Pfad der Datei für Spannungsmesswerte
-t: temperature	Pfad der Datei für Temperaturmesswerte
-c: control	Pfad der Steuerdatei des Programms
-l: loops	Anzahl der Zyklierungen (Zyklierschleifen)

Tabelle 2.2.: Übergabeparameter der Steuersoftware

Ein Beispiel der Steuerdatei ist in folgendem Listing 2.2 zu sehen. Dabei ist die Einheit der Spannung in Volt [V], des Stroms in Microampere [μA], der Ruhezeit „sleeping" in Sekunden [s] und die RGB-Werte sind im Bereich von 0 bis 2^8 bzw. 2^{10} anzugeben.

```
1  spannung   2.0
2  strom      10.0
3  sleeping   180
4  led_R    0
5  led_G    0
6  led_B    0
7
8  spannung   4.0
9  strom      10.0
10 sleeping   180
11 led_R    0
12 led_G    0
13 led_B    0
14
15 end
```

Listing 2.2: Steuerfile enthält Zyklierparameter für die Steuerung

2.2.3. Auslesen der Mikroskopkamera

Die Mikroskopkamera wird von dem Kommandozeilenprogramm „fswebcam" ausgelesen. Dieses liefert keine Bilder in Echtzeit und wird daher zum einmaligen-Justieren der Kameralinse von dem Programm „luvcview" abgelöst. Beide Programme müssen nach einem neu aufgesetzten Betriebssystem nachinstalliert werden mit:

Konsolenaufruf:

→ sudo apt-get install fswebcam luvcview

Das Programm „luvcview" kann direkt aus der Konsole aufgerufen werden und braucht keine Übergabeparameter. Das Programm „fswebcam" hingegen benötigt eine ganze Zeile an Parametern. Die in Listing 4.1 in Zeile 37 verwendeten Übergabeparameter haben folgende Bedeutung.

Konsolenaufruf:

→ fswebcam -d /dev/video0 –loop 10 -F5 –png 0 –scale 1600x1200 –set "White Balance Temperature, Auto=False-save fig.png

Parameter	**Bedeutung**
-d: Device	Kamera die durch eine Datei repräsentiert wird
–loop:	Aufnahmeintervall in Sekunden
-F 5	Mittelwertbildung von 5 Bildern
–png 0	Formatangabe der Bilddateien. Die 0 steht für keine Kompression
–scale	Auflösung von 1600x1200 Pixeln
–set	Keinen Weißabgleich vornehmen, sondern Rohwerte der Pixel behalten.
–save	Pfad der zu erstellenden Bilddatei

Tabelle 2.3.: Übergabeparameter der Kamerasoftware

Die Auswertung der Bilddaten geschieht nach Abschluss einer Zyklierung. Dafür gibt es bereits Tools und Matlab Skripte aus vergangen Arbeiten (z.B. Grießbach [15] und Palliyaguruge [21]).

2.3. Inbetriebnahme des bestehenden Systems

Von Herrn Geist wurden am Ende der Entwicklung erfolgreiche Systemtests gefahren und in seiner Thesis dargestellt [14]. Eine Testzyklierung einer Zelle mit 20μA Lade-/Entladestrom zwischen 2V Entladeschluss- und 4V Ladeschlussspannung über 2,5 Tage ist auf folgenden Abbildungen 2.8 zu sehen. In der ersten Abbildung stellen die ersten beiden Kurvenverläufe den Strom dar. Oben gemessen mit einem externen Tektronix DMM4020 und in der Mitte mit dem Onboard ADC auf der Messplatine. Die dritte Messkurve stellt die gemessene Zellspannung dar.

In Abbildung 2.9 ist auf den Werten des Strom-ADC ein Offset von 3μA und ein größeres Rauschen als auf den präziseren Werten des Tektronix Messgerätes zu erkennen.

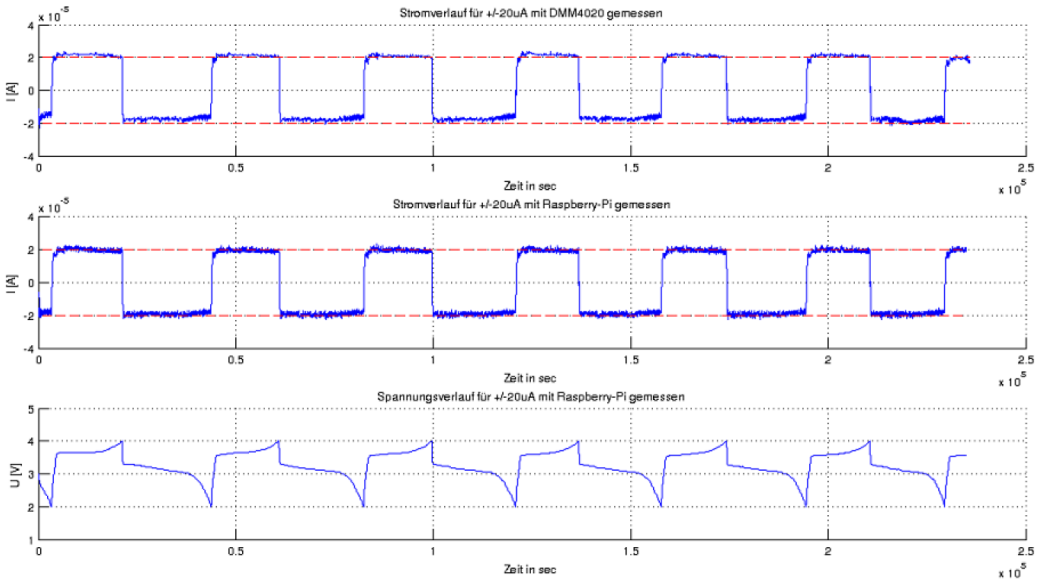

Abbildung 2.8.: Probezyklierung mit 20μA Lade-/Entladestrom über 2,5 Tage nach [14]

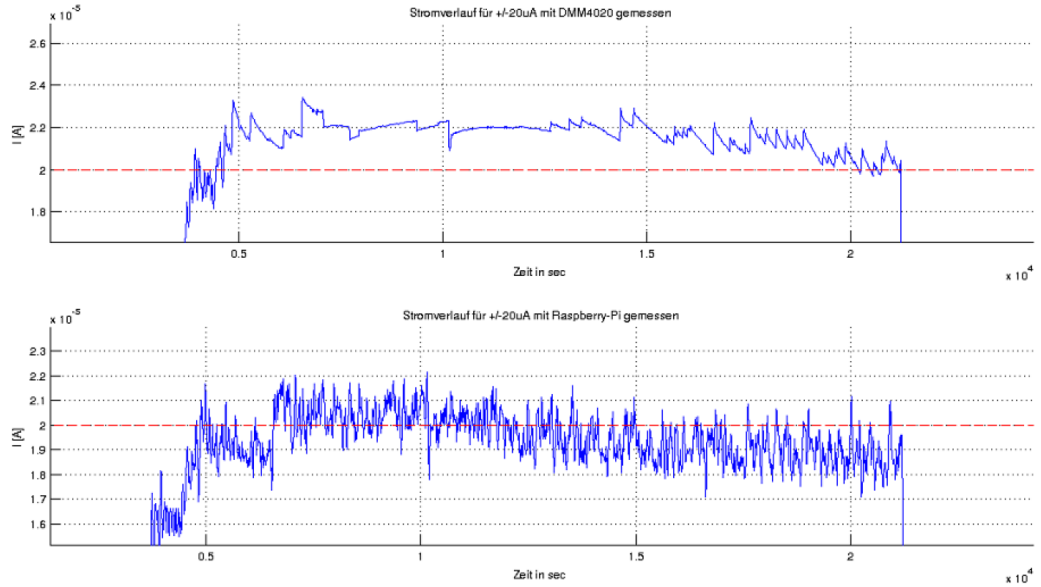

Abbildung 2.9.: Zoom einer Ladephase mit 20μA nach [14]

Die Inbetriebnahme nach der Projektübergabe erwies sich als problematisch. Eine wie in obiger Abbildung 2.8 entsprechende regulierte Batteriezyklierung konnte nicht wiederholt werden. Als Testzelle wurde eine 3,3V LiFePo4 Zelle angeschlossen, die mit den Parametern aus Listing 2.2 zykliert werden sollte. Das System fing mit einer Entladung an und regelte langsam den Strom hoch. Es regelte jedoch nicht auf einen Sollwert von 10μA, sondern stieg immer höher an, bis auf die Entladephase umgeschaltet wurde. Eine Vermutung war, dass ein stärkeres Rauschen als zu den Tests von Herrn Geist an den ADC-Eingängen gemessen wurde. Um dieses Hardware-Problem zu umgehen, wurde das Steuerungsprogramm modifiziert. Es kann so zuverlässige Stromwerte vom DMM4020_reader_v2 lesen, statt von dem Onboard-ADC. Dies funktioniert mit Hilfe einer named Pipe (FIFO), in die der DMM4020_reader_v2 Werte schreibt und das Steuerungs-Programm sie ausliest. Durch diese Maßnahme konnte mit dem Programm und der Messplatine, bis auf die Strommessschaltung, eine Zyklierung mit +-500μA Lade-/Entladestrom über 4 Tage gefahren werden. Die restlichen Parameter sind identisch zu dem Steuerfile aus Listing 2.2. Als Testzellen dienen in diesem Versuch zwei in Reihe geschaltete AA Alkali-Batterien. Auf den folgenden Abbildungen 2.10 und 2.11 sind der gesamte Zyklus von Strom und Spannung und eine Vergrößerung für die ersten 60.000 Sekunden bzw. 16,6 Stunden geplottet.

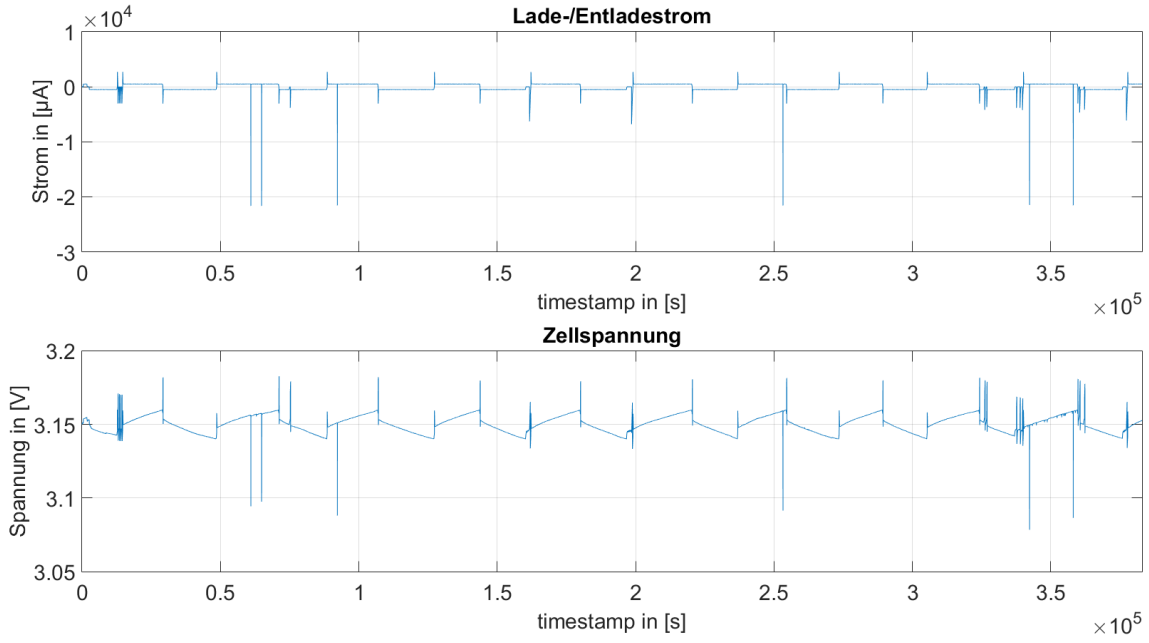

Abbildung 2.10.: Zyklierung mit Steuerungsprogramm und Tektronix DMM4020 Anbindung, über 4 Tage, mit 2xAA Alkali Batterien

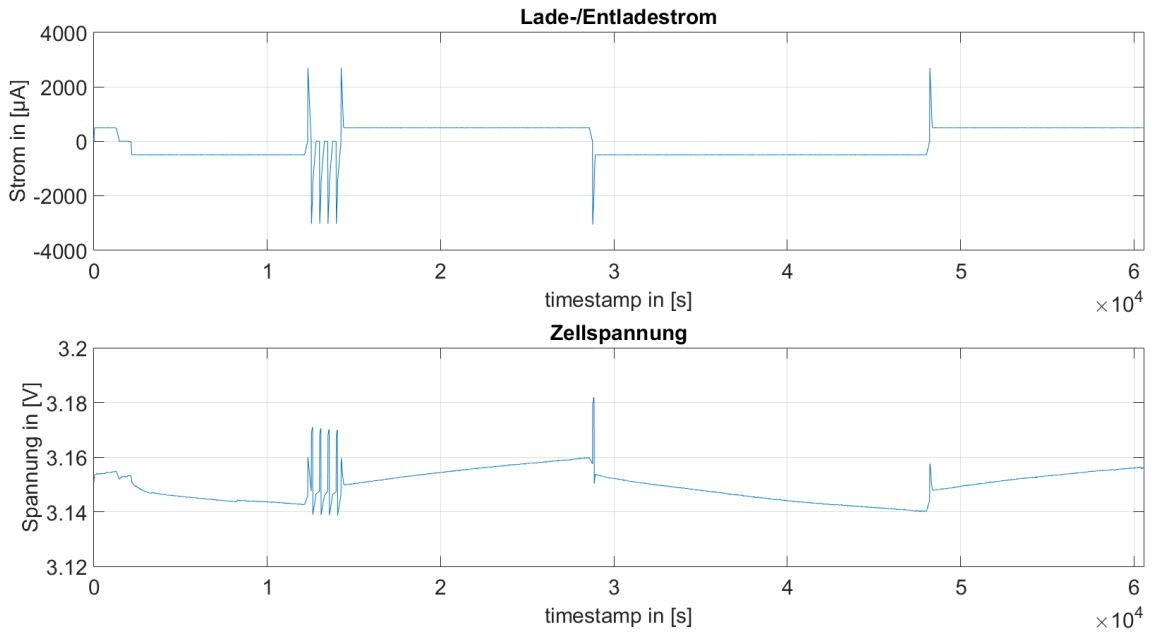

Abbildung 2.11.: Zyklierung mit Steuerungsprogramm und Tektronix DMM4020 Anbindung, über die ersten 16,6 Stunden mit 2xAA Alkali Batterien

In den beiden Abbildungen sind deutliche Artefakte zu erkennen. Mal gibt es kurze starke Ausreißer auf unter 3,1V und mal gibt es ein Hin- und Herschalten des Strompfadrelais. Die nach jeder Lade-/Entladeperiode zu sehenden Peaks von Strom und Spannung sind auf den batteriespezifischen Zellspannungsanstieg beim Erreichen der Ladeschlussspannung zurückzuführen. Während der Erholungszeit ist das Relais im Strompfad geöffnet. Diese Momente sind auf den obigen Verläufen nicht zu sehen, da das Steuerungsprogramm während dieser Zeit keine Messdaten aufnimmt. Es folgt eine Bewertung des Zykliersystems im nächsten Abschnitt.

2.4. Fehleranalyse und Korrekturmaßnahmen

Die Stromregelung war stark am Schwingen, was auf ein Rauschen in den Werten des Strom-ADC zurückzuführen war. Die Steuerungs-Software versucht dies durch eine Mittelwertbildung über 5x100 Werte zu kompensieren, was einem Tiefpassverhalten nachkommt. Ein Vergleich mit Referenzwerten von einem Tektronix-Labormultimeter lässt einen Offset von ca. 9μA erkennen. Leichte Schwankungen waren trotz Mittelwertbildung immer noch enthalten. Ein Verdacht auf eine ungenaue 1,25V Referenzspannung für den Nullpunkt-Offset des Strom-ADC lag nahe. Es mussten also noch Umbauarbeiten an der Hardware

durchgeführt werden, damit eine erfolgreiche Testzyklierung gefahren werden konnte. So war es der 1,25V LDO LD39015, welcher als Referenzspannungsquelle tatsächlich mit $\pm$2% Toleranz zu ungenau war und gegen den LM4121 LDVR mit $\pm$0,2% Toleranz getauscht wurde.

Hinzu kam, dass die Schaltung im hörbaren Frequenzbereich gefiept hat. Die Ursache dieses Verhaltens war schwer zu finden. Mit Hilfe eines Oszilloskops wurden Spannungen an etlichen Punkten in der Schaltung beobachtet. Allgemein waren viele Oberwellen an vielen Punkten der Schaltung zu beobachten. Besonders auffällig periodisch waren Oberwellen auf den Spannungsebenen im Digitalbereich.

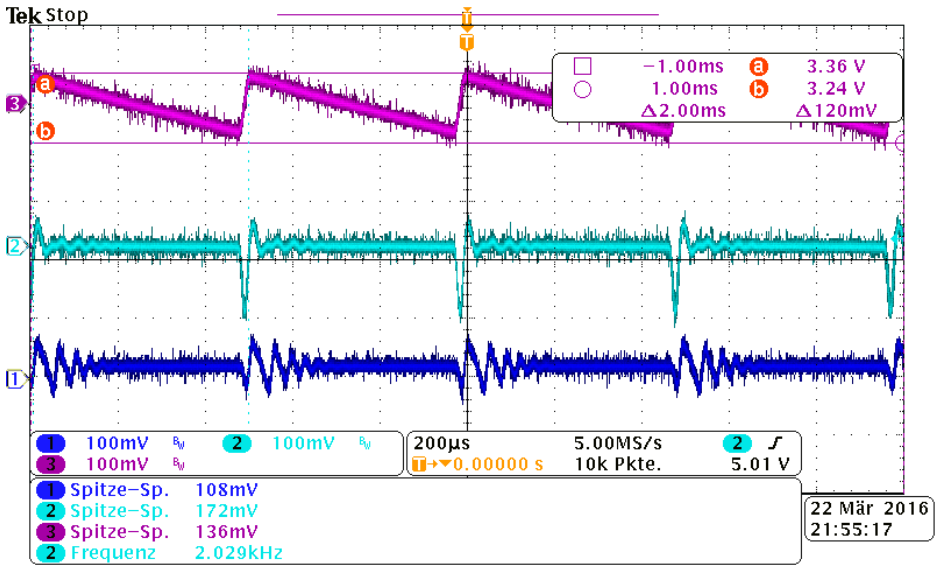

Abbildung 2.12.: Lila: 3,3V Ebene, türkis: Ebene, blau: 10V Ebene

Die Oberwellen wurden von dem 3,3V LDO TLV217 für die SPI Versorgung verursacht. Ein Blick in dessen Datenblatt [7] ließ offenbaren, dass in seiner Aussenbeschaltung ein Kondensator unterdimensioniert war. Dieser im Schaltplan benannte C15 Kondensator wurde von 1μF auf 22μF erhöht. Das hörbare Fiepen war daraufhin weg und das erweist sich mit folgendem Oszillogramm in Abbildung 2.13

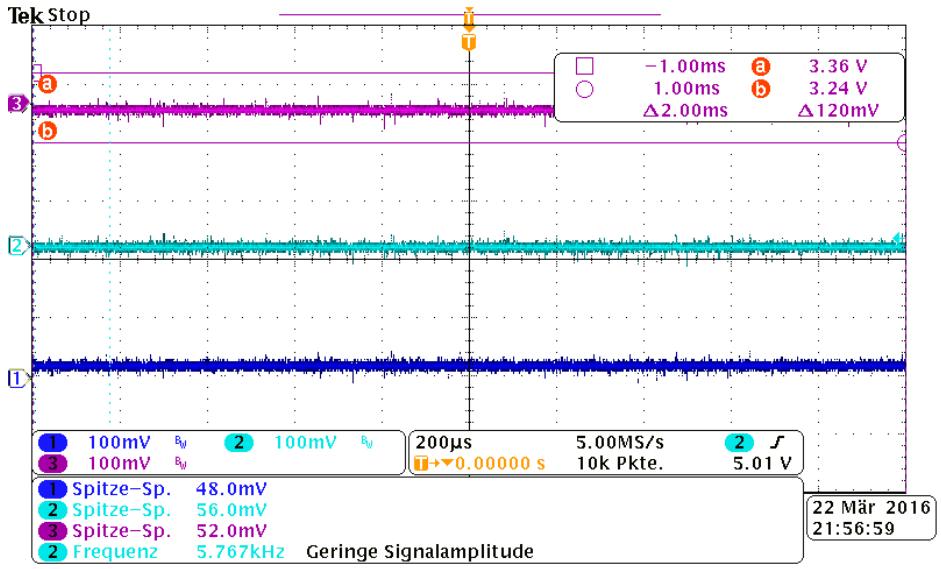

Abbildung 2.13.: Lila: 3,3V Ebene, türkis: Ebene, blau: 10V Ebene

Die Glättungskondensatoren C12=0,47μF und C17=2,2μF am Eingang der 10V und -10V Spannungsregler wurden beide auf 4,7μF höher dimensioniert. Dieses dient als Spannungsglättung bei kurzen Stromanstiegen.

Alle Änderungen am bestehenden Zykliersystem sind im Folgenden als Errata-Sheet aufgelistet.

- C12 von 0,47μF in 4,7μF geändert
- C17 von 2,2μF in 4,7μF geändert
- LD39015 (1,25V $\pm$2% tol) wurde gegen einen LM4121 ausgetauscht und C32 = 32nF hinzugefügt. C16 = 1μF ist so geblieben.
- C15 der 1μF Kondensator am Ausgang des 3,3V LDO wurde gegen einen 22μF Kondesator getauscht.
- TPS799 LDO (2,5V $\pm$1% tol) wurde gegen einen Widerstand und eine Zenerdiode (ADR50k1) ausgetauscht. Die Kondensatoren C6 und C7 fallen dabei weg.
- Strommesswerterfassung mit Tektronix Labormessgerät Softwareschnittstelle über named Pipe.
- An den differentiellen Eingänge des Strom-ADC wurde ein 75nF Kondensator parallel an die Pins JP4 geschaltet, damit erhöhte sich die Messgenauigkeit.

Tabelle 2.4.: Errata Sheet des bestehendes Systems

Später stellte sich heraus, dass die Grenzfrequenz des analogen RC-Filters aus R21 und C23 vor dem Eingang des Strom-ADC zu hoch dimensioniert war. Durch den letzten Schritt des Errata-Sheets 2.4 wurde die Grenzfrequenz verringert. Als Folge verringerte sich auch das Rauschen auf den Werten des Strom-ADC. Der modifizierte Schaltplan ist aus Anhang zu entnehmen.

2.5. Bewertung des bestehenden Systems

Durch den Workaround mittels Pipe und DMM4020_reader_v2 kann die Steuersoftware getestet und beurteilt werden. Eine Anmerkung zur Steuersoftware ist, sie rechnet vollständig in Festkomma Arithmetik. Die aus dem Steuerfile eingelesenen Parameter in Gleitkommazahlen werden in dem Initialisierungsbereich des Programms in Festkommazahlen gewandelt. Der bcm2836 Multimedia Prozessor des RPI2 hat eine Gleitkommaeinheit (FPU) im Kern integriert [2]. Es wäre also möglich die komplette Regelung in Gleitkomma-Arithmetik zu realisieren. Zwar sind Festpunktzahlen minimal performanter, da die Anwendung jedoch nicht zeitkritisch ist, die Ein-und Ausgabewerte in Gleitpunktzahlen umgerechnet werden und es das Debuggen der Regelalgorithmen erleichtern würde, ist es sinnvoll die gesamte Regelung in Gleitkomma-Arithmetik auszuführen. Es werden die Vor-und Nachteile des bisherigen Zykliersystems abgewogen.

Pro:

- Die Zyklierung orientiert sich an dem Ladezustand der Zelle.
- Der Lade-/Entladestrom lässt sich konstant regeln.
- Vor jedem Start wird der Zyklierablauf im Controlfile auf Plausibilität geprüft.

Kontra:

- Während der Ruhezeit nimmt dieses System keine Strom-, Spannungs- und Temperaturwerte auf. Dadurch verpasst man interessantes Batterieverhalten.
- nur konstante Strom Regelung (cC) möglich.
- langsame Einregelung an den Stromsollwert. Bei $LiFePO_4$ Zelle ca. 1 Std um auf $10\mu A$ zu regeln. Bei 2xAA Alkali Batterie ca. 2,5 Min um auf $500\mu A$ zu regeln.
- Es ist nicht intuitiv zu verstehen in welchen Einheiten die Steuersoftware via Steuerdatei bedient werden soll.
- Die Strom- und Spannungsverläufe enthalten noch Artefakte, die auch an der Zelle anliegen.

- Ungenau bei Strömen < $|5\mu A|$.

Da die Nachteile überwiegend softwareseitig zu beheben sind, wurde entschieden ein Software-Redesign des Steuerprogramms zu entwickeln. So konzentrieren sich alle folgenden Kapitel auf die Neuentwicklung einer Mess-und Regelsoftware.

3. Neukonzeption der Software-Architektur

Für ein Redesign eines bereits bestehenden Systems steht im Vordergrund, dass bekannte Fehler und Kinderkrankheiten beseitigt und eventuell Erweiterungen hinzukommen. So auch bei diesem Redesign. Da das bestehende Steuerprogramm einige Mankos hat, die in Kapitel 2.5 aufgeführt sind, konzentrieren sich alle folgenden Kapitel auf die Neuentwicklung einer Mess-und Regelsoftware.
Da die OnBoard-Strommessung der Messplatine nicht ausreichend genau ist, soll eine Möglichkeit geschaffen werden, die Messwerte eines externen Labormessgeräts in die Software einzubinden. Das bietet sich ausserdem auch an, weil von Herrn Grießbach bereits das Programm „DMM4020_Reader_v2" entwickelt wurde, welches über die serielle RS232 Schnittstelle das Labormessgerät ausliest. Das Programm kann seine erfassten Messdaten in eine „named Pipe" schreiben, die der neu entwickelten Software als geöffneter Filedeskriptor zur Verfügung steht.

Es sind zwei Konzeptideen entstanden, die in den beiden Kapiteln 3.1 und 3.2 miteinander verglichen werden: Das konventionelle Konzept der bestehenden Zykliersoftware, welches in seiner Struktur monolithisch aufgebaut ist, übernehmen bzw. neu implementieren oder ein anderes Konzept verfolgen. Letzteres ist ein Paradigmenwechsel, der aber auch eine Reihe an Vorteilen bezüglich der Wartung, Wiederverwendbarkeit und Erweiterbarkeit mit sich bringt. Dies war ein Vorschlag von Herrn Grießbach, da es in der Vergangenheit viele Probleme mit komplexen monolithischen Programmen in Hinsicht der Wartung und Erweiterbarkeit gab.

3.1. Monolithisches Software Konzept

Ein monolithisches System ist im Grunde ein geschlossenes, für einen speziellen Zweck zugeschnittenes System. Ein Vorteil ist der durch den schlanken Aufbau resultierende Performance Gewinn. Bei einem für nur einen Zweck zugeschnittenem System, muss viel weniger berücksichtigt zu werden, als ein System, was viele Anwendungsfälle abdecken soll. Hinzu kommt, dass Test- und Debug Methoden einfacher zu bewältigen sind. Da alles

sequentiell bzw. single threaded in einem Prozess abgearbeitet wird, reicht der Debugger der IDE alleine aus. Single Thread Programme sind übersichtlicher, da Funktionen, die die Hardware ansprechen nicht threadsave sein brauchen. Dies erspart die Benutzung von Synchronisierungsmechanismen wie Semaphoren oder Mutex, um gleichzeitigen Zugriff auf Ressourcen zu vermeiden.

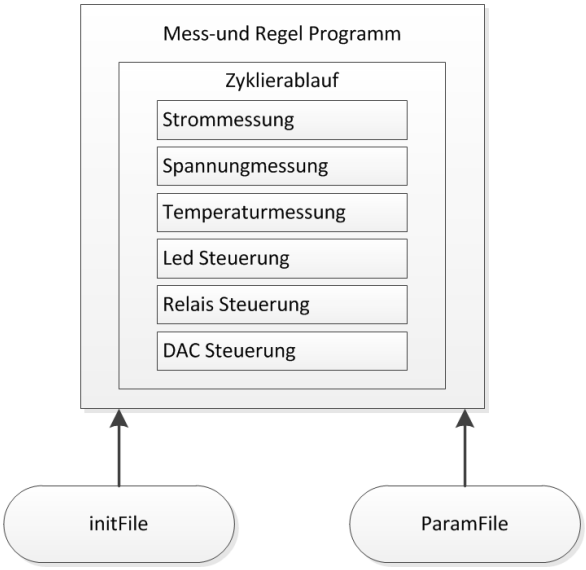

Abbildung 3.1.: Monolithische Architektur

Vorteile:

- Einfache Debugging Möglichkeiten.
- Übersichtlicher, da Programm in einem Task als single threaded ausgeführt wird.

Nachteile:

- Monolithische Software ist schwer erweiterbar, da die Teile dieses Systems nur mit erheblichem Aufwand modifiziert und an neue Bedingungen angepasst werden kann [17].
- Geringe Wiederverwendbarkeit des gesamten Software-Systems.
- Wenn viele Personen daran arbeiten ist die Software ab einer bestimmten Größe nicht mehr beherrschbar. Es kommt bei Modifikation zu nicht vorhersehbaren Nebeneffekten.
- Teile der Software können durch die starre Kopplung zueinander nicht nebenläufig auf verteilten Systemen ausgeführt werden [17].

3.2. Modulares Software Konzept

Entgegen dem ersten Konzept stehen heute moderne Anwendungs-Architekturen, sogenannte Client-Server Anwendungen. GUI-Applikationen, Datenaufnahme, Ergebnisweitergabe werden als eigenstehende Komponenten von der Kernfunktionalität getrennt aufgebaut und wirken über eine definierte Schnittstelle mit dieser zusammen. Insgesamt hat sich also ein Paradigmenwechsel weg von monolithischen Architekturen hin zu serviceorientierten Architekturen (sogenannten SOA) [17] vollzogen. Diese abstrakten Architekturmuster schaffen eine Grundlage für verteilte Systeme.

In diesem Konzept wird ein sogenannter Rich-Client mit drei Messservern und drei Stellservern kommunizieren. Dabei wird von den Servern die Hardware Ansprache abstrahiert, sodass der Client Hardware unabhängig kompiliert werden kann. Damit ist er auf vielen Linux Plattformen lauffähig. Die Mess- und Stellserver sind jedoch von der Hardware des Raspberry Pi 2 abhängig, da sie deren Framework-Funktionen direkt aufrufen. Zugegriffen werden muss z.B. auf den SPI-Bus, einige GPIO-Ports oder evtl. einem PWM Pin, um externe Hardwarekomponenten auf der Messplatine von Herrn Geist zu steuern.
Die Kommunikation zwischen Mess-und Stellservern und dem Client findet über ein anwendungsspezifisches Protokoll statt. Dieses sorgt für einen reibungslosen, geregelten und standardisierten Informationsfluss zwischen den Prozessen.

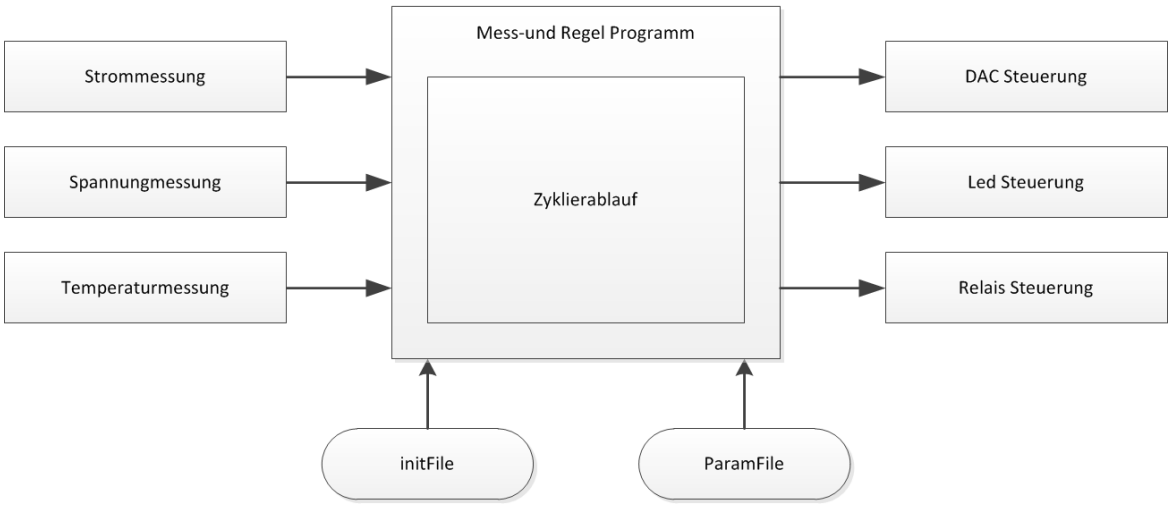

Abbildung 3.2.: Modulare Architektur

Um eine Interprozesskommunikation (IPC) aufzubauen, gibt es vier Möglichkeiten in Linux. Dazu gehören Shared Memory Methoden, Message Queues, Pipes/ FIFOS und Unix Domain Sockets. Auf sie wird im nachfolgendem Kapitel näher eingegangen. Die Vor- und Nachteile der modularen Software-Architektur sind noch einmal zusammengefasst [17]:

Vorteile:

- Hohe Wiederverwendbarkeit einzelner Module. Dadurch flexiblerer Einsatz möglich.
- Wartungsfreundlicher, da Probleme auf Teilmodule differenziert werden können.
- Software-Erweiterungen sind leichter in das modulare System zu implementieren, da nicht der gesamte Quellcode angesehen werden muss. Es müssen jedoch die Rahmenbedingungen der Schnittstellen eingehalten werden.
- Potentiell mehr Absturzsicherheit, da ein Überwachungsprozess das nicht ordnungsgemäß beendete Modul einfach wieder neu instantiieren kann.
- Durch die verbindungsorientierte Schnittstellenkommunikation mittels Sockets können Module der Software nicht nur nebenläufig auf einer Plattform ausgeführt werden, sondern auch zwecks Lastenverteilung auf verteilten Plattformen.

Nachteile:

- Minimal schlechtere Performance, da die Software und ihre Funktionen nicht nur für einen Zweck zugeschnitten sind. Höhere Modularität geht mit größerem Framework Overhead einher.
- Um Testen und Debuggen zu können sind eigens entwickelte Debug-Tools notwendig, da mehrere Prozesse gleichzeitig voneinander abhängen. Ein IDE-Debugger kann innerhalb der IDE nur einen Thread bzw. Prozess zur Zeit debuggen.

3.3. Entscheidung für eine Architektur

Im Sinne der Anwendung soll eine zuverlässige und stabile Software entstehen, die jedoch zeitnah fertig gestellt werden soll. Der Mehrvorteil des zweiten Konzepts, der gleichzeitig aber auch mit einem Mehraufwand und längerer Entwicklungszeit einhergeht, lässt eine Entscheidung des Entwicklungsweges gut überlegt sein. Durch die überwiegenden Vorteile und das Interesse, tiefer in die Linux Systemprogrammierung einzusteigen, fällt die Entscheidung auf das modulare Konzept.
Zu klären ist noch welche Möglichkeit zur Interprozesskommunikation gewählt wird, um zuverlässige, schnelle und einfache Prozessschnittstellen zu implementieren. Die im Kapitel 3.2 erwähnten Möglichkeiten sind wie folgt zusammengefasst.

Shared Memory ist ein gemeinsam benutzbarer Speicherbereich, der innerhalb der Prozesse als Datei repräsentiert wird. Dieser muss mit systemglobalen Sperrsynchronisationsvariablen wie Semaphore oder Mutex vor gleichzeitigem Zugriff mehrerer Prozesse synchronisiert werden. Das ist jedoch für mehrere Teilnehmer umständlich. Diese Aufgaben und weitere werden von den anderen Kommunikationsbausteinen nicht sichtbar durch ein Übergabeprotokoll geregelt. [24]

Pipes und FIFOs bieten eine etwas schnellere Kommunikation. Diese sind eine Linux-Erweiterung des ANSI-C Standards und stellen einen unidirektional gepufferten Übertragungskanal zwischen Prozessen dar. Sie arbeiten auch mit gemeinsamem Speicherbereich, dieser wird jedoch optimal vom Betriebssystem verwaltet, und da es unidirektional ist, bedarf es keiner Sperrsynchronisation. „FIFOs" oder auch „named Pipes" bilden den gemeinsamen Speicherbereich als Datei auf dem Dateisystem ab. Alle Prozesse mit entsprechenden Rechten können auf diese Datei zugreifen. Sie bleibt nach Beendung aller Teilnehmer Prozesse erhalten, bis sie aktiv gelöscht wird [24].

Message Queues sind Nachrichtenwarteschlangen, die auch als Dateien abgebildet werden. Diese werden aus Geschwindigkeitsgründen jedoch nicht auf dem Massenspeicher, sondern über einem Betriebssystem nativen Treiber, in einem virtuellen Bereich des Arbeitsspeichers angelegt. Sie sind Nachrichten orientiert. Die Teilnehmer können die Nachrichten mit Prioritäten versehen, um sie nach einer geordneten Reihenfolge aus der Queue zu lesen. [24]

Unix Domain Sockets sind moderne verbindungsorientierte Kommunikationsbausteine. Es muss immer zuerst eine Verbindung der miteinander kommunizierenden Prozesse aufgebaut werden. Sind zwei Prozesse verbunden, kann kein anderer Prozess die Nachrichten über diese private Verbindung mitlesen oder manipulieren. Um viele Verbindungen gleichzeitig verwalten zu können, bedarf es serverseitig für jeden Kanal eigene Filedeskriptoren. Das Socket wird als .uds-Datei auf dem Dateisystem repräsentiert.
Im Gegensatz dazu arbeiten Pipes Message Queues oder Shared Memory verbindungslos. Jeder Prozess mit entsprechenden Rechtevergaben kann eine Nachricht lesen, ohne dabei eine Verbindung zum Senderprozess aufgebaut zu haben. Damit könnte der eigentliche Empfängerprozess verhungern [16, S.838].

Die Entscheidung fällt auf die Unix Domain Sockets, da sie einen größeren Beitrag zu mehr Modularität und die Möglichkeit, räumlich getrennte Systeme aufzubauen, verschafft. Mit entsprechenden Einstellungen in den Initialisierungs-Funktionen können Sockets auch netzwerkfähig mit dem verbindungsorientierten TCP/IP Protokoll implementiert werden. So könnte z.B. der Client Prozess über LAN oder Internet von einer anderen Linux-Plattform agieren [20].

4. Entwicklung der BatCycle Software

Nachdem die Entscheidung auf das modulare Konzept gefallen ist, wird angefangen dieses zu verfeinern und detaillierter zu strukturieren. Alle in Bild 4.1 beteiligten Programme sollen zusammen den Namen BatCycle-Software tragen, was für Batterie Zyklierungs Software steht. Jegliche Namensgebung, der gesamte Quellcode sowie die In- und Output Files sind in englischer Nomenklatur. Bei der Startparameterübergabe per Kommandozeile sollen die Programme sich strikt an den Linux Stil halten.

Es ist noch ein Logger Programm zur Messwert-Speicherung dazu gekommen, der „BC_DataLogger_CP". Zuerst bestand die Idee, die Messwertspeicherung vom MainController übernehmen zu lassen, doch es entspricht dem modularen Konzept mehr, diese Teilaufgabe wieder in einem extra Prozess zu isolieren, um Komplexität aus dem MainController zu nehmen.

Die beiden .cfg-Dateien „bc_init.cfg" und „bc_param.cfg" sind Konfigurierdateien für den MainController. Jegliche Einstellungen und Zyklierabläufe ausschließlich als Kommandozeilenparameter dem Programm zu übergeben, wäre zu umständlich. Daher werden alle Parameter, die den Zyklierablauf angehen, separat durch eine „bc_param.cfg" Datei dem Programm übergeben. Parameter, die die innere Funktionalität, die Socket-Pfade oder die Messplatine betreffen, werden selten geändert und ebenfalls separat über eine Datei „bc_init.cfg" dem Programm übergeben. Diese werden von den Funktionen in „bc_config.h" über eine Parser Library von Marc A. Lindner [19] eingelesen.

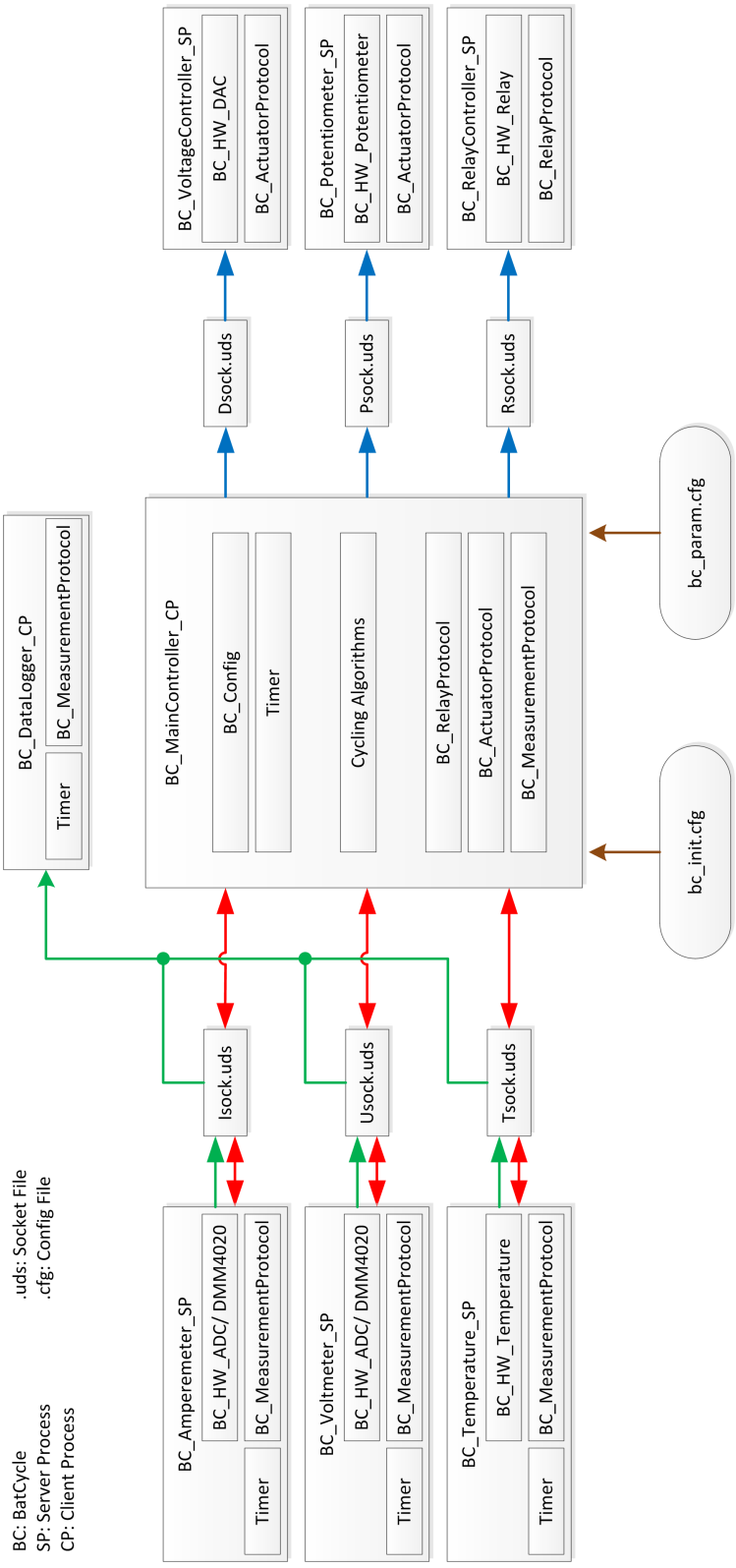

Abbildung 4.1.: BatCycle Prozess Diagramm

4.1. Übersicht der einzelnen Programme

Die BatCycle Software lässt sich gruppieren in Messprogramme, Regelprogramm, Stellprogramme und Logger.

Anmerkung: Alle Programme die auf die Hardware des Raspberry Pi 2 zugreifen, brauchen Vollberechtigung. Dies geschieht mit „sudo" vor jedem Programmaufruf in der Konsole.

4.1.1. BC_Amperemeter_SP

Dieses Programm liest den Strom-ADC auf der Messplatine aus. Damit wird der Lade-/Entladestrom gemessen. Gestartet wird es per Kommandozeile wie folgt:

```
sudo ./BC_Amperemeter_SP -h

USAGE:        ./BC_Amperemeter_SP [-p sock] [-n clients] [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. Default is '/tmp/lsock.uds'.
              -n    Max clients; default limit is 10 clients.
              -h    Help; view help text.

DESCRIPTION: Server process for current measurement.
```

Listing 4.1.: Hilfetext von BC_Amperemeter_SP

Das Programm agiert als Socketserver und wird über das „BC_MeasurementProtocol" von einem zum Master Clienten angemeldeten Teilnehmer angesprochen. Alle weiteren Clienten sind Slaves und können nur zuhören. Der Master Client hat Berechtigung einen Wert anzufordern oder den Messserver in einen „Continuous mode" zu versetzen. Dadurch werden allen verbundenen Clienten, durch ein vom Master Clienten mitgeteiltes Intervall, periodisch Werte gesendet. Die Betriebsparameter des ADC können jederzeit in der Headerdatei „BC_HW_DAC.h" angepasst werden.

4.1.2. BC_Voltmeter_SP

Das Programm liest den Onboard ADC auf der Messplatine zur Spannungsmessung der Batteriezelle aus und sendet sie an seine verbundenen Clienten. Der Aufruf in der Kommandozeile ist:

```
sudo ./BC_Voltmeter_SP -h

USAGE:        ./BC_Voltmeter_SP [-p sock] [-n clients] [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Vsock.uds'.
              -n    Max clients; default limit is 10 clients.
              -h    Help; view help text.

DESCRIPTION:  Server process for voltage measurement.
```

Listing 4.2.: Hilfetext von BC_Voltmeter_SP

Das Programm agiert als Socketserver und wird über das „BC_MeasurementProtocol" von einem zum Master Clienten angemeldeten Teilnehmer angesprochen. Alle weiteren Clienten sind Slaves und können nur zuhören. Die Modi sind gleich wie bei „BC_Amperemeter_SP".

4.1.3. BC_Temperature_SP

Das Programm liest den nah an dem Messobjekt montierten Temperatursensor aus und sendet den Wert an seine verbundenen Clienten. Der Aufruf in der Kommandozeile ist:

```
sudo ./BC_Temperature_SP -h

USAGE:          ./BC_Temperature_SP [-p sock] [-n clients] [-h help]

OPTIONS:        -p      Socketfile; will be overwritten if it exists. Default is '/tmp/Tsock.uds'.
                -n      Max clients; default limit is 10 clients.
                -h      Help; view help text.

DESCRIPTION:    Server process for temperature measurement.
```

Listing 4.3.: Hilfetext von BC_Temperature_SP

Das Programm agiert als Socketserver und wird über das „BC_MeasurementProtocol" von einem zum Master Clienten angemeldeten Teilnehmer angesprochen. Alle weiteren Clienten sind Slaves und können nur zuhören. Die Modi sind gleich wie bei „BC_Amperemeter_SP".

4.1.4. BC_DMM4020Reader_SP

Dieses Programm liest über serielle RS232 Schnittstelle die externen Labormessgeräte Tektronix DMM4020 und Fluke aus. Es ist unabhängig von der Messplatine und kann so auf vielen Linux Distributionen ausgeführt werden. Die Kommandozeilenparameter sind:

```
sudo ./BC_DMM4020Reader_SP -h

USAGE:      ./BC_DMM4020Reader_SP -d dev [-p sock] [-n clients] [-m mode] [-h help]

OPTIONS:    -d    Serial port device
            -p    Socketfile; will be overwritten if it exists. Default is
                  '/tmp/DMM4020sock.uds'.
            -n    Max clients; the default limit is 10 clients.

            mode:   0: volts DC
                    1: volts AC
                    2: amperes DC
                    3: amperes AC
                    4: volts true RMS
                    5: amperes true RMS
                    6: ohms

DESCRIPTION: Server process which is a Tektronix DMM4020 Reader.
```

Listing 4.4.: Hilfetext von BC_DMM4020Reader_SP

Das Programm agiert als Socketserver und wird über das „BC_MeasurementProtocol" von einem zum Master Clienten angemeldeten Teilnehmer angesprochen. Alle weiteren Clienten sind Slaves und können nur zuhören. Die Modi des Protokolls sind gleich wie bei „BC_Amperemeter_SP".

Anmerkung: bis zu einer Abtastperiode von Ta = 0,5s funktioniert die Messwertauslesung zuverlässig. Wird dieses Intervall unterschritten, kann es passieren dass Werte nicht mehr richtig übertragen werden. In dem Fall wird „NAN" als Wert übermittelt.

4.1.5. BC_VoltageController_SP

Das Programm steuert die einstellbare Spannungsquelle der Messplatine an. Der Verstärker nach dem DAC wird von dem Programm mit einbezogen.

```
sudo ./BC_VoltageController_SP -h

USAGE:        ./BC_VoltageController_SP [-p sock] [-n clients] [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Dsock.uds'.
              -n    Max clients; default limit is 1 clients.
              -h    Help; view help text.

DESCRIPTION: Server process to control the DAC voltage.
```

Listing 4.5.: Hilfetext von BC_VoltageController_SP

Dieser Socketserver lässt sich über das „BC_ActuatorProtocol" von einem verbundenen Clienten ansprechen. Es ist auf einen Ausgangsspannungsbereich von 0V bis 6V in $U_{lsb} = \frac{U_{ref}}{2^{16}}$ Schritten ausgelegt. Es kann jedoch auch auf beliebige DAC Parameter wie Referenzspannung und Auflösung in der Headerdatei „BC_HW_DAC.h" angepasst werden.

4.1.6. BC_Potentiometer_SP

Dieses Programm steuert drei Potentiometer auf der Messplatine an. Diese regulieren die Kamera RGB-LED-Beleuchtung zur optischen Spektraluntersuchung der Fotozelle.

```
sudo ./BC_Potentiometer_SP -h

USAGE:        ./BC_Potentiometer_SP [-p sock] [-n clients] [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Psock.uds'.
              -n    Max clients; default limit is 1 clients.
              -h    Help; view help text.

DESCRIPTION: Server process to control digital potentiometer.
```

Listing 4.6.: Hilfetext von BC_Potentiometer_SP

Anmerkung: Dieses Programm wurde noch nicht im Zusammenspiel mit den digitalen Potentiometern der Messplatine getestet.

4.1.7. BC_RelayController_SP

Das Programm dient zur Ansprache der drei Relais auf der Messplatine. Es handelt sich dabei um die Relais zur zweipoligen Trennng (RELAY_VP und RELAY_VN) und um das Lastkreisrelais (RELAY_I). Die Kommandozeilenparameter sind wie folgt erklärt:

```
sudo ./BC_RelayController_SP -h

USAGE:        ./BC_RelayController_SP [-p sock] [-n clients] [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Rsock.uds'.
              -n    Max clients; default limit is 1 clients.
              -h    Help; view help text.

DESCRIPTION: Server process to access the relays.
```

Listing 4.7.: Hilfetext von BC_RelayController_SP

Ansprache des Socketserver Programms findet über das „BC_RelayProtocol" statt. Mit diesem lassen sich die Relais einzeln pro Frame öffnen und schließen, sowie umkippen (toggeln). Es können aber auch alle Relais auf einmal pro Frame in unterschiedliche Zustände gebracht werden.

4.1.8. BC_MainController_CP

Dies ist das Hauptprogramm, welches über sämtliche Sockets Werte von den Messservern abfragt, sie verrechnet und sie dann zu den Stellservern schreibt. Es liest zwei Config Dateien ein. Das erste dient der Initialisierung der Socketpfade und das zweite beinhaltet den genauen Zyklierablaufplan. Letzteres ist ein Parameterfile und sollte an die zu zyklierende Zelle angepasst sein. Das Programm überprüft beide Dateien auf Fehler, bevor eine Batteriezyklierung gefahren wird. Die Kommandozeilenparameter sind:

```
./BC_MainController_CP -h

USAGE:        ./BC_MainController_CP [-i initFile] [-p paramFile] [-h help]

OPTIONS:      -i    initFile; will be read. Default is 'in/bc_init.cfg'.
              -p    paramFile; will be read. Default is 'in/bc_param.cfg'.
              -h    help; view help text.

DESCRIPTION:  Client process to control the measurement and actuator processes from
              modular batcycle process family.
```

Listing 4.8.: Hilfetext von BC_MainController_CP

Es können drei unterschiedliche Lade-/Entlademodi gefahren werden. Diese werden in dem Parameterfile mit angegeben. Zur Auswahl steht der cV Modus (constant Voltage charging), cC Modus (constant Current charging) oder der cCcV Modus (constant Current constant Voltage charging).

4.1.9. BC_DataLogger_CP

Dem Programm wird per Kommandozeile ein existierender Socket eines Messservers und ein mit Schreibrechten versehenes Logfile übergeben. Der DataLogger verbindet sich als Slave Client auf dem Messserver und empfängt von diesem alle Zeitstempel und Messwerte. Diese dienen der Messdatenaufnahme zum späteren Plotten der Kurvenverläufe. Existiert das Logfile noch nicht, wird es neu angelegt, ansonsten werden Daten angehängt.
Weitere optionale Parameterzeilen können in der Anordnung beliebig oft angehängt werden. Der DataLogger kann so von mehreren Messservern Daten empfangen.

```
./BC_DataLogger_CP -h

USAGE:      ./BC_DataLogger_CP sockfile:logfile[:mode:interval]
                               [sockfile:logfile[:mode:interval]]
                               [...]

OPTIONS:    sockfile:  existing path to any BCMP-socket.
            logfile:   name of logfile that will be created. If file exists, it will be
                       appended.

            mode:      0: default mode; DataLogger logs every value and timestamp
                          on socket.
                       1: standalone mode; DataLogger becomes master client on
                          BCMP-server. Server will send values in continuous mode.
                       2: äquidistant mode; DataLogger reads values from socket in
                          quasi äquidistant time, which is delivered in interval.

            interval:  time in msec in combination with an mode argument greater
                       than 0.

DESCRIPTION: BatCycle Measurement Protocol Logger.
```

Listing 4.9.: Hilfetext von BC_DataLogger_CP

Die in den eckigen Klammern stehenden Parameter „Mode" und „Interval" sind optional. Letzterer ist nur im Zusammenhang mit einem Parameter für Mode > 0 möglich.

Als Modi gibt es drei zur Auswahl. Wird der optionale mode Parameter weggelassen, ist das Programm standardmäßig in „Mode 0" eingestellt. Dieser empfängt und loggt alle vom Messserver gesendeten Datenframes mit.
„Mode 1" ist ein „Standalone mode", in dem der DataLogger selbstständig eine Master Client Anfrage zum korrespondierenden Messserver sendet, um ihn dann in seinen Continuous Mode zu versetzten.
Der Äquidistanz Modus „Mode 2" empfängt alle Datenframes der Messserver, loggt aber nur alle dem Intervall entsprechend ankommenden Werte mit. Das Intervall wird von einer Toleranz von $\pm 5 ms$ umgeben. Es wird daher nur eine quasi Äquidistanz erreicht. Die Werte können nur so äquidistant sein, wie der Server sie sendet.

4.1.10. MP_FrameDebugger

Dies ist ein Testprogramm, das zum Debuggen der Measurement Server verwendet wird. Es hat das „BC_MeasurementProtocol" implementiert und kann die Messserver in Continuous Modus versetzen oder einzelne Werte abfragen. Es kann sich nur mit einem Messserver zur Zeit verbinden. Jedoch können mehrere dieser Programme gleichzeitig ausgeführt werden. Aufgerufen wird es mit:

```
./MP_FrameDebugger -h

USAGE:        ./MP_FrameDebugger -p sock [-h help]

OPTIONS:      -p    Socketfile; will be overwritten if it exists. No default
              -h    Help; view help text

DESCRIPTION:  Debug process for the measurement protocol
```

Listing 4.10.: Hilfetext von MP_FrameDebugger

Wenn dass Programm sich mit einem Messserver-Socket verbunden hat, werden vier Optionen im Standardoutput angezeigt: [n]ew Frame, [s]end Frame, [v]iew Protocol, [q]uit. Nach Drücken der Taste in eckigen Klammern, lässt sich die zugehörige Option wählen. Groß und Kleinbuchstaben haben gleiche Wirkung. Zuerst muss ein neuer Frame angelegt werden, danach kann er beliebig oft mit [s] zum Messserver gesendet werden. Mit [v] wird dem Anwender das zugehörige Protokoll noch einmal angezeigt und mit [q] wird ein Quit-Frame gesendet und der FrameDebugger beendet sich.

4.1.11. RP_FrameDebugger

Auch dieses ist ein Testprogramm und wird zum Debuggen des „BC_RelaisController_SP" verwendet. Daher ist das „BC_RelaisProtocol" implementiert. Die Konsolenanleitung des Programms ist:

```
./RP_FrameDebugger -h

USAGE:          ./RP_FrameDebugger [-p sock] [-h help]

OPTIONS:        -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Rsock.uds'
                -h    Help; view help text

DESCRIPTION:    Debug process for the measurement protocol
```

Listing 4.11.: Hilfetext von RP_FrameDebugger

Auch hier ist wieder ein gleiches Menü wie im „MP_FrameDebugger” eingebaut.

4.1.12. AP_FrameDebugger

Dieses Testprogramm ist der FrameDebugger des „BC_ActuatorProtocol”. Damit werden die beiden Stellserver „BC_VoltageController_SP” und „BC_Potentiometer_SP” angesprochen. Die Kurzanleitung für einen Konsolenaufruf ist:

```
./AP_FrameDebugger -h

USAGE:          ./AP_FrameDebugger [-p sock] [-h help]

OPTIONS:        -p    Socketfile; will be overwritten if it exists. Default is '/tmp/Dsock.uds'
                -h    Help; view help text

DESCRIPTION:    Debug process for the actuator protocol
```

Listing 4.12.: Hilfetext von AP_FrameDebugger

Das Optionsmenü ist gleich wie das der beiden vorherigen FrameDebugger. Es wird vom

Anwender selbst über das „BC_ActuatorProtocol” entschieden, ob es ein Frame für den VoltageController oder für den Potentiometer Prozess senden soll.

4.1.13. LinTest

Das letzte Testprogramm ist zum Aufzeichnen der ADC- und DAC-Kennlinien. Da jeder ADC und DAC gewöhnlich Gain-, Offset- und Linearitätsfehler (INL und DNL) hat, kann mit diesem Programm eine Kennlinienkorrektur vorgenommen werden. Da das Programm auf die Hardware der Messplatine direkt zugreift, muss es mit sudo aufgerufen werden.

```
./LinTest -h

USAGE:      ./LinTest [-1 sockFileVDAC] [-2 sockFile] [-h help]

OPTIONS:    -1   Socketfile of external device to measure DAC voltage; will be
                 overwritten if it exists. Default is '/tmp/DMM4020VDACsock.uds'.
            -2   Socketfile of external device to compare ADC_I or ADC_V;
            -h   Help; view help text.

DESCRIPTION: Test program to drive up linearly the DAC, ADC_I or ADC_V range and
             measure their diviations like gain error, offset error and compare
             their results against an external professional high resoloution device.
```

Listing 4.13.: Hilfetext von LinTest

Nach Programmstart hat der Anwender drei Optionen [D]ac, ADC[I] und ADC[V], die mit Tastendruck des Buchstaben in eckigen Klammern gewählt werden. Die Option [D]ac stuft in einem abgefragten Spannungsintervall schrittweise den DAC von einem Startwert zu einem Endwert hoch. Die Ist-Werte werden dabei von einem extern angeschlossenen DMM4020 abgefragt und mit geloggt. Bei den Optionen ADC[I] und ADC[V] wird die DAC Kennlinie ebenso linear hochgefahren, dabei werden zusätzlich die Werte der entsprechenden ADC für jeden Schritt abgefragt und geloggt. Ein zweiter DMM4020 ist hier für den entsprechenden ADC notwendig. Die entsprechenden Relais müssen vorher mit dem „BC_RelaisController_SP” geschlossen oder hardwaremäßig auf der Messplatine gebrückt werden.

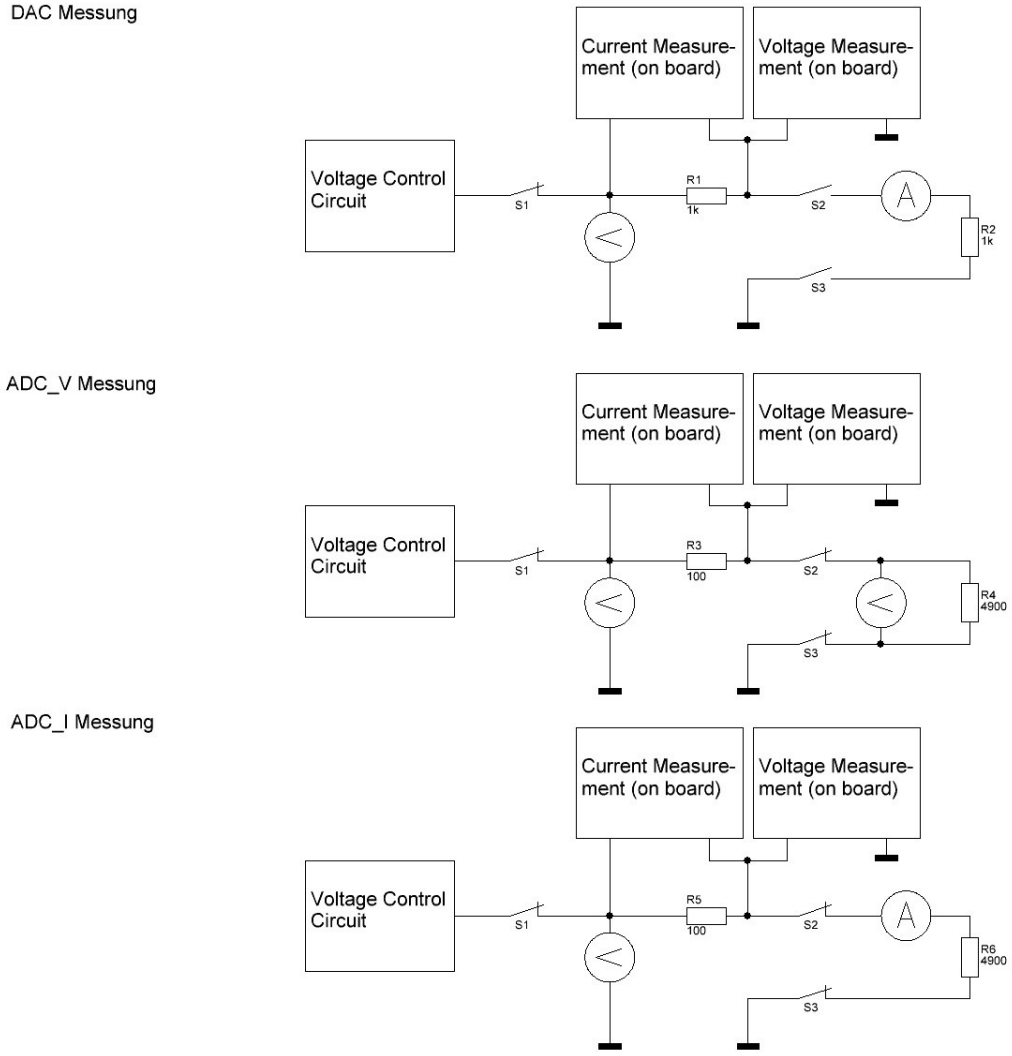

Abbildung 4.2.: Anschlussplan der drei Kennlinienmessungen für LinTest

4.2. Kommunikationsprotokolle BCMP, BCAP und BCRP

Alle BatCycle Protokolle sind im Ordnerpfad „/src/BC_Protocol" angelegt. Sie inkludieren alle ein gemeinsames Headerfile „BC_Protocol.h". In diesem sind Grundfunktionen, globale Makros und Typdefinitionen definiert. Versions Updates sind im jeweiligen Source- und Headerfile Kopf kommentiert. Client- und serverseitig beträgt die Sende-und Empfangs-Buffergröße 23 Byte. Die Länge eines Frames darf nicht größer sein als diese Zahl. Da die Frames unterschiedlich lang sein können und es ineffizient wäre, jeden Frame bis zur maximalen Größe aufzustopfen, werden alle Frames mit einem Terminatorsymbol abgeschlossen. Dieses ist das „new Line" Symbol '\n' (ASCII 0x0A).

4.2.1. BC_MeasurementProtocol (BCMP)

Dieses Protokoll dient zur Messdatenabfrage der Messserver in bidirektionaler Halbduplex Kommunikation mit einem Client Prozess. Client und Messserver haben beide die Headerfiles „BC_MeasurementProtocol.h" inkludiert. Alle Messserver sind in „BMP_ServerMain.c" eingebettet und auf dieses Protokoll ausgelegt. Über das Protokoll können sie in einen Continuous und einen Listen Modus versetzt werden.

Request Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
GetValue	0x01	0	-	value request frame (success only if client is authorized as master).
StartContinuous	0x02	4	interval in ms [uint32]	starts the interval mode (success only if client is authorized as master).
StopContinuous	0x03	0	-	stops the interval mode (success only if client is authorized as master).
GetControl	0x04	0	-	control request frame for authorizing as master client.
RetControl	0x05	0	-	control frame for releasing the master authorization.
reserve	0x06	0	-	free space
⋮	⋮	⋮	⋮	⋮
Quit	0x7F	0	-	quit frame to terminate the server (success only if client is authorized as master).

Response Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
Value	0x81	16	timestamp [uint64], value [double]	value response frame to all connected clients.
Control	0x84	1	0: denial, 1: grant, 2: release	response frame with authorization frame.
reserve	0x85	0	-	free space
⋮	⋮	⋮	⋮	⋮

Tabelle 4.1.: Measurement Protocol v.0.2

Alle möglichen Befehle des Protokolls sind als „MPCommand_t" Enum Typ in „BC_MeasurementProtocol.h" definiert. Der Addressbereich der Request Frames erstreckt sich von 0x00 bis 0x7F. Dabei sind 0x01 bis 0x05 belegte Commands, der Rest bis 0x7E unbelegte und 0x7F der Quit Command. Die Response Frames belegen einen Bereich von 0x80 bis 0xFF. Durch diese Aufteilung des Addressbereichs kann am hochwertigsten Bit (MSB) erkannt werden, ob es sich um einen Request oder Response Frame handelt. Jeder Command hat unterschiedliche Argumente und Framelängen. Diese sind in Listing 4.1 aufgelistet.

4.2.2. BC_ActuatorProtocol (BCAP)

Das Protokoll ermöglicht ein unidirektionales Senden von Sollwert Frames zu den Stellservern. Bei den Stellservern handelt es sich konkret um den „BC_VoltageController_SP", der den DAC mit dem empfangenen Spannungswert einstellt und um den „BC_Potentiometer_SP" Prozess, der mit einem Value Frame alle drei Potis auf einen Widerstandswert einstellt. Clienten und Stellserver haben alle das „BC_ActuatorProtocol.h" Headerfile inkludiert. Die Stellserver nutzen alle das „BCAP_ServerMain.c" Sourcefile, welches das Fundament eines Stellservers ist.

Request Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
SetDAC	0x01	8	value [uint8]	value frame for DAC input.
SetLED	0x02	12	values 3x[uint8]	value frame to RGB Lightning controller.
reserve	0x03	0	-	free space
⋮	⋮	⋮	⋮	⋮
Quit	0x7F	0	-	quit frame to terminate the server (success only if client is authorized as master).
Response Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
reserve	0x80	0	-	free space
⋮	⋮	⋮	⋮	⋮

Tabelle 4.2.: Actuator Protocol v.0.2

Da für diese Anwendungen nur unidirektionale Kommunikation von Bedarf ist, gibt es keine Response Frames. Alle Befehle sind vereint in dem Enum Typ „APCommand_t", der in dem Headerfile „BC_ActuatorProtocol.h" definiert ist. Dieses inkludiert auch wieder das „BC_Protocol.h" Headerfile. Der Addressbereich der Request Frames ist trotz der Unidirektionalität auch hier wie bei dem Measurement Protocol von 0x00 bis 0x7F. Der Quit Command ist auf 0x7F. Um eventuell spätere Ausbaufähigkeit zu ermöglichen und um standardisierte Normung für mehr Übersichtlichkeit zu haben. Kommandos für eine Master Client Autorisation enthält das Protokoll nicht, da die Stellserver für nur einen Clienten gedacht sind.

4.2.3. BC_RelayProtocol (BCRP)

Das Relais Protokoll lässt eine unidirektionale Kommunikation von einem Clienten zu dem „BC_RelayControler_SP" zu, die ein Schalten der Relais initiieren soll. Der Relais Controller bettet auch das „BCAP_ServerMain.c" Sourcefile (Kapitel 5.22) ein und ist ein Stellserver. Das Protokoll des RelayControllers ist jedoch separat entstanden.

Request Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
SetRegister	0x01	1	Pattern	switch all relays in different states with one pattern frame
CloseRelay	0x02	1	Relay nr x	switch only the selected relay to closed state with one frame
OpenRelay	0x03	1	Relay nr x	switch only the selected relay to opened state with one frame
ToggleRelay	0x04	1	Relay nr x	negate the current state from selected relay with one frame
reserve	0x06	0	-	free space
⋮	⋮	⋮	⋮	⋮
Quit	0x7F	0	-	quit frame to terminate the server (success only if client is authorized as master).
Response Frame	**Command Address**	**Arglen** [**uint8**]	**Arguments** [**uint8**]	**Comments**
reserve	0x80	0	-	free space
⋮	⋮	⋮	⋮	⋮

Tabelle 4.3.: Relay Protocol v.0.2

Arguments	Mnemonic	Byte code [uint8]
Relay nr 1	RELAY_VN	0x01
Relay nr 2	RELAY_VP	0x02
Relay nr 3	RELAY_I	0x03
Pattern	RELAY_VN | RELAY_VP | RELAY_I	0b0000.0xxx

Tabelle 4.4.: Arguments of Relay Protocol v.0.2

In dem Headerfile „BC_RelayProtocol.h" sind alle möglichen Befehle in dem Enum Typ „RP-Command_t" und alle möglichen Argumente im Enum Typ „RPArgument_t" definiert. Bei den Argumenten handelt es sich um die Relais, die zum Schalten ausgewählt werden. In dem Headerfile wird das Strompfadrelais „BRCP_RELAY_I" und die zwei Relais zum beid-poligen Trennen der Batteriezelle „BCRP_RELAY_VP" und „BCRP_RELAY_VN" genannt. Für den Fall, dass alle Relais gleichzeitig geschaltet werden sollen, gibt es das Argument „BCRP_RELAY_ALL", welches nur im Zusammenhang mit dem „SetRegister" Frame gesen-det werden kann. Dieser Frame ist der einzige der es zulässt, mehrere Relaiskombinationen als Pattern gleichzeitig in einem Frame zu übertragen. Für weitere Pattern-Konstellationen dieses Frames müssen die Relaisnummern als „One-Hot" Codierung ODER-verknüpft wer-den. Dabei steht eine 1 für schließen und eine 0 für öffnen.

4.3. Implementation

Alle Programme der BatCycle Familie sind in der Sprache C implementiert. Auch wenn diese Sprache nicht objektorientiert ist, können OOP-Paradigmen zum Teil angewendet werden. Die Strukturierung der BatCycle Software hält sich z.B. strikt an das klassische Programmierparadigma, der Kapselung bzw. Verdeckung von Teilaufgaben.

Alle Dateien der Softwareentwicklung sind in dem Archiv „BatCycle_Dev" unkomprimiert gebündelt. Entpackt wird dieses mit dem Konsolenaufruf: „tar xf Projects.tar" oder mit Rechtsklick auf „entpacken". Es entsteht ein Ordner „Projects" mit drei Unterordnern „bin" für die binär Dateien, „src" für den Quellcode, symbolische Links, Makefile und „codeblocks_projects" für die IDE-Projektdateien. Bevor der Ordner „BatCycle_Dev" auf andere Plattformen übertragen werden soll, muss dieser wieder mit „tar cvf Projects.tar Projects" gepackt werden, da symbolische Links in diesem Projekt verwendet werden. Diese gehen auf anderen Plattformen, wie Windows, verloren.

Mit den symbolischen Links werden dem Makefile und Codeblocks ermöglicht, von unterschiedlichen Pfaden aus, auf eine verlinkte Datei oder einem Ordner lesend zuzugreifen. Dadurch werden Änderungen in den verlinkten Sourcefiles in allen Programmen die diese benutzen wirksam. Ansonsten müssten mehrfach gleiche Dateien auf dem Filesystem für jedes Programm einzeln hinterlegt werden. Bei Änderungen, müsste so in jedem Programmordner das kopierte File geändert werden.

Der Quellcode jeglicher BatCycle Programme kann mit der Linux/Windows Cross-Plattform IDE „Codeblocks" bearbeitet und kompiliert werden. Bei einem neu aufgesetzten Betriebssystem muss diese erst mit einer Konsoleneingabe „sudo apt-get install codeblocks" installiert werden. Dann müssen entsprechende Einstellungen von Codeblocks vorgenommen werden, um die BatCycle Programme linken und kompilieren zu können. Dazu müssen in dem Einstellungsfenster unter:

Settings → Compiler and Debugger → Linker Settings

die „bcm2835", „lrt" und „pthread" Bibliotheken eingetragen werden. Diese sind für den Zugriff auf die GPIO Ports des RPI2 Prozessors, für die Semaphore-Funktionen und für den POSIX Timer verantwortlich. Ein Beispiel ist in Abbildung 4.3 zu sehen.

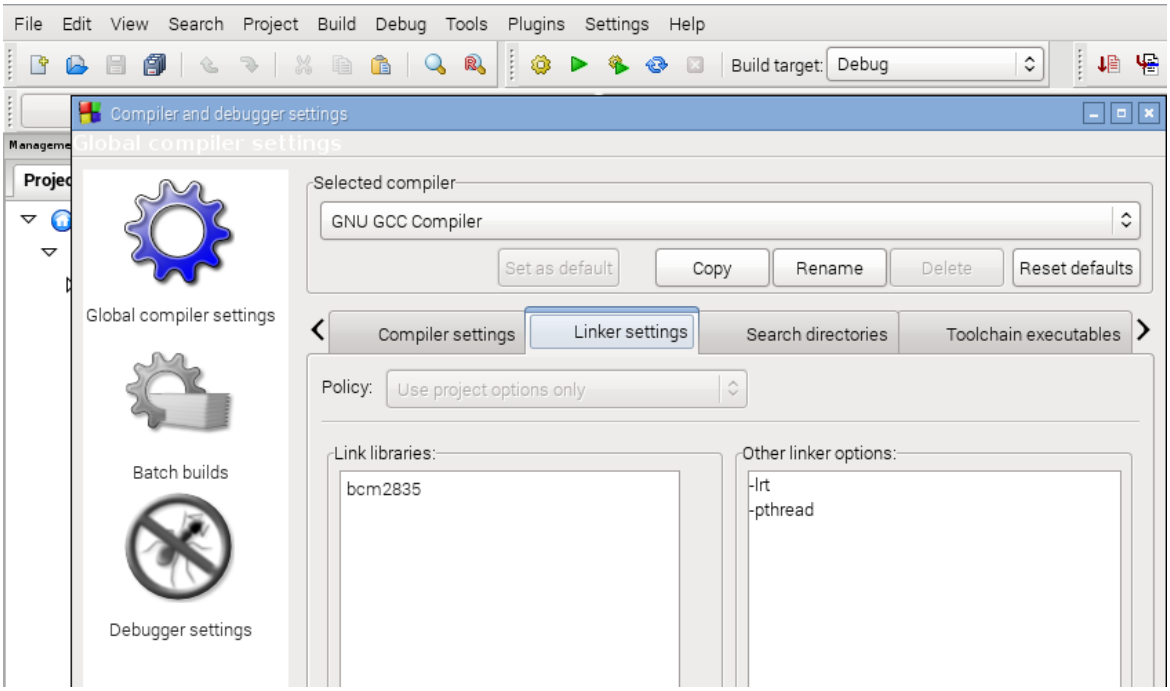

Abbildung 4.3.: Linker Einstellungen

Eine komfortablere Möglichkeit alle BatCycle Programme auf einmal kompilieren zu können, wird durch ein Makefile realisiert. Die drei Bibliotheken werden hier ebenfalls mit gelinkt. Die Ausführung des Makefiles erfolgt für die Release-Versionen per Konsole mit:

→ make

Und für die Debug-Versionen mit:

→ make debug

und anschließend, wenn die erstellten Binärdateien im Ordner „src/bin" an den richtigen Ort kopiert wurden:

→ make clean

um diese und die Objectfiles wieder zu löschen.

4.3.1. Measurement Servers

Alle Messserver (die der Onboard ADC sowie der DMM4020Reader) bedienen sich über die symbolischen Links der Source-und Headerfiles „BCMP_Server.c", „BCMP_Server.h" und „BCMP_ServerMain.c". So werden Teilaufgaben gekapselt und gleiche Programmteile nicht in unterschiedlichen Sourcefiles wiederholt eingefügt. Diese drei Dateien bilden das Hauptprogramm des Messservers. Dieser ist fest auf das entsprechende „BC_MeasurementProtocol" zugeschnitten. Für die unterschiedliche Ansprache von Hardwarekomponenten der Messplatine, ist das Sourcefile „BC_HW_ADC_I.c" als Implementationsbereich angelegt. Über externe Interface-Funktionen wird dieser individuelle Bereich der Mainfunktion zugänglich gemacht:

- void usage(char *progname)
- int init_hardware(int argc, char* argv[])
- double getValue(void)
- void reset_hardware(void)

Der von diesem Serverprogramm angelegte Unix-Socket wird im Streaming und Non-Blocking Mode instanziiert. Mittels der POSIX Funktion „select(..)" wird jeder offene Deskriptor auf eingehende Nachrichten abgefragt. Bei leerem Eingangsbuffer wird von der „select-Funktion" der Prozess für einen Timeout schlafen gelegt. Bei nicht leerem Buffer wird eine dem Protokoll entsprechende Antwort zurückgeschickt.

In folgender Abbildung ist das Hauptprogramm der Messserver mit einem Nassi-Schneiderman Struktogramm stark vereinfacht dargestellt.

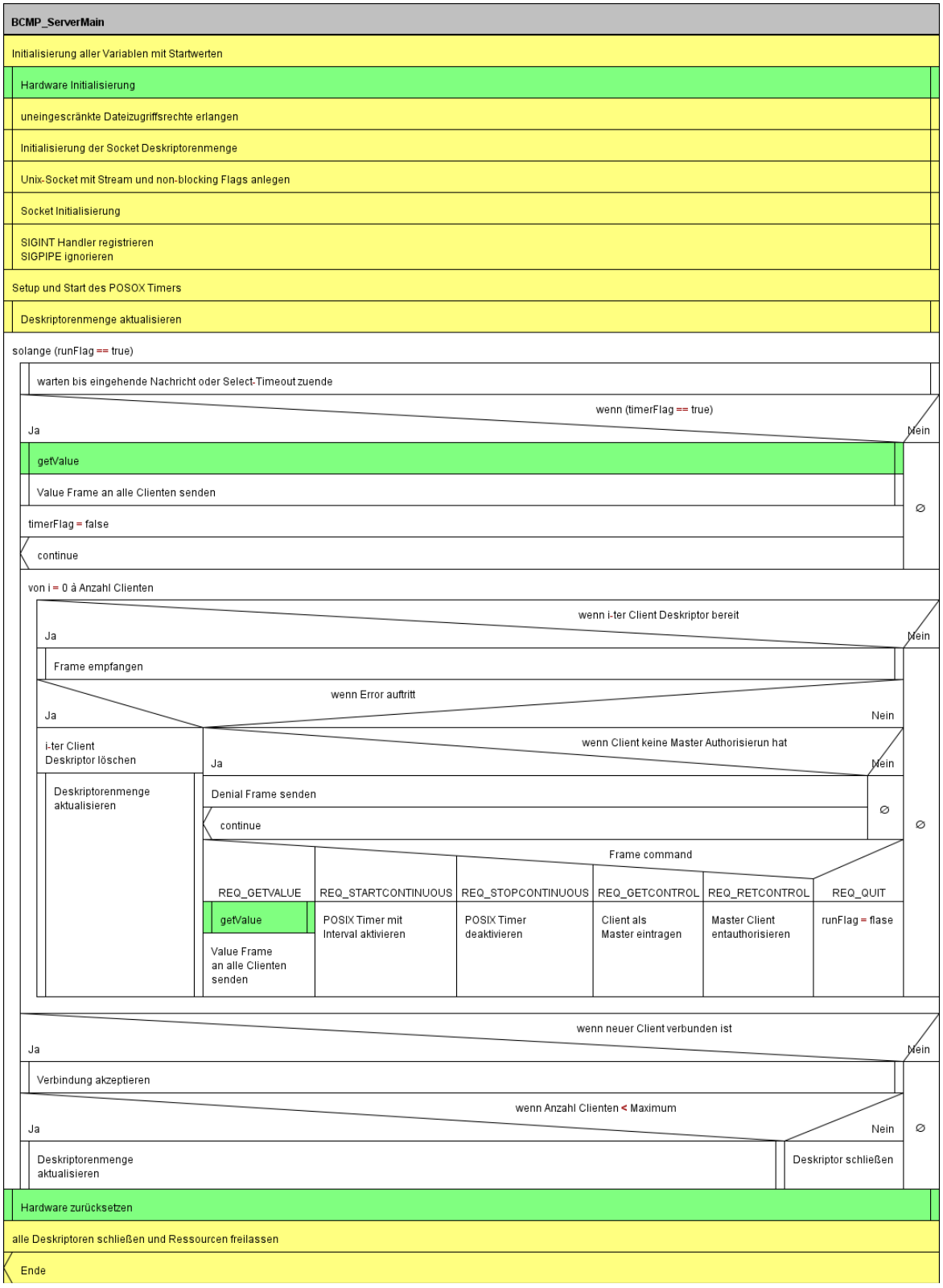

Abbildung 4.4.: BCMP_ServerMain (gelb: Initialisierung und Freigabe, grün: Interface Funktionen)

4.3.2. Actuator Servers

Die Stellserver „BC_RelayController_SP", „BC_VoltageController_SP" und „BC_Potentiometer_SP" werden mit Hilfe der Source-und Headerfiles „BCAP_Server.c", „BCAP_Server.h" und „BCAP_ServerMain.c" kompiliert. Diese halten sich an das ActuatorProtocol und sind dementsprechend auf dieses zugeschnitten. Auch wie bei den Messservern ist hier ein freier Implementationsbereich für unterschiedliches Bauteilverhalten geschaffen. Die externen Interface-Funktionen dafür sind:

- void usage(char *progname)
- int init_hardware(int argc, char* argv[])
- int behavioral(BCFrame_t *inputframe)
- void reset_hardware(void)

Der Zugriff auf die drei Source-und Headerfiles von einem Stellprogramm aus wird auch über symbolische Links ermöglicht.

Der Unterschied zum Programmablauf der Messserver ist, dass die Stellserver auf das ActuatorProtocol ausgelegt sind und daher keinen „Continuous Mode" und keinen POSIX Timer benötigen.

Der Programmablauf ist in dem Nassi-Schneiderman Struktogramm 4.5 veranschaulicht.

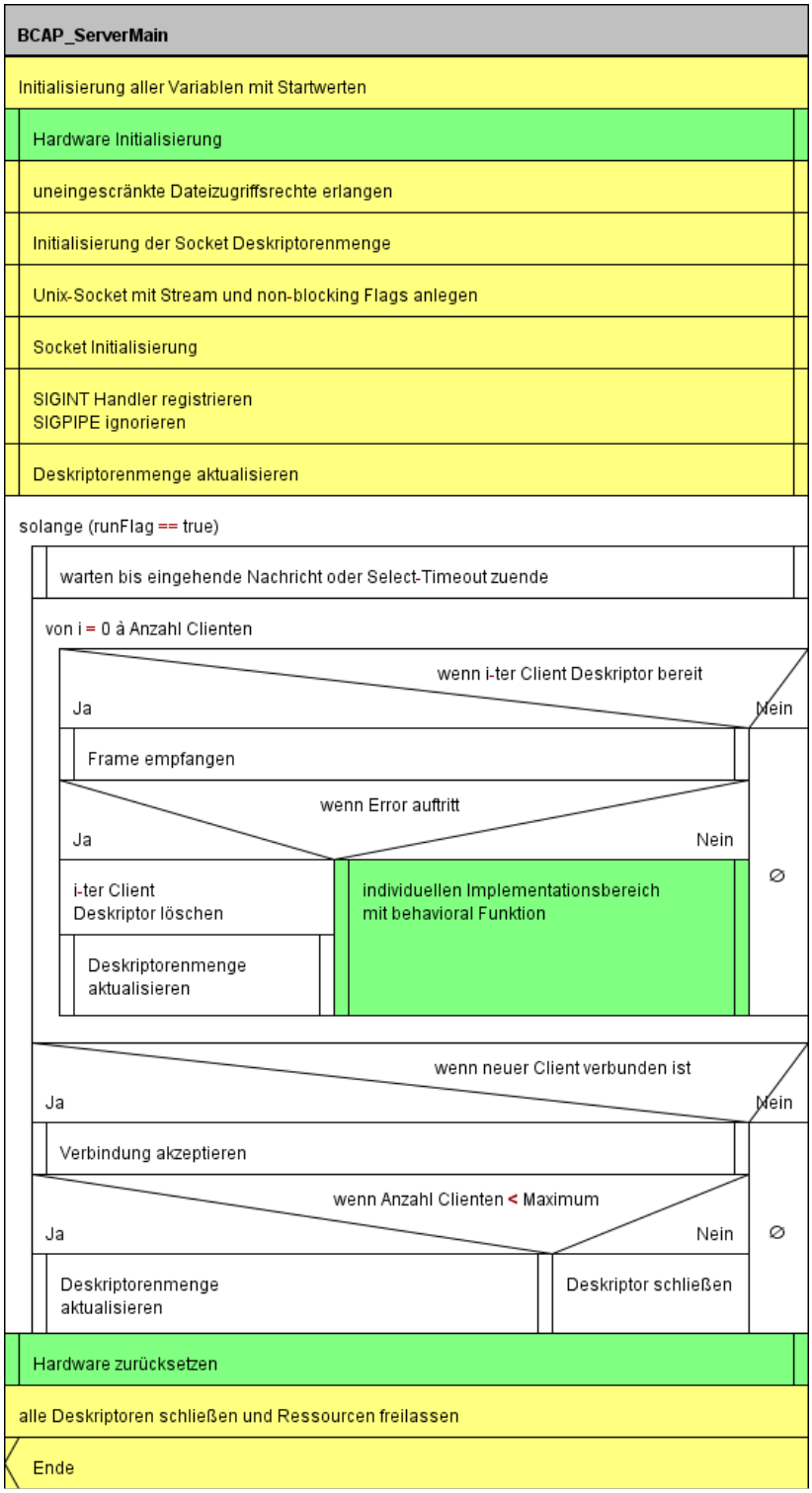

Abbildung 4.5.: BAMP_ServerMain (gelb: Initialisierung und Freigabe, grün: Interface Funktionen)

4.3.3. MainController Client

Das „BC_MainController_CP" Programm liest in der Initialisierungsphase ein Initfile und ein Paramfile ein. Mit Einbindung der Config Parser Library von Marc A. Lindner werden Syntax-Checks und Datensatzerkennung nach dem Einlesen vorgenommen. Die Parameterwerte aus den Dateien werden in lokalen Structs gespeichtert. Diese werden per Call by Reference an Unterfunktionen wie die Regel-Algorithmen übergeben, um von dort auf die Werte zuzugreifen. Mit einer global definierten runFlag soll es möglich sein, durch ein SIGINT Signal (Str + c) den Instruction Pointer (auch Programmzähler genannt) aus jeder Schleife aussteigen zu lassen, um das Programm zum geordneten Ende zu bringen. Wie bei den Mess- und Stellservern werden die Unix-Sockets als Streamfile geöffnet. Ein Unterschied ist jedoch, dass die Sockets auf Blocking Mode mit einem Timeout von einer Sekunde eingestellt werden. Sollte ein Messserver aus sonstigen Gründen terminiert worden sein oder nicht mehr reagieren, wird das vom MainController Client Programm mit einer Fehlermeldung auf der Konsole quittiert. Als Erweiterungsidee könnte der MainController aber an der Stelle auch dahingehend automatisiert werden, dass über ein Überwachungsprozess der abgestürzte Server neu instanziiert wird und der MainController einen Reconnect versucht.

Ein Beispiel des Initfiles und des Paramfile sind im Anhang A.1 und A.2 zu entnemen. In beiden Dateien ist der erste einzulesende Parameter die Version der Datei. Sie sollte nur erhöht werden, wenn neue Parameter und Funktionalitäten hinzukommen, bzw. an der Struktur etwas geändert wird. Bei einer Versionserhöhung muss auch der zuständige Quellcode „bc_config.h" und „bc_config.c" angepasst und dessen Versionskonstanten „PROG_VERSION_INIT" und „PROG_VERSION_PARAM" erhöht werden. Anzumerken ist, dass die Versionsnummern der Configfiles immer kleiner gleich der Quellcode Versionskonstanten sein sollten. Dadurch wird eine Abwärtskompatibilität aufrechterhalten. Bei Nichteinhalten dieser Regel wird dem Benutzer eine Fehlermeldung vom MainController ausgegeben.

Des Weiteren besteht das Initfile aus der Parametergruppe „Inputgroup" zur Socket-Initialisierung und einer „BCHWgroup" in der hardwarespezifische Werte der Messplatine eingetragen werden. In dem Paramfile kommt nach dem Versionsparameter eine Parametergruppe „batterygroup". In diese werden batteriespezifische Angaben wie Typ und maximaler Innenwiderstand eingetragen. Letzterer ist nur relevant für die cV-Regelung. In der nächsten Gruppe „Limitgroup" werden Ober- und Untergrenzwerte für Spannung, Strom und Temperatur eingetragen, die von den Regelalgorithmen nicht überschritten werden dürfen.

Der vollständige Zyklierablauf ist in der „Cyclegroup" Parametergruppe enthalten. Sie ist aufgeteilt in die Untergruppen „Startphase", um die Zelle von einem unbekannten Zustand auf ein definiertes Spannungsniveau zu bringen und einer ‚Cyclephase" Gruppe, die den eigentlichen Zyklierablauf mit Intervall gesteuerter Entladung und Ladung vorgibt. Die „Cyclephase" Gruppe enthält eine Parameterliste „Cyclelist", die von oben nach unten vom Programm

abgearbeitet wird und einen Parameter „Cycles”, der die Anzahl an Wiederholungen dieser Liste angibt. Die Zyklierparameter sind in immer gleichen Parameterblöcken enthalten. Das folgende Listing ist ein Parameter Set eines Zyklierablaufs aus einer bc_param.cfg Datei.

```
1
2  cyclelist = (
3      {                           //Entladung mit konst. Strom
4          mode                = "cC";     Lade-/Entlade Modus: "cC", "cV" oder "cCcV"
5          mode_time           = 600;      Zeitliche Modus Begrenzung in [s]
6          threshold_voltage   = 2.8;      Lade-/Entladeschlussspannung in [V]
7          nominal_current     = -10.0;    Bei cC-Mode: Lade-/Entlade Sollstrom
8                                          Bei cV-Mode: Minimalstrom für Beendigungskriterium
9          convalescence       = 300;      Erholungszeit in [s]
10         led_R               = 100.0;    Led-Rot-Wert in [%]
11         led_G               = 50.0;     Led-Grün-Wert in [%]
12         led_B               = 0.0;      Led-Blau-Wert in [%]
13     },
14     {                           //Entladung mit konst. Strom
15         mode                = "cC";     Lade-/Entlade Modus: "cC", "cV" oder "cCcV"
16         mode_time           = 600;      Zeitliche Modus Begrenzung in [s]
17         threshold_voltage   = 2.8;      Lade-/Entladeschlussspannung in [V]
18         nominal_current     = 10.0;     Bei cC-Mode: Lade-/Entlade Sollstrom
19                                         Bei cV-Mode: Minimalstrom für Beendigungskriterium
20         convalescence       = 300;      Erholungszeit in [s]
21         led_R               = 100.0;    Led-Rot-Wert in [%]
22         led_G               = 50.0;     Led-Grün-Wert in [%]
23         led_B               = 0.0;      Led-Blau-Wert in [%]
24     }
25 )
```

Listing 4.1: CycleParamSet.cfg

Der Programmablauf ist gut dargestellt in folgendem Nassi-Schneiderman Struktogramm.

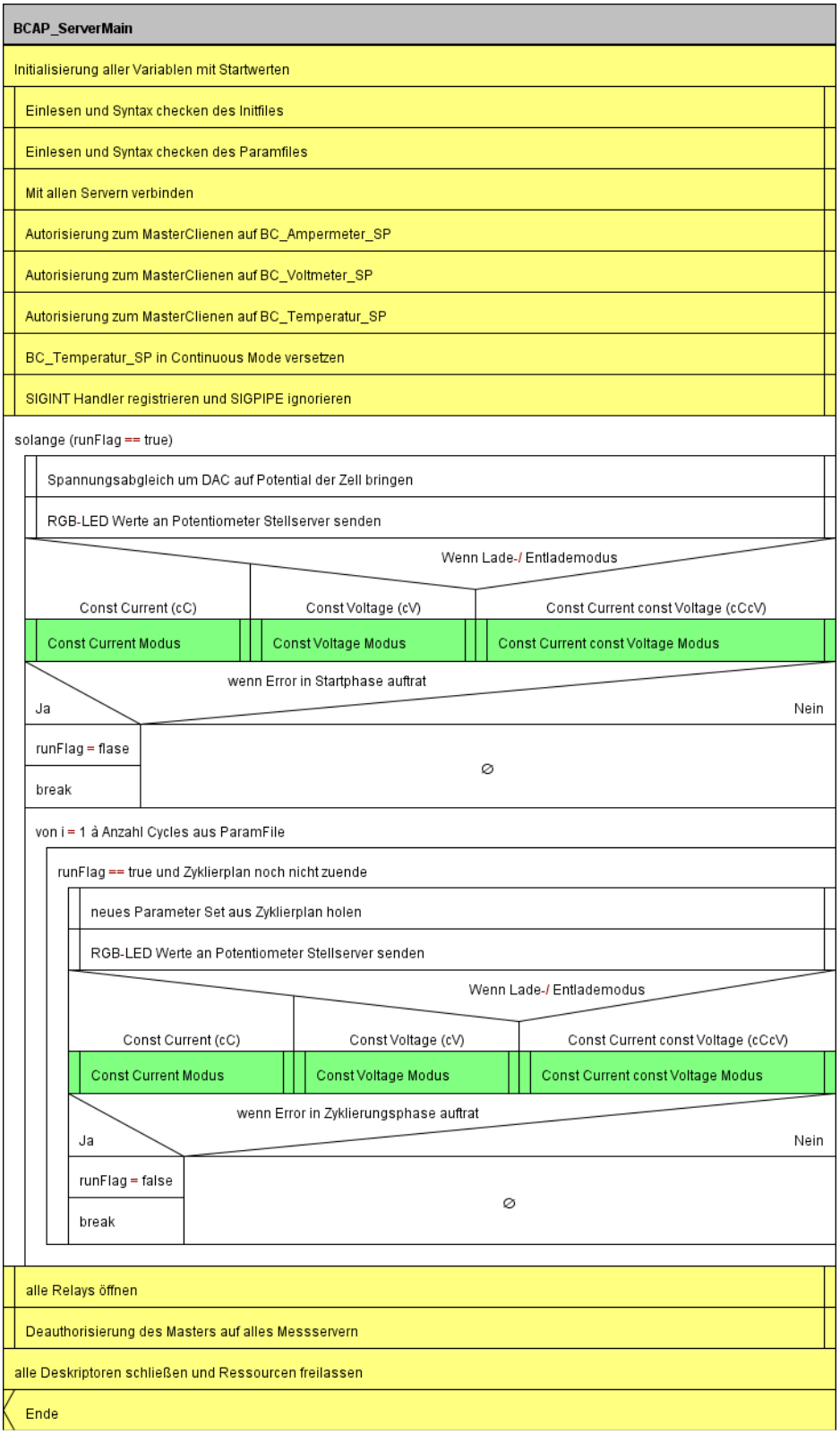

Abbildung 4.6.: MainCOntroller (gelb: Initialisierung und Freigabe, grün: Interface Funktionen)

cC-Regelung

Die Ladung-/ Entladung mit konstanter Stromregelung ist in der Funktion

int cC_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);

implementiert. Sie regelt den Strom auf einen konstanten Sollwert, der im Paramfile mit „nominal_current" angegeben wird. Zu Beginn wird die Ist-Spannung gemessen und mit der Lade-/Entladeschlussspannung verglichen. Wenn die Ist-Spannung größer als die Schlussspannung ist, wird die Zelle entladen und wenn sie kleiner ist, wird die Zelle aufgeladen.

Der Algorithmus zur Stromregelung realisiert einen vierstufigen Schwellwertregler. Dessen Funktion ist es, in jeder Schleifeniteration die Stromdifferenz von Ist-Strom zu Soll-Strom zu ermitteln. Der Betrag der Differenz wird dann in 4 Stufen zugeordnet und mit einem der Stufe entsprechenden Stellschritt gegengeregelt. Die Faktoren der Stufen und Stellschrittweiten sind in dem Headerfile „BC_Regulation.h" definiert und können mit wenig Aufwand neu angepasst werden.

Bei dieser Funktion beginnt der Countdown in Sekunden, der mit „mode_time" im Paramfile angegeben wird, ab dem ersten Schleifendurchlauf. Wenn für den Parameter der Wert 0 eingetragen ist, wird der Countdown deaktiviert und der cC-Modus beendet sich erst, wenn die Ist-Spannung gleich der Lade-/Entladeschlussspannung ist.

Die Stufen und Stellschritte sind in dem folgenden Nassi-Schneiderman Struktogramm Abbildung 4.7 in hellblau und gelb markiert. Die Beendigungskriterien sind pink markiert.

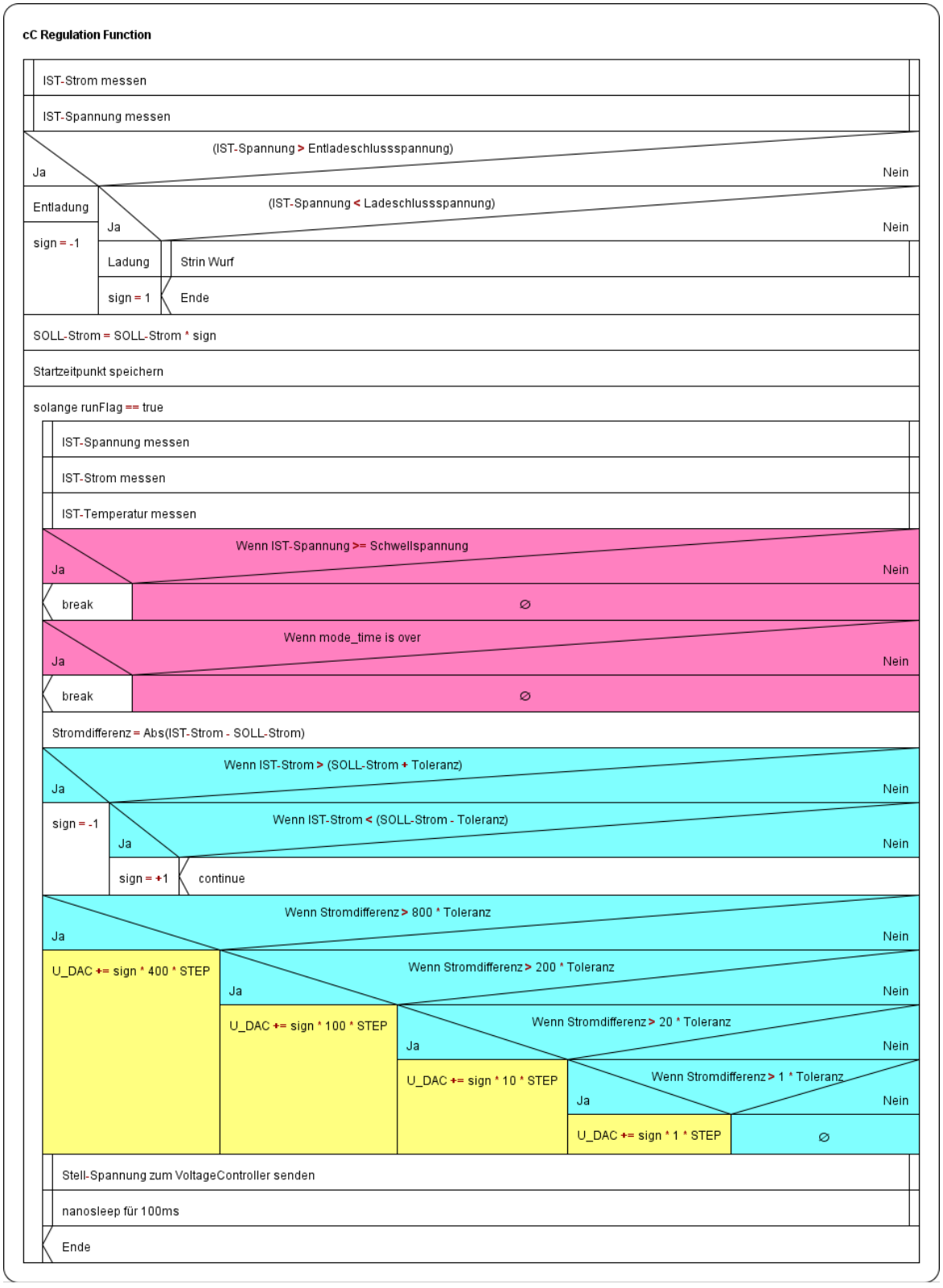

Abbildung 4.7.: cC-Regelungs Funktion (pink: Ist-Wert Überwachung, hellblau: Regelstufen, gelb: Stellschritte)

cV-Regelung

Die cV-Regelug ist implementiert in der Funktion:

int cV_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);

Der Algorithmus regelt die Batteriespannung auf einen aus dem Paramfile gelesenen konstanten Sollwert. Weil bei einer cV-Regelung ein zu hoher Strom fließen und die Batterie schädigen könnte, ist dagegen eine Sicherheit eingebaut. Es wird heuristisch ein potentieller Strom zuvor berechnet, indem ein geschätzter Toleranzwiderstand mit dem Messwiederstand (Shunt) der Platine addiert und dann durch die Spannungsdifferenz von Soll-Spannung zu Ist-Spannung dividiert wird. Dies verdeutlicht Gleichung 4.1.

$$R_{ges} = R_{mess} + R_{tol} \tag{4.1}$$

$$I_{heu} = \frac{U_{Soll} - U_{Ist}}{R_{ges}} \tag{4.2}$$

Der Toleranzwiderstand stammt aus dem Paramfile und der Messwiderstand aus dem Initfile. Wenn der errechnete Strom kleiner als der zulässige maximal Strom „max_current” ist, wird auf Soll-Spannung geregelt. Wenn der Schätzstrom jedoch größer als zulässig ist, wird die Stell-Spannung um das Produkt von maximal zulässigem Strom und dem Gesamtwiderstand herabgesetzt.

$$U_{Stell} = U_{Ist} + I_{max} * R_{ges} \tag{4.3}$$

Als weitere Sicherheit wird der Ist-Strom in jeder Schleifeniteration überwacht. Falls dieser tatsächlich den maximal zulässigen Strom überschreitet, wird der Gesamtwiderstand um 10% verringert. Das führt in der heuristischen Stromberechnung dann zu einem höheren Schätzstrom und damit zu einer Herabsetzung der Spannung U_{Soll}, die einen kleineren Strom fließen lässt.

Nach einem festen Zeitintervall und wenn der Ist-Strom im zulässigen Bereich liegt, wird der Messwiderstand und die Ausgangsspannung wieder auf Anfangswerte gesetzt. So kann die Batterie nach dem Intervall mit konstant geregelter Soll-Spannung aus dem Paramfile geladen bzw. entladen werden.

Das Beendigungskriterium des cV-Regelungs Algorithmus ist, dass der Ist-Strom kleiner als der nominale Strom des Paramfiles sein muss. Ab dann regelt der Algorithmus für den im Paramfile angegebenen Parameter „mode_time" weiter und beendet sich dann. Wenn bei „mode_time" 0 eingetragen ist, beendet sich die Funktion sofort nach dem Unterschreiten des nominalen Stromes.

Ein Nassi-Schneiderman Struktogramm ist in Abbildung 4.8 zu sehen. Die Regelschritte sind in hellblau und gelb markiert. Das Beendigungskriterium ist pink markiert.

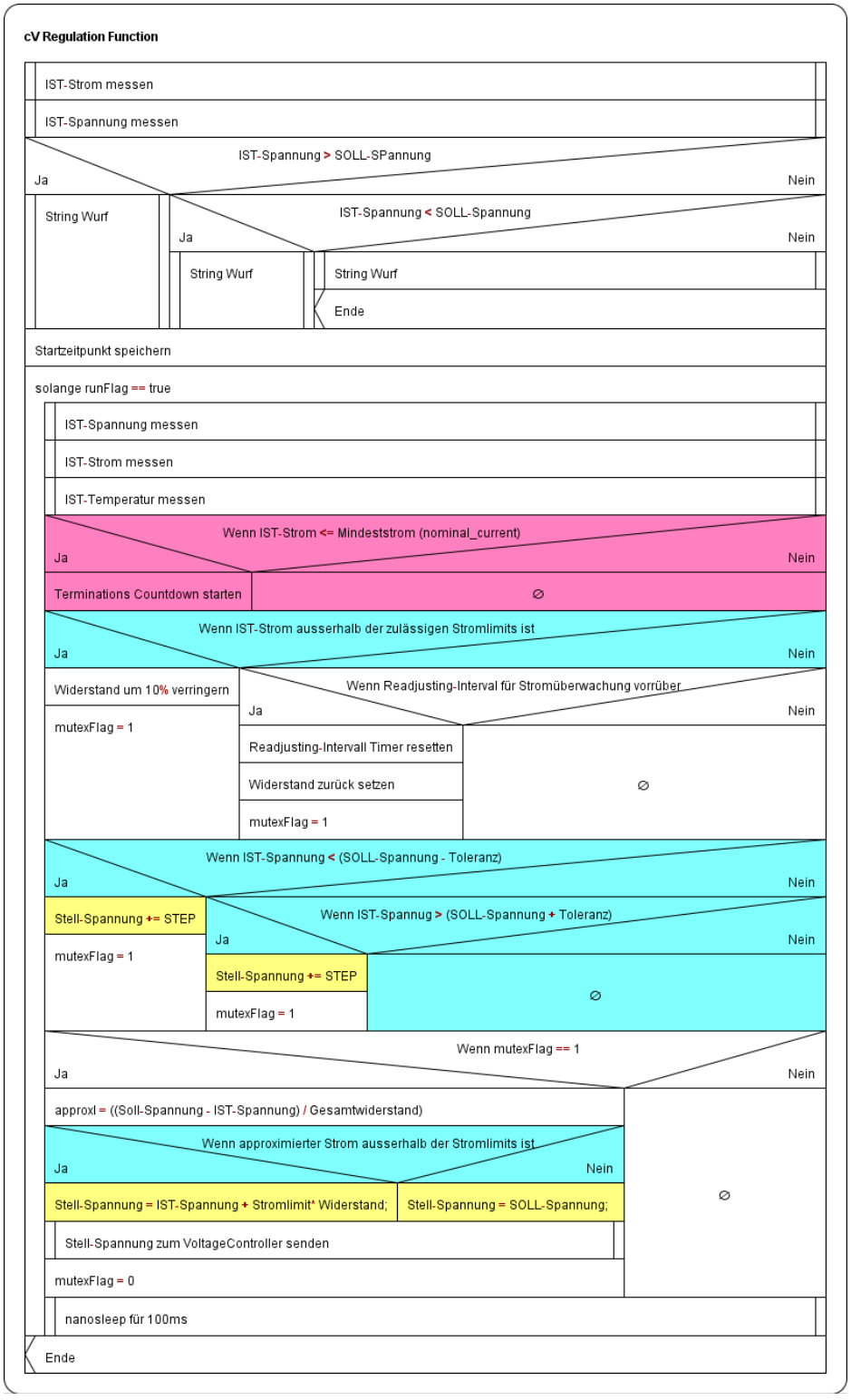

Abbildung 4.8.: cV-Regelungs Funktion (pink: Beendigungskriterien, hellblau: Ist-Wert Überprüfungen, gelb: Stellschritte)

cCcV-Regelung

Die cCcV-Regelung ist im Grunde eine sequentielle Ausführung der cC-Funktion und der cV-Funktion. Eine kleine Anpassung des Parameters „mode_Time" wird jedoch gemacht, da ein Timeout hier erst im cV-Modus sinnvoll ist. Der Parameter wird daher für den Aufruf der cC-Regulation Funktion auf 0 gesetzt und vor Beginn der cV-Regulation Funktion wieder auf den ursprünglichen Wert im Paramfile gesetzt.
Ebenso wird der Parameter „nominal_current" für die cC-Phase als Soll-Strom benutzt und für die cV-Phase als Mindest-Strom für das Beendigungskriterium. Der Mindest-Strom der cV-Phase beträgt immer 10% des nominalen Stroms. Jedoch nur, wenn dieser nicht kleiner $|10\mu A|$ beträgt, ansonsten wird als Mindest-Stromgrenze $|10\mu A|$ verarbeitet.

5. Erprobung und Tests des Gesamtsystems

In diesem Kapitel werden abschließende Erprobungen und Testläufe des Gesamtsystems analysiert. Dazu gehören funktionelle Tests für die Lade-/Entlademodi, ein Test des Temperatursensors und eine vollständige Zyklierung einer ITO-Zelle im cCcV-Modus mit optischer Messung. Die durch die Lade-/Entlade Moditests ersichtlichen Artefakte und Abweichungen werden anschließend mit Hilfe des Korrekturprogramms „LinTest" behoben. In Abbildung 5.1 ist eines von sechs Zykliersystemen bei offenem Gehäuse zu sehen.

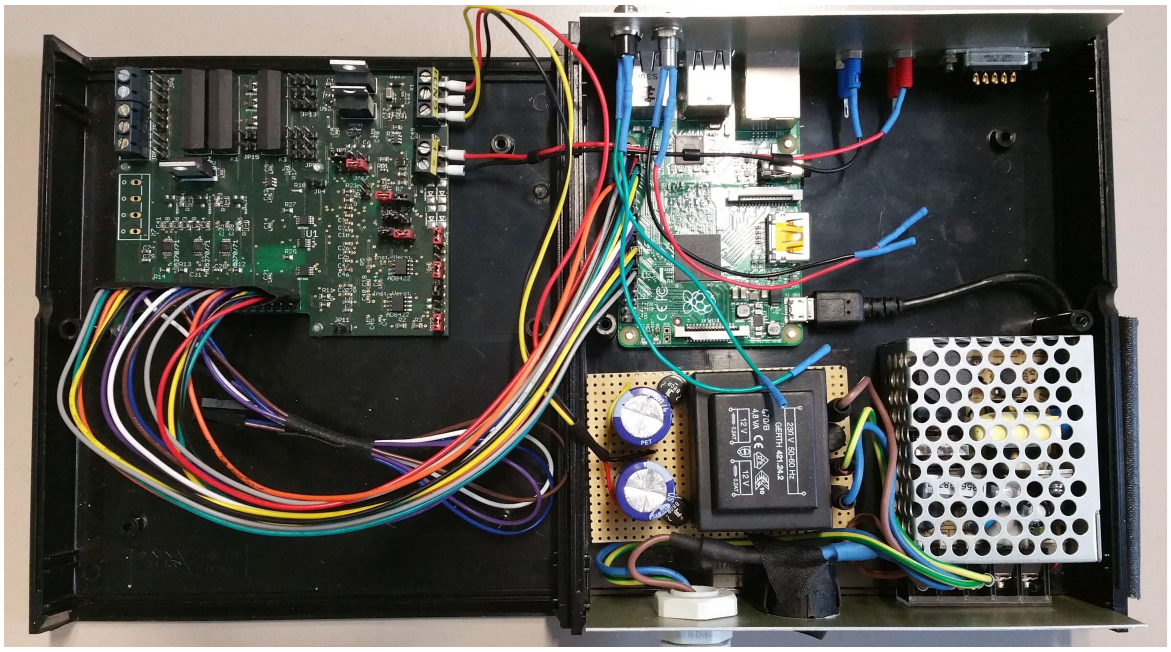

Abbildung 5.1.: Gesamtes Zykliersystem

Oben links auf dem Deckel ist die von Herrn Geist entwickelte Messplatine montiert. Auf dem rechten Gehäuseboden ist oben der Raspberry Pi 2 an die Gehäuse Schnittstelle montiert. Darunter liegen links das Netzteil für die Messplatine und rechts daneben ein Schaltnetzteil für den Einplatinen Computer.

5.1. Finale Systemtests

5.1.1. cC-Regelung

Eine Testzyklierung im cC-Modus mit zwei BC_DMM4020Reader als Strom- und Spannungs-Messserver ist in folgender Abbildung 5.2 zu sehen. Als Testzelle dienen hier zwei serielle 1,55V Silberoxid-Zink-Knopfzellen von Renata Batteries. Zykliert wird zweimal in der Reihenfolge $\pm 1\mu A$, $\pm 2\mu A$, $\pm 5\mu A$, $\pm 10\mu A$, $\pm 20\mu A$, $\pm 50\mu A$, $\pm 100\mu A$, $\pm 200\mu A$, $\pm 500\mu A$, $\pm 900\mu A$. Eine Lade-/Entladephase beträgt 10 Minuten und die Erholungszeiten dazwischen 5 Minuten (grün markiert).

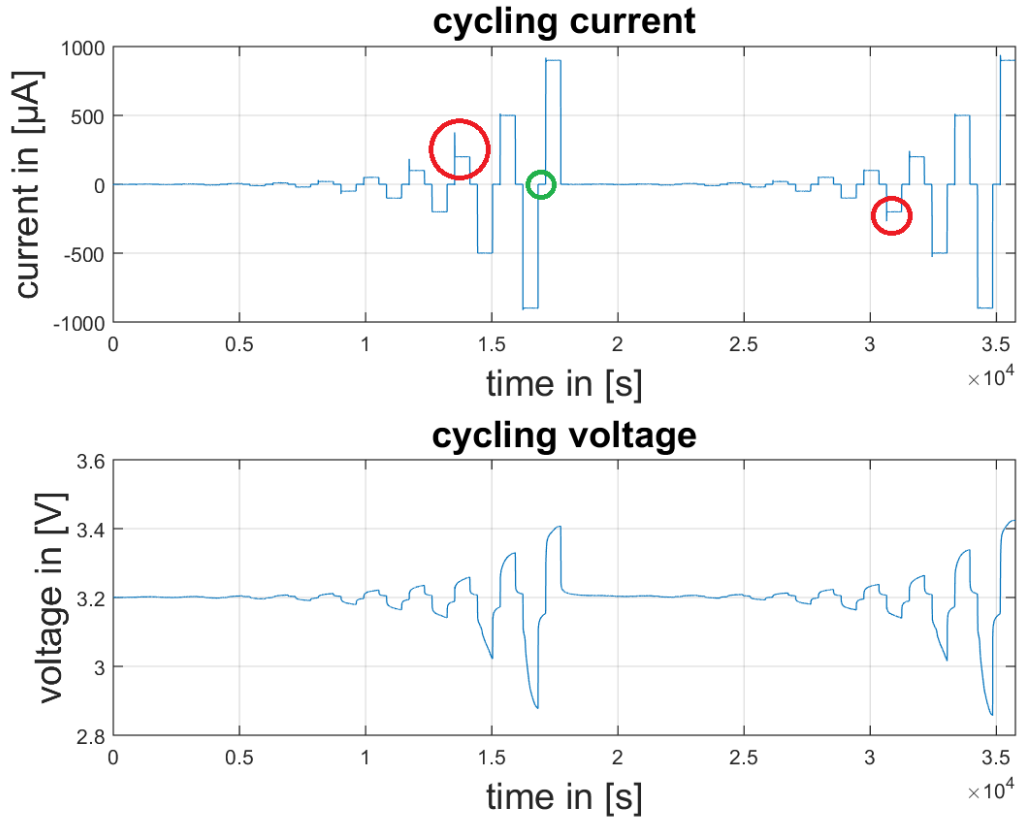

Abbildung 5.2.: Zyklierung zweier serieller Knopfzellen mit cC-Regelung von $1\mu A$ bis $900\mu A$, Messaufbau mit Tektronix DMM4020 Multimetern

In dem oberen Kurvenverlauf des Stromes sind Überschwinger der Regelung zu erkennen (rote Markierungen). Diese sind auf das Verhalten des cC-Regelungsalgorithmus und auf

das Zeitglied des Regelkreises zurückzuführen. Dieses ist von der inneren Zellimpedanz und von den Eingangsfiltern der Messglieder stark abhängig.

In einer weiteren Zyklierung, diesmal mit den Onboard ADC als Strom- und Spannungs-Messglieder, sind solch große Regelungsüberschwinger nicht mehr enthalten. Der Grund ist die kleinere Zeitkonstante des Strom-ADC Eingangsfilters, im Gegensatz zu der des DMM4020 Messgerätes im Gleichstrom-Messmodus. Dieser geht mit einer größeren Zeitkonstante einher.

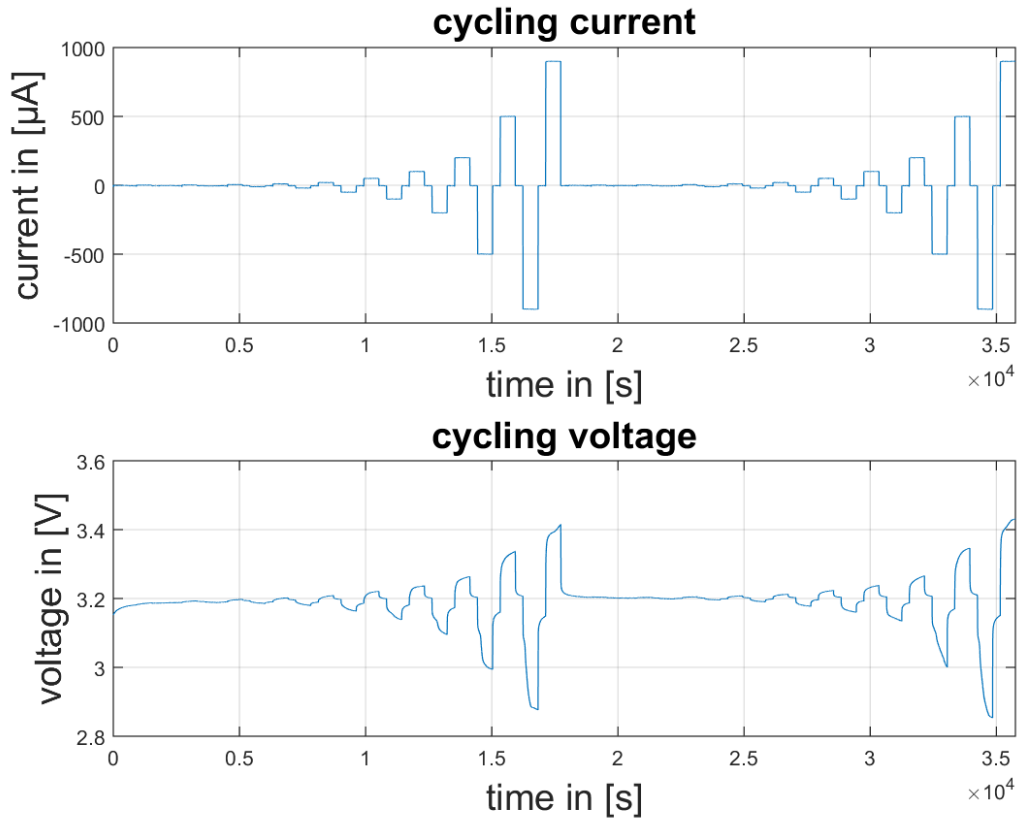

Abbildung 5.3.: Zyklierung zweier serieller Knopfzellen mit cC-Regelung von $1\mu A$ bis $900\mu A$, gemessen mit den Onboard ADC

5.1.2. cCcV-Regelung

Für einen Test des cCcV-Modus ist ein Paramfile mit nur einer Lade- und Entladephase im cCcV-Modus in der Cyclelist angelegt, die sich viermal wiederholt. Die Testzellen sind

hier dieselben wie im cC-Modus Test und wurden mit zwei BC_DMM4020_Readern gemessen. Der Soll-Strom der cC-Lade-/Entladephase beträgt $\pm 500\mu$A. Für die anschließende cV-Phase beträgt die Ladeschlussspannung 3,5V und die Entladeschlussspannung 2,9V. Nach den beiden Phasen folgt die Ruhephase, die 5 Minuten dauert. Dieser kurze Moment ist grün markiert.

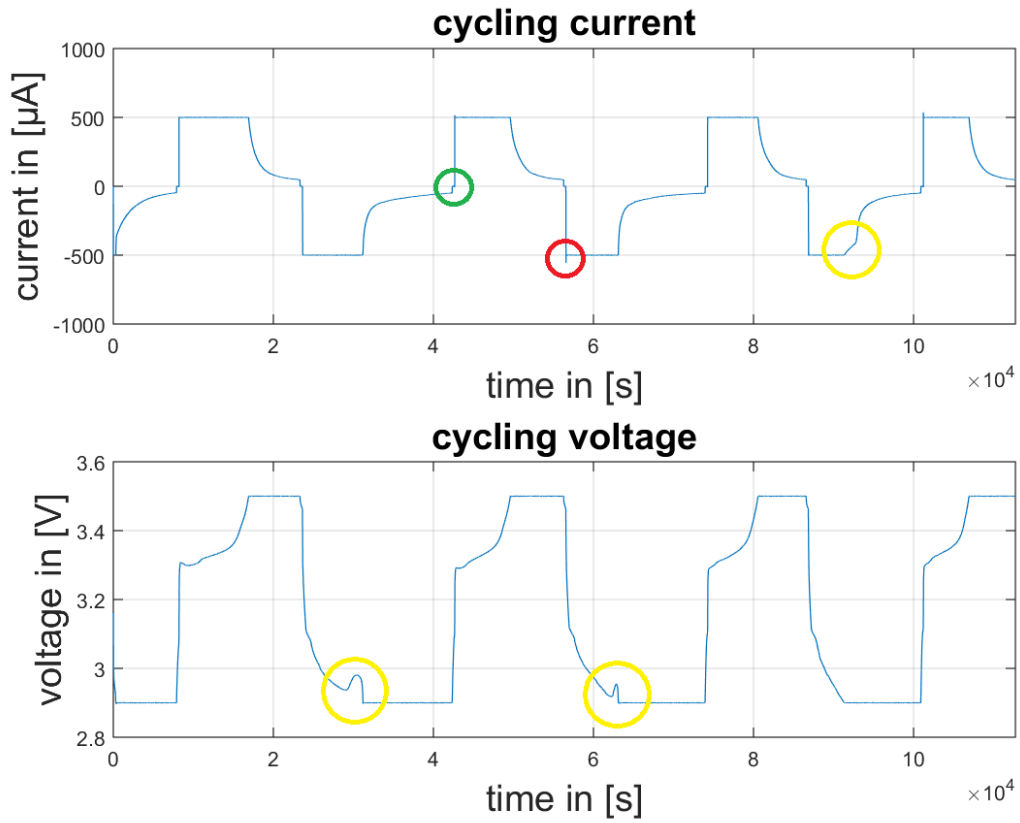

Abbildung 5.4.: Zyklierung zweier serieller Knopfzellen mit cCcV-Regelung von $\pm 500\mu A$, Messaufbau mit dem Tektronix DMM4020 Multimetern

Wie im vorherigen Test sind auch hier Regelschwingungen in der cC-Phase enthalten (rote Markierung). Des Weiteren fallen Artefakte im Spannungsverlauf während der cC-Entladephasen auf (gelbe Markierungen). Es wird angenommen, dass diese zellchemischer Natur sind.

5.1.3. Temperatursensor

Um die Temperatur Datenerfassung auf Funktionalität zu testen, wurde eine weitere Zyklierung im cC-Modus gefahren. Die Strom- und Spannungs Messwerte stammen hierbei von den externen DMM4020 Messgeräten. Der mittlere Temperaturverlauf hat keinen Zusammenhang mit der wirklichen Zelltemperatur, weil der Temperatursensor nur probeweise, freiliegend angeschlossen wurde.

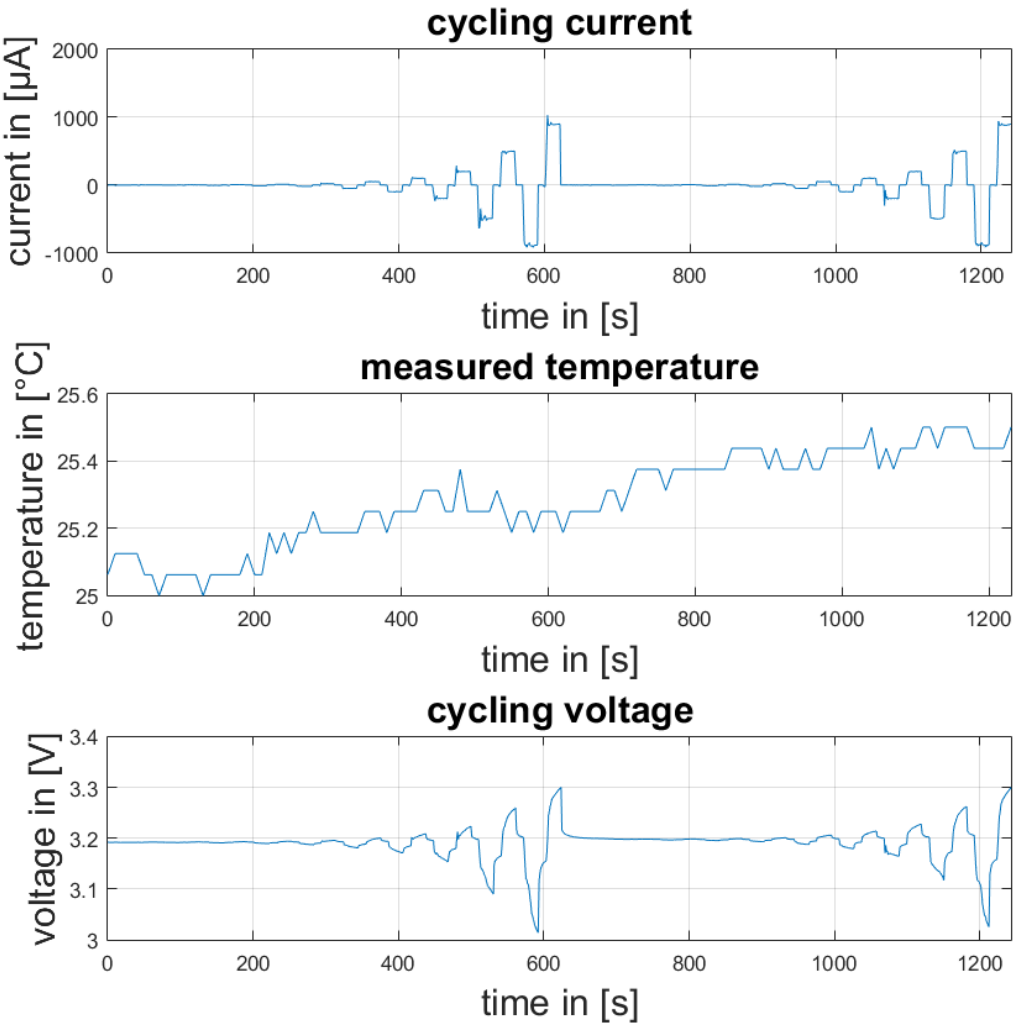

Abbildung 5.5.: Test der Temperatur Datenerfassung mit LM74 SPI/Microwire Sensor

Im Verlaufe weiterer Testzyklierungen im Zusammenspiel von Temperatursensor und Onboard ADC, stellte sich heraus, dass keine korrekten Strom- und Spannungswerte mehr vom Raspberry Pi 2 empfangen wurden. Damit konnte keine zuverlässige Regelung mehr stattfinden. Eine Analyse der seriellen Daten auf dem SPI Bus führte zu folgenden Erkenntnissen: Der Temperatursensor sendet nach jeder Temperaturwandlung seine Werte über die MISO Leitung an den Raspberry PI 2. Dies geschieht ununterbrochen, sobald er aktiviert ist. Dadurch werden alle anderen SPI Teilnehmer, die über die MISO Leitung kommunizieren, in ihrer Kommunikation gestört. Dies betrifft in dem Fall die beiden ADC der Messplatine. Der Temperatursensor lässt sich laut Datenblatt [Quelle: Temperatursensor] in einen Shutdown- oder einen Continuous Conversion Modus versetzen. Ein Datenframe besteht aus 32 Bits, von denen die ersten 16 als Eingangsbefehl und die nächsten 16 als Ausgangs-Temperaturwert verstanden werden. Da er aber MISO und MOSI Leitung zu einer Halbduplex Datenleitung SI/O vereint, ist dieser Temperatursensor für diese Anwendung nicht geeignet. Denn auf der Messplatine sind MOSI und MISO Leitung mit Absicht voneinander getrennt.

5.1.4. optische Messungen

Für einen Test der periodischen Bilddatenaufnahme zur optischen Messung wurde eine ITO-Forschungszelle mit cCcV-Regelung von 1,4V in 0,2V Schritten auf 4V geladen. Der Timeout von 10 Minuten wird gestartet, wenn der Strom unterhalb 10μA beträgt. Es wurden vier Zyklen gefahren. Dies ist zugleich ein Test des Gesamtsystems, da alle Komponenten wie Bilddatenaufnahme, Messungen, Regelung, etc. für diesen Test zusammen funktionieren müssen. Die Kamerabeleuchtung stammt weiterhin von drei konstant hellen Weißlicht LED, die zwar neben der Kameralinse montiert sind, aber nicht über den BC_Potentiometer Prozess gesteuert werden.

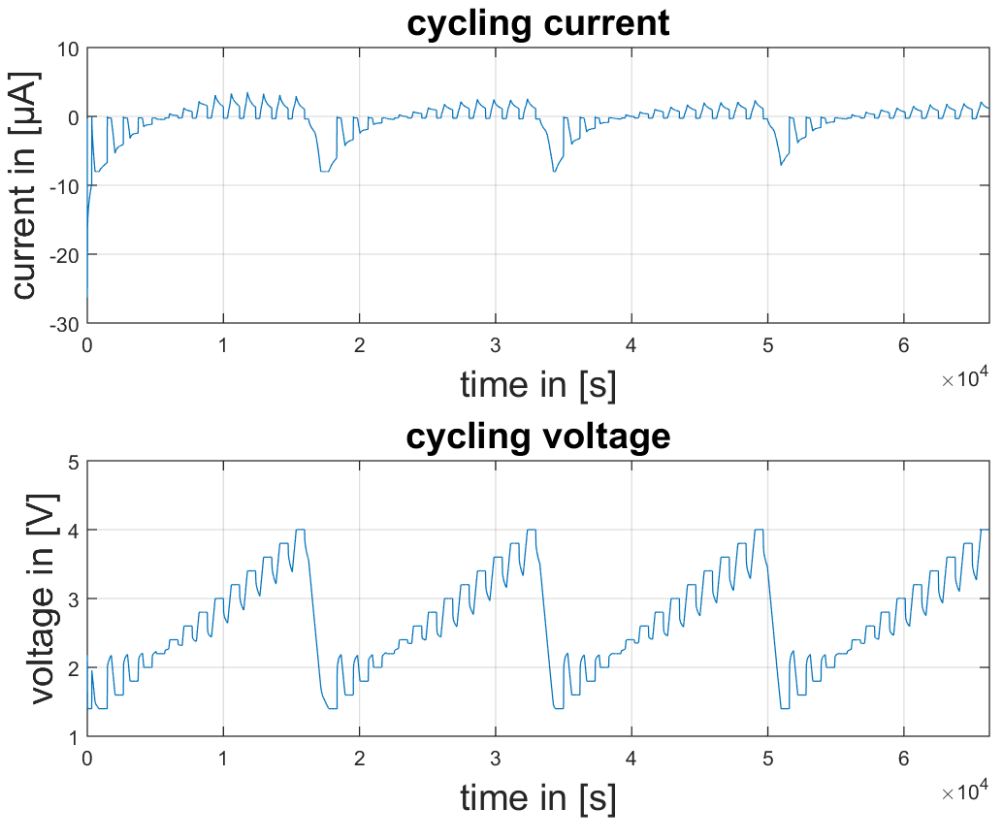

Abbildung 5.6.: Zyklierung einer ITO Zelle mit cCcV-Regelung von 1,4V bis 4V mit Tektronix DMM4020 Messgeräten

Aus dem Spannungsverlauf ist gut zu erkennen, dass die Zelle vorher eine Ruhespannung von ca. 2,1V hatte. Der Strom ist bei dieser Zelle sehr gering. Er fällt nach wenigen μA gleich wieder ab. Der Grund dafür ist, dass eine ITO-Zelle nur eine sehr kleine Kapazität hat. Die Ist-Spannung folgt der Soll-Spannung bei dieser Zelle daher am schnellsten, im Gegensatz zu den beiden anderen Testzellen.

In einer anschließenden Auswertung der Bilddaten via Matlab Skript wurden die RGB-Werte von den Pixeln des Kathodenfensters (der Area of Interest) herausgezogen und die Differenzen zu Pixelfarbwerten einer Referenzfläche berechnet. So sind in folgender Abbildung 5.7 im ersten Plot die unkorrigierten Lichtintensitäten für Rot, Grün und Blau über der Zeit aufgetragen und in dem zweiten Plot ist eine Mittelung dieser Werte aufgeführt. Im dritten Plot ist deren Differenz zueinander enthalten.

Um einen korrelativen Zusammenhang von den optischen Daten zu den elektrischen zu erhalten, ist in der darauf folgenden Abbildung 5.8 noch einmal alles untereinander aufgezeigt.

So sind die elektrischen Kurvenverläufe in den ersten beiden Plots noch einmal ersichtlich. Hinzu kommt im dritten Kurvenverlauf die aufgenommene Ladung, für die der Strom über der Zeit numerisch integriert wurde.

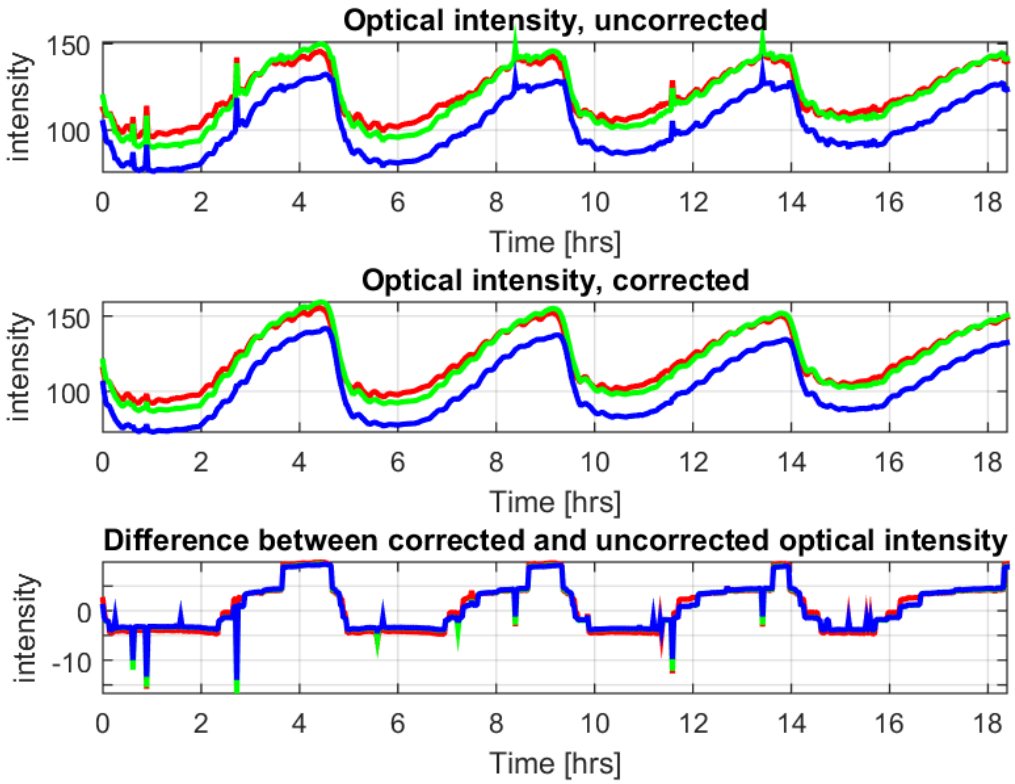

Abbildung 5.7.: Intensität der RGB Anteile im optischen Spektrum des Kathodenfensters der ITO-Zelle

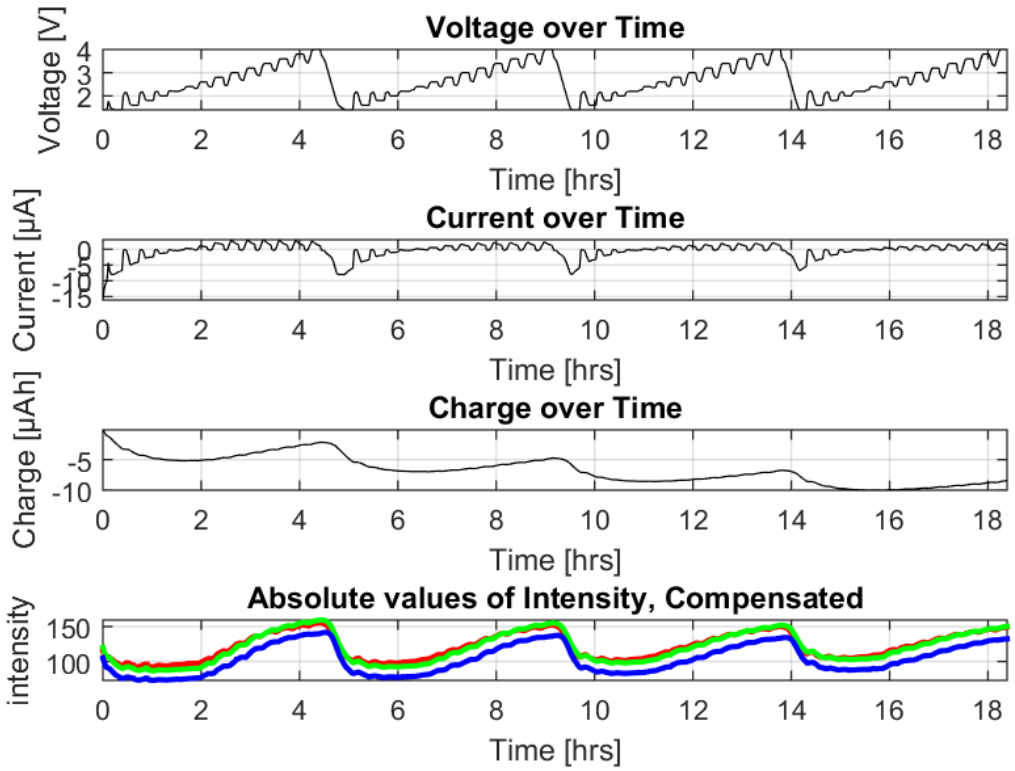

Abbildung 5.8.: Zusammenhang von elektrischen und optischen Verläufen der ITO-Zelle

In der letzten Abbildung 5.9 sind alle Kurvenverläufe und ein Rohdaten-Kamerabild des Kathodenfensters auf einer Seite zusammengefasst.

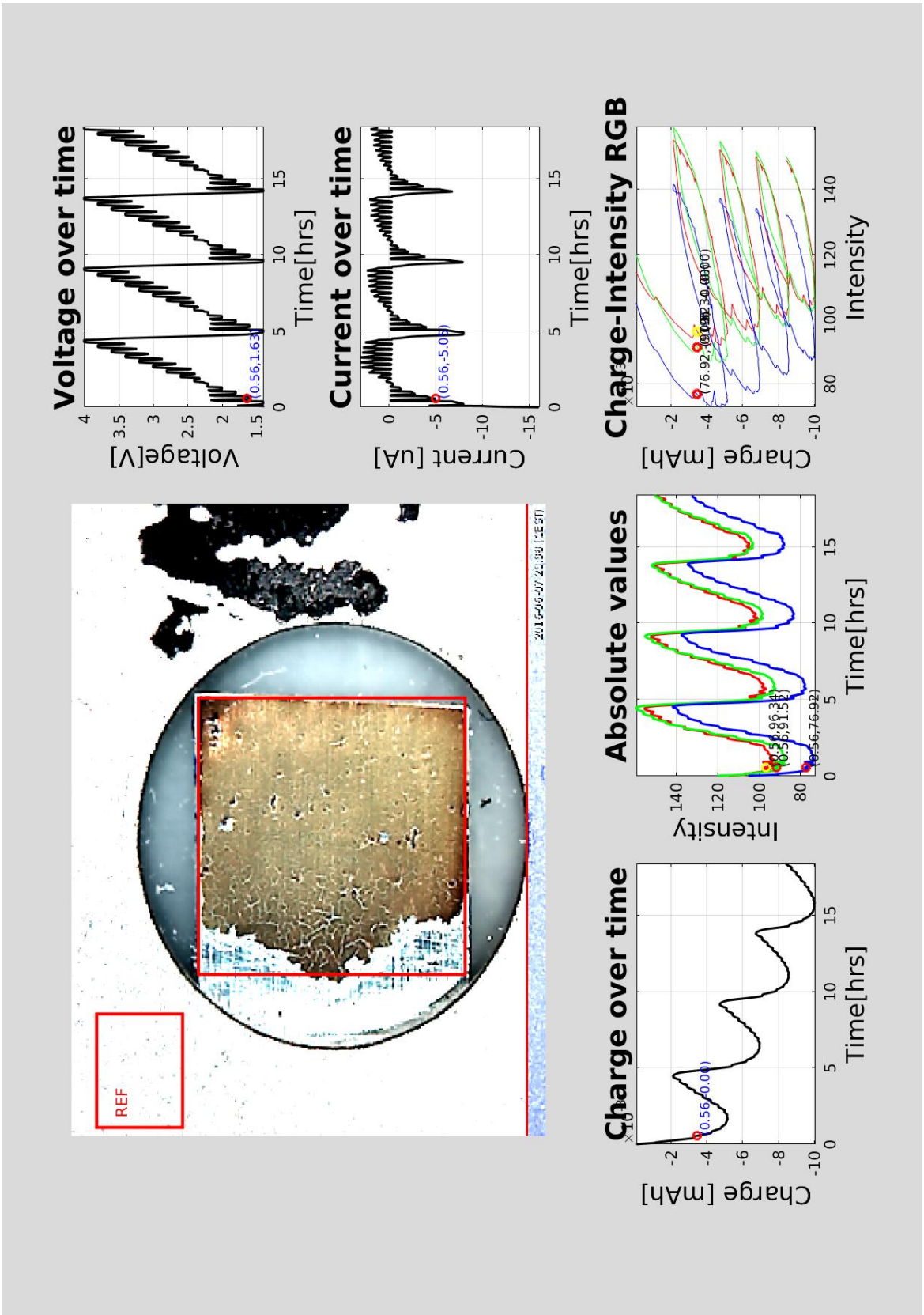

Abbildung 5.9.: Gesamtmessung

5.2. Korrekturmaßnahmen

Besonders bei dem Test der cC-Regelung von Abbildung 5.2 fällt auf, dass bei kleinen Strömen unter $|10\mu A|$ ein Ausreißer von ca. -6,7μA nach Abschluss der Erholungsphase auftritt (orange Markierung in Abbildung 5.10). Bei höheren Strömen fällt diese Ungenauigkeit nicht gravierend ins Gewicht und bei negativen Strömen nur, wenn diese unterhalb -6,7μA sind. Die Ursache dafür ist eine nicht ideale Übertragungskennlinie des DAC und dessen Verstärkers, die durch Offset-, Gain- und Linearitäts-Fehler der Bauteile verursacht wird. Das führt dazu, dass die tatsächliche Ausgangsspannung des Stellgliedes minimal von der gewünschten Spannung abweicht. Diese Abweichungen machen sich immer dann bemerkbar, wenn das Stellglied für den Beginn einer Zyklusperiode durch die Spannungsabgleichs-Routine auf Zellspannung gebracht wird. Da Spannung des Stellgliedes und Zellspannung eben nicht exakt abgeglichen sind, fließt hier nach Schließen des Strom-Relais ein kleiner Ausgleichsstrom, in diesem Fall von der Zelle in die Schaltung hinein.

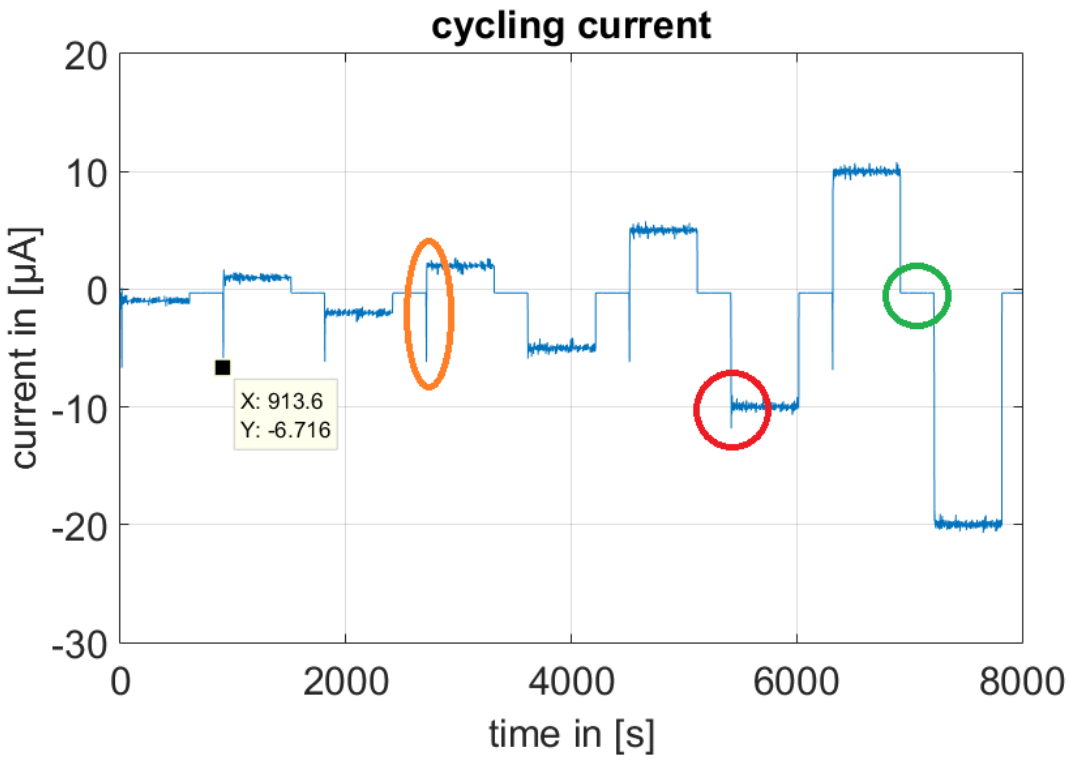

Abbildung 5.10.: Zoom der Zyklierung mit cC-Regelung von $\pm 1\mu A$ bis $\pm 900\mu A$ mit Kopfzelle und Tektronix DMM4020 (aus obiger Abbildung 5.2)

Hier kommt die Anwendung des Programms „LinTest” ins Spiel, mit dem der Wertebereich

des Stellgliedes und der beiden Messglieder für Strom und Spannung abgefahren und aufgezeichnet wird. Für alle drei Kennlinienmessungen wird der Wertebereich von 0 bis Fullscale in 0,05V Schritten hochgestuft. Anstatt einer Batteriezelle, wird ein Testwiderstand von 5kΩ angeschlossen. Ein Anschlussplan des Messaufbaus ist in Kapitel 4.1.13, Abildung 4.2 zu entnehmen.

Anmerkung: Dieser Korrekturvorgang muss für jede Messplatine einzeln durchgeführt werden, da die Ungenauigkeiten der DAC individuell streuen.

5.2.1. DAC Korrektur

Nach Erhalt der Daten des „LinTest" Programms und deren Matlab Plot (mit Anhang: plotData.m), kann in Abbildung 5.11 die Übertragungskennlinie des Stellgliedes entnommen werden. Nur bei Vergrößerung sind hier Offset- und Gain-Error Abweichungen zur idealen Kennlinie zu erkennen. Diese sind in Abbildung 5.12 als absolute und relative Abweichungen in zwei Graphen geplottet.

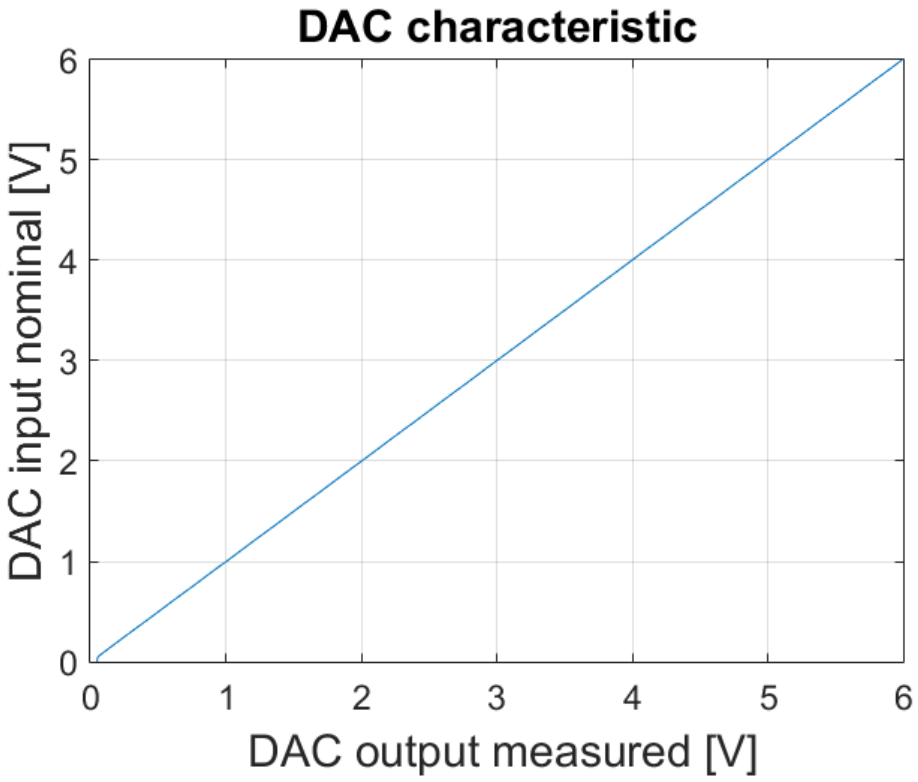

Abbildung 5.11.: Kennlinie des Stellgliedes mit 3V DAC-Referenzspannung und einer Verstärkung von A=2.

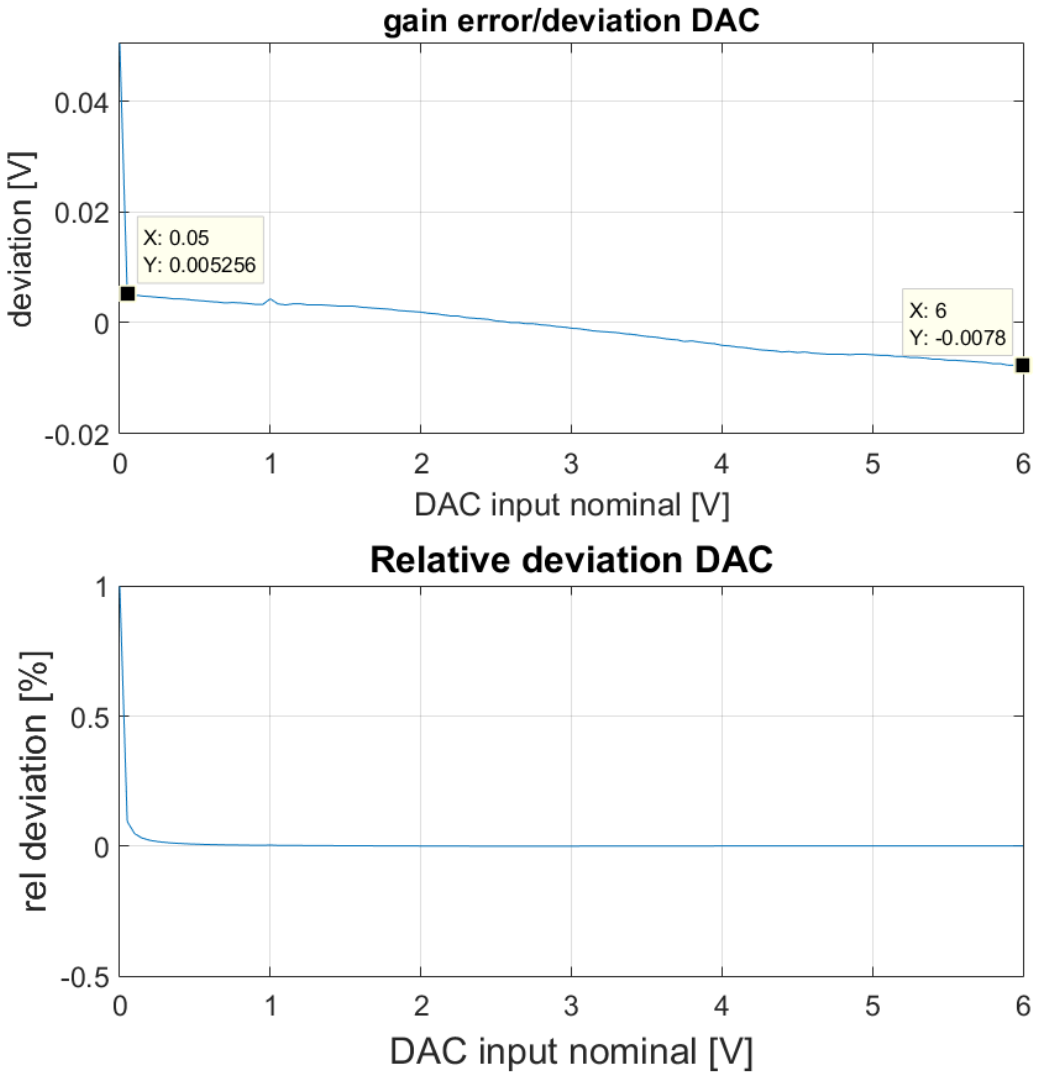

Abbildung 5.12.: Abweichung des Stellgliedes durch Gain-, Offset Fehler und Linearitätsfehler (INL und DNL)

Die DAC-Korrektur wird vorgenommen, in dem die Abweichung aus dem ersten Plott von Abbildung 5.12 zur Vereinfachung linearisiert wird. Somit werden nur der Gain- und Offset-Fehler behoben, was ausreichend ist. Beschrieben wird dies mit der Geradengleichung:

$$f(x) = mx + b \tag{5.1}$$

Der Offset $b \approx 5,256mV$ kann abgelesen und die Steigung m über den Start- und Endpunkt ermittelt werden.

$$m = \frac{\triangle Y}{\triangle X} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-7,8mV - 5,25mV}{6V - 50mV} = -0,00219 \tag{5.2}$$

Die Geradengleichung wird anschließend in die Software eingepflegt. Dies geschieht in der Funktion „BCHW_parseToDACvalue(double voltage)" des Headerfiles „BC_HW_Lib/BC_HW_DAC.h". Diese wandelt den Sollwert von Double in einen dem DAC entsprechenden Integer Datentyp um. An dieser Stelle wird nun noch im Double Bereich die Gain-Error Geradengleichung von der DAC-Spannung abgezogen.

$$U_{INT} = \frac{U_{Soll} - f(U_{Soll})}{U_{LSB}} = \frac{U_{Soll} - (m * U_{Soll} + b)}{U_{LSB}} \tag{5.3}$$

Das Ergebnis dieser Korrekturmaßnahme ist in folgender Abbildung 5.13 zu sehen. Die in orange markierten Stellen machen deutlich, dass der Fehler sogar etwas überkompensiert wurde, weil nun ein Strompeak von ca. +0,05μA, anstatt wie in Abbildung 5.10 von -6,7μA auftritt. Mit einer Anforderung von mindestens 1μA Genauigkeit ist dieser kleine Peak nun akzeptabel.

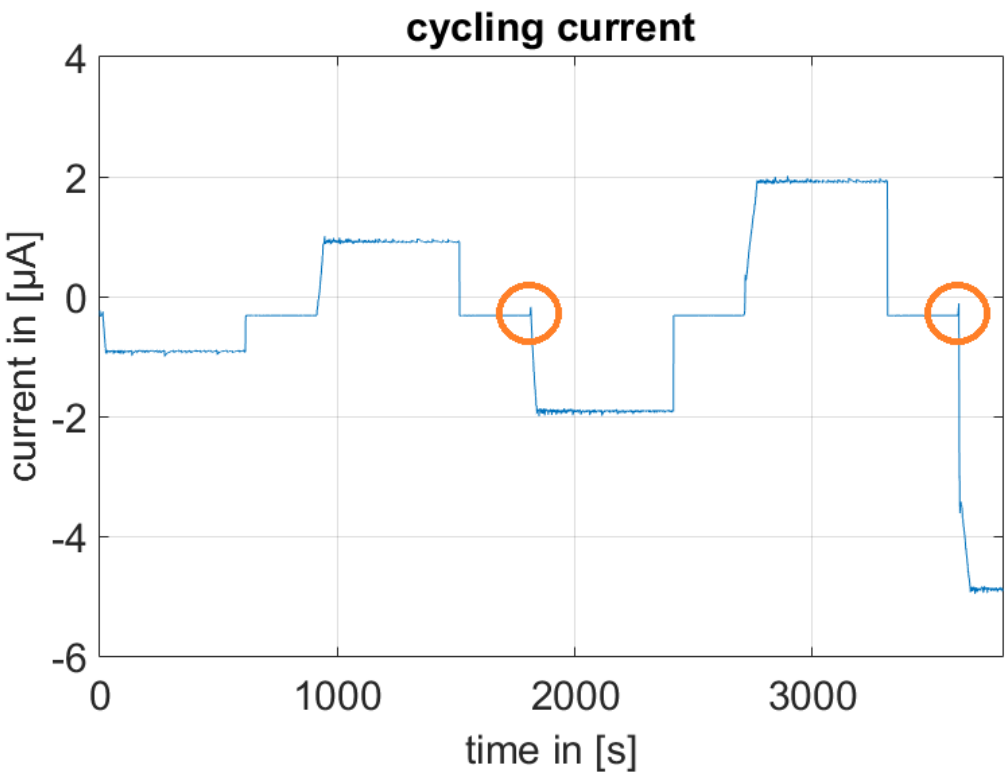

Abbildung 5.13.: Wiederholte cC-Zyklierung selbiger Knopfzelle nach der Korrektur

5.2.2. Strom-ADC Abweichungen

Für die Untersuchung der Abweichungen in Abbildung 5.14 des Strom-ADC wird der zu messende Strom, von einem BC_DMM4020_Reader Prozess für jede DAC Stufe mit gemessen. Für die Kennlinienmessung Abbildung 5.15 führt das Programm zehn weitere Messungen pro Stellschritt durch. Nach der Messung werden die Daten geplottet. In Abbildung 5.14 ist der mit dem externen Messgerät gemessene Strom auf der Abszisse und der absolute und relative Fehler auf der Ordinate aufgetragen. Diese berechnen sich zu

Absoluter Fehler:

$$\delta = |x - \tilde{x}| \tag{5.4}$$

Relativer Fehler:

$$\delta = \left| \frac{x - \tilde{x}}{x} \right| \tag{5.5}$$

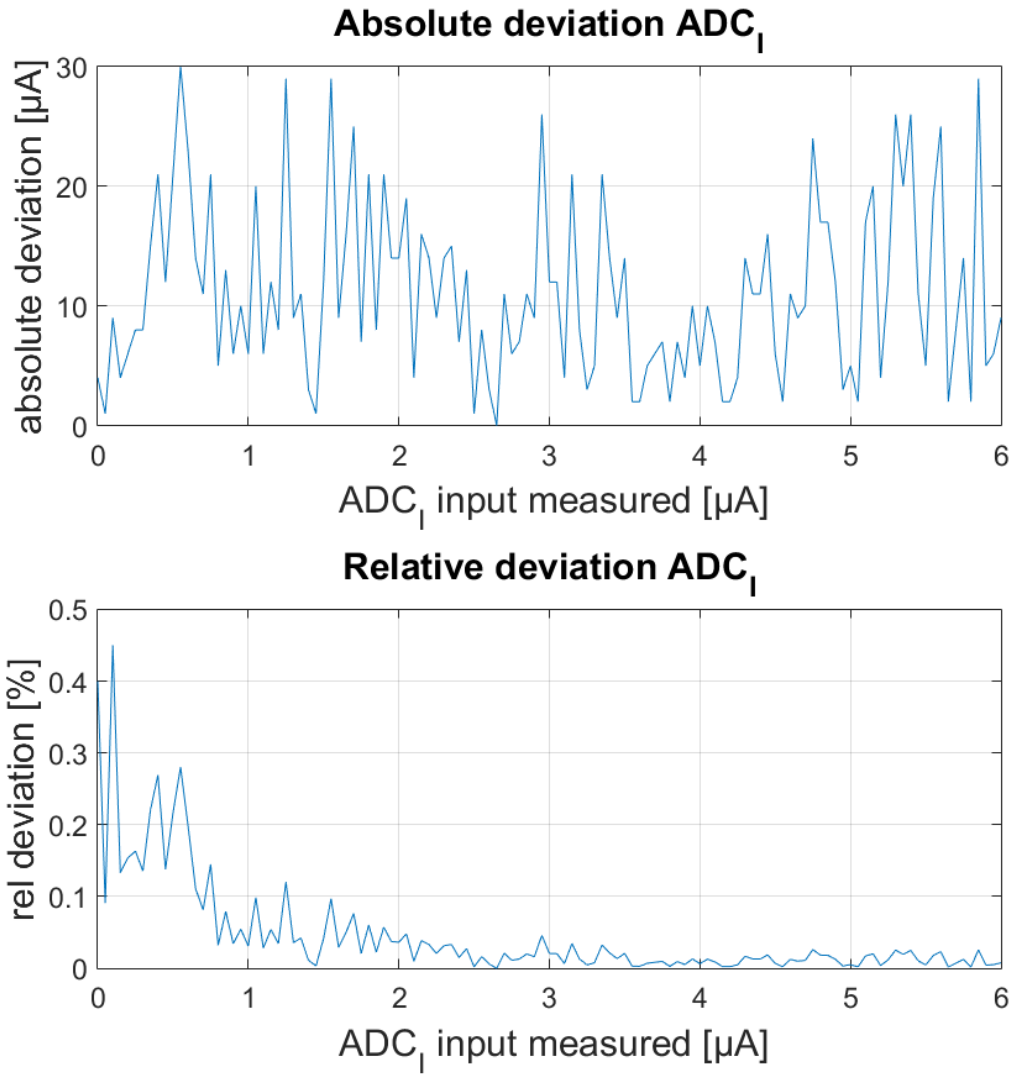

Abbildung 5.14.: Ursprüngliche Abweichung des Onboard-ADC für Strommessung

In Abbildung 5.15 ist auf der Abszisse der gemessene Strom und auf der Ordinate der Digitalwert des ADC aufgetragen. Die roten Messpunkte ergeben ein Störband von ca. 30μA um die ideale ADC Kennlinie.

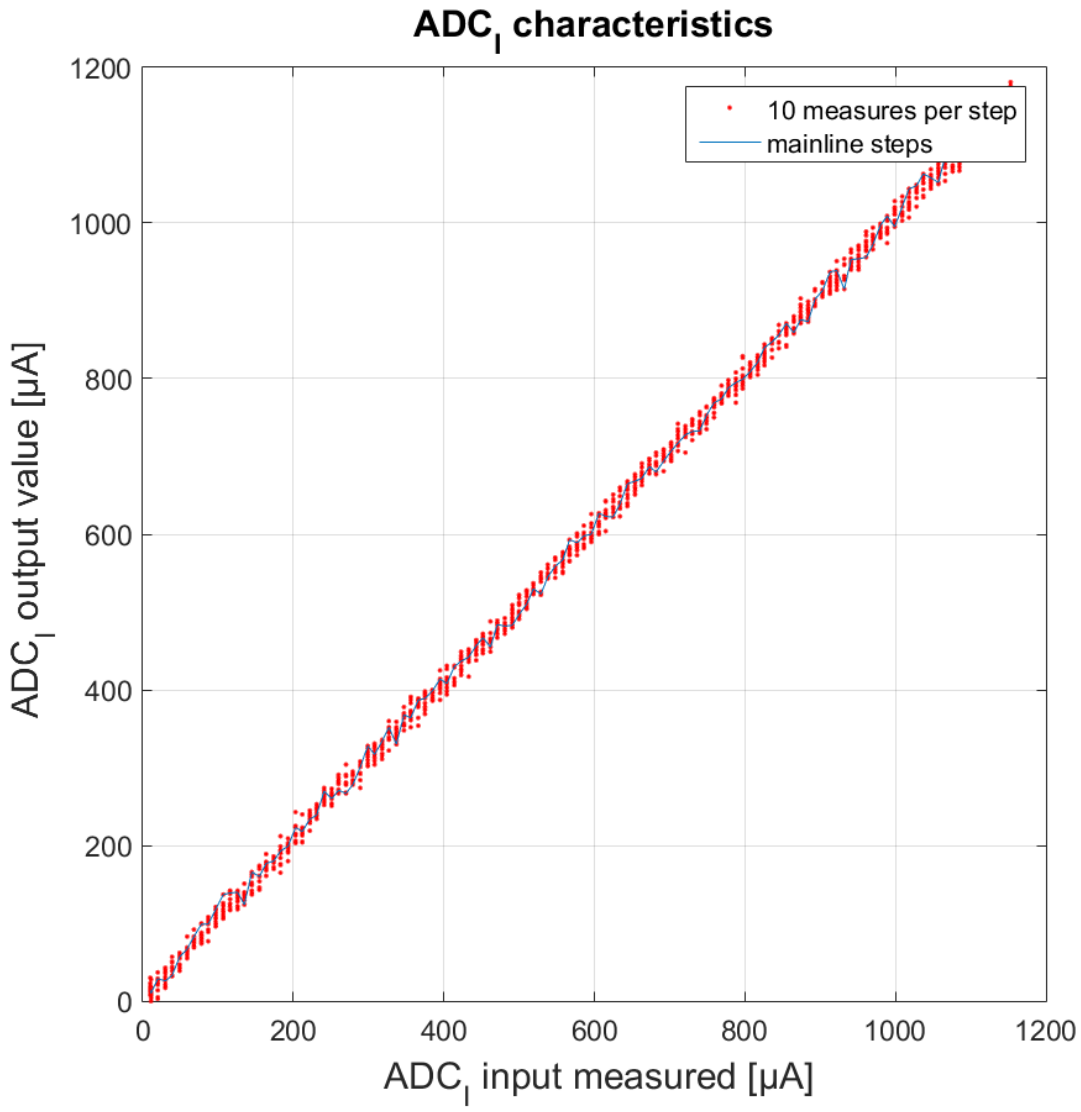

Abbildung 5.15.: Ursprüngliche Kennlinie des Onboard-ADC für Strommessung

Nach Änderung des Eingangsfilters des Strom-ADC verkleinern sich die Abweichungen erheblich. Es ist nun ein 75nF Keramikkondensator vor dem ADC Eingang und hinter dem ursprünglichem RC-Tiefpass parallel an die Pins JP4 auf der Platine geschaltet. Dieser verringert die Grenzfrequenz und dämpft die Amplituden hoher Frequenzen. Nach einer wiederholten Messung der Kennlinie und anschließenden Plots in Abbildung 5.16 und 5.17, ist das Ergebnis deutlich zu sehen.

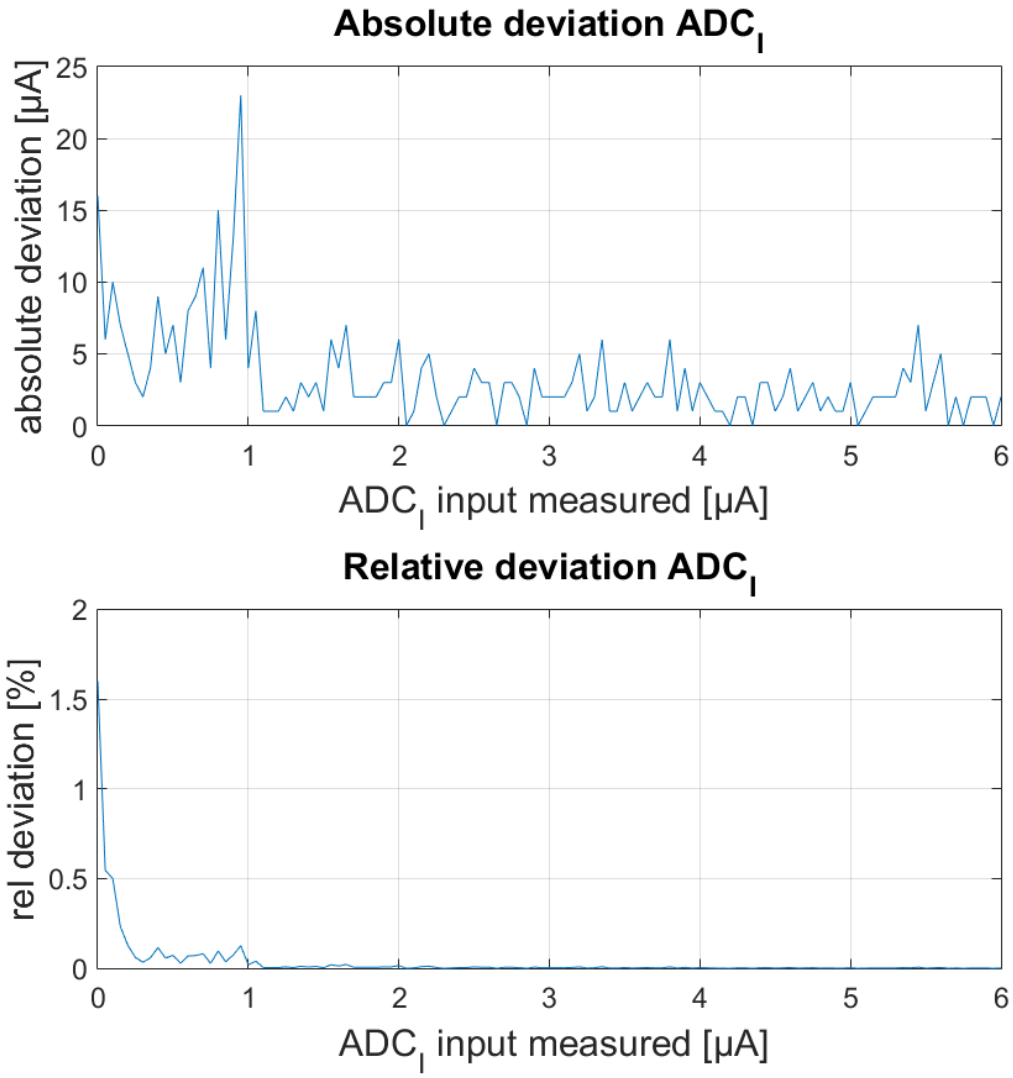

Abbildung 5.16.: Abweichung des Onboard-ADC für Strommessung nach verändertem Eingangsfilter

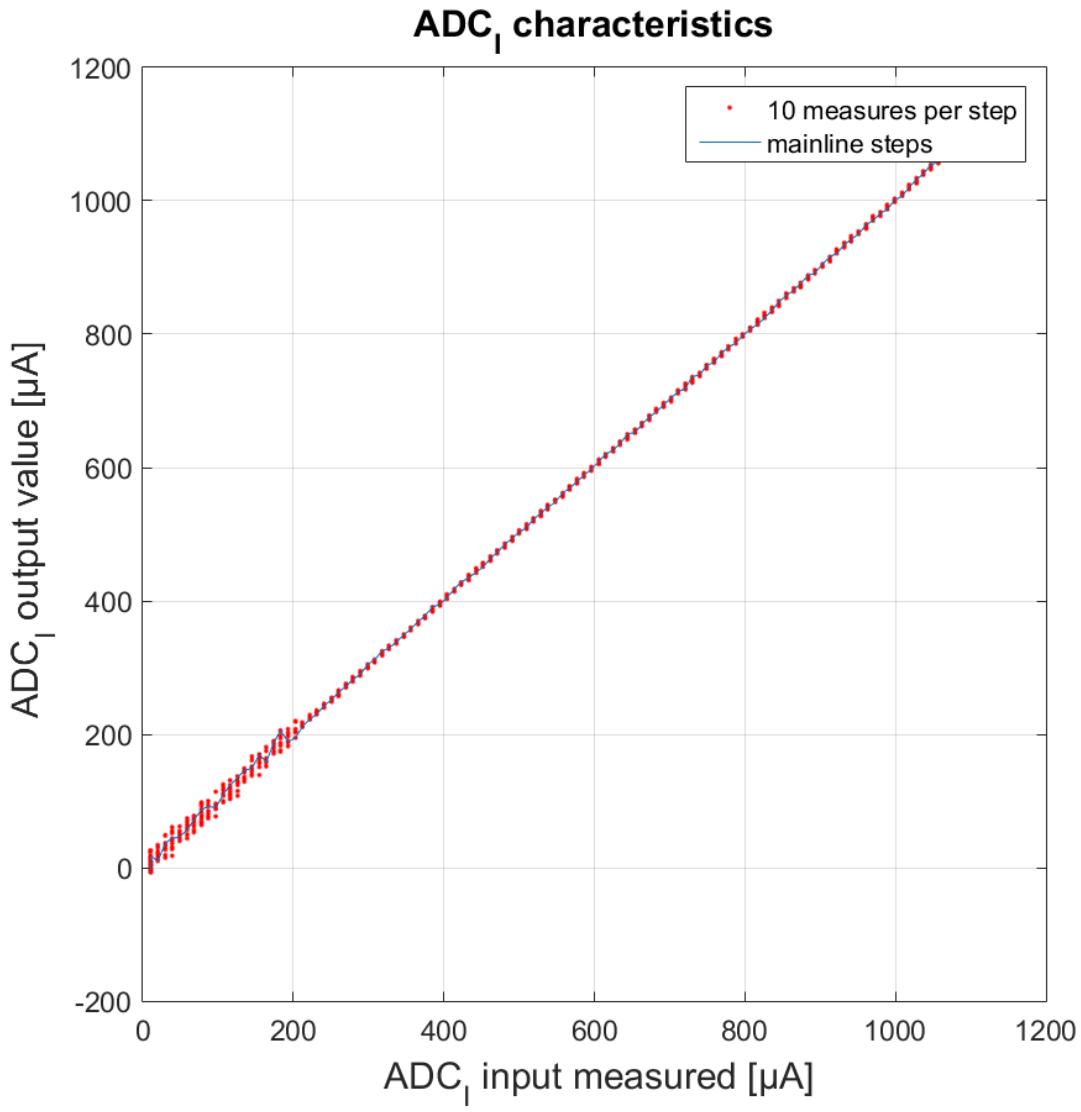

Abbildung 5.17.: Kennlinie des Onboard-ADC für Strommessung nach verändertem Eingangsfilter

5.2.3. Spannungs-ADC Abweichungen

Die Messergebnisse des Spannungs-ADC zeigen in Abbildung 5.18 eine maximale absolute Abweichung von ca. 2,5mV. Das ist eine relative Abweichung von fast 0,05%. In folgender Abbildung 5.19 fallen daher keine großen Abweichungen von der idealen Kennlinie auf. Das ist tolerierbar und erfordert daher keine Verbesserungsmaßnahmen.

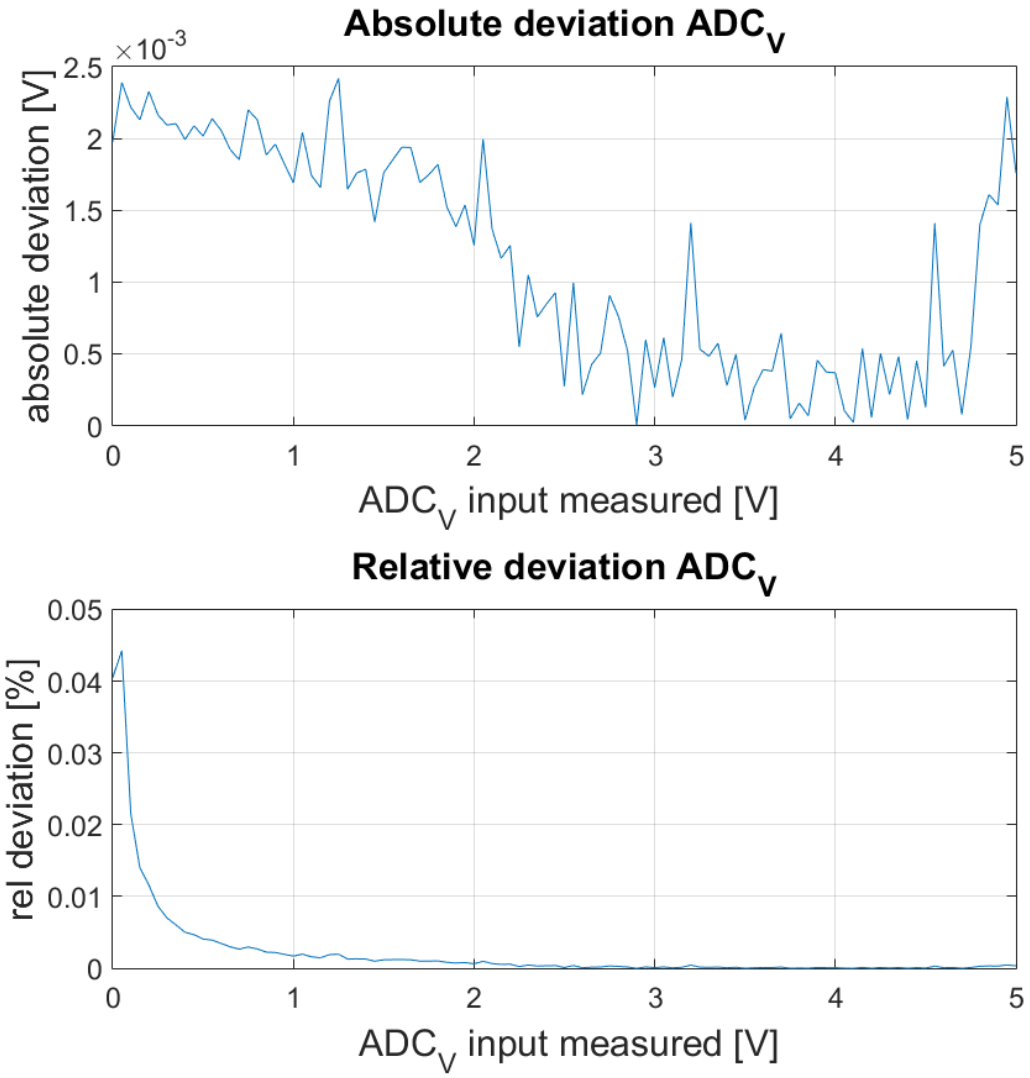

Abbildung 5.18.: Abweichung des Onboard-ADC für Spannungsmessung

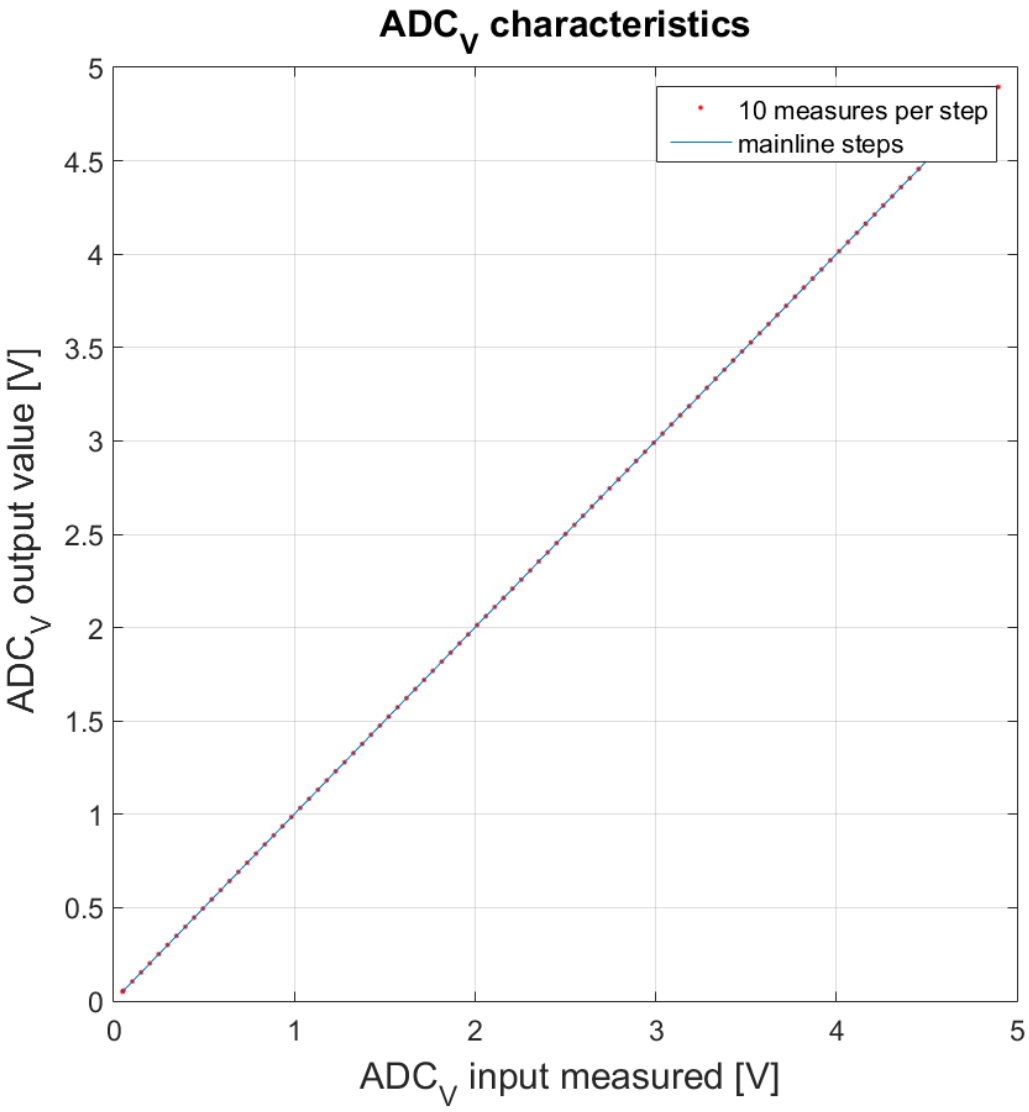

Abbildung 5.19.: Kennlinie des Onboard-ADC für Spannungsmessung

6. Fazit und Ausblick

In diesem Kapitel wird der Verlauf der Arbeit zusammengefasst und die erreichten Ziele mit den Vorgaben verglichen. Im Ausblick werden anschließend aufgetretene Probleme und mögliche Lösungen und Erweiterungen erläutert.

6.1. Zusammenfassung

Aufgabe war es, die Regelsoftware eines bestehenden Zykliersystems für einen Messplatz zur optischen Batterie-Analyse zu entwickeln. Die Software soll auf einer Raspberry PI 2 Plattform operieren, flexibel und vielfältig einsetzbar sein und viele Möglichkeiten des Betriebssystems nutzen. Primär sollte das System eine Zelle mit einem konstanten Strom laden und entladen können. Ausserdem sollte die Software permanent die Zellspannung überwachen und sich an ihren Ladeschlussspannungen orientieren. Eine Fotoaufnahme des Zellen-Elektrodenfensters sollte automatisiert ausgelöst werden. Neben der Stromregelung sollte auch eine Spannungsregelung implementiert und eine Möglichkeit geschaffen werden, die elektrischen Messwerte von externen Messgeräten auszulesen. Bilder und Messdaten sollen auf einer externen USB-Festplatte für anschließende Auswertungen gespeichert werden.

Für die Umsetzung der Aufgabenstellung wurde die modulare BatCycle Software programmiert. Diese besteht aus parallel ausführbaren Prozessen, in denen Teilaufgaben des Zykliersystems modularisiert sind. So gibt es zwei Stellprozesse, um den DAC und die LEDs anzusteuern, drei Messprozesse für Temperatur-, Strom- und Spannungswerte, einen Relais-Prozess, um die drei Relais auf der Messplatine zu schalten und einen Hauptprozess, in dem der Zyklierablauf, die Istwert-Überwachung und Regelung stattfindet. Die Interprozesskommunikation ist über Unix-Sockets aufgebaut. Dabei entspricht der Hauptprozess einem Clienten und alle anderen sind Mess- und Stellserver. Mit Jan Griessbach wurde ein Protokoll entwickelt, um einen Kommunikationsstandard und fehlerfreies Senden und Empfangen zu gewährleisten. Für die Möglichkeit externe Messgeräte auszulesen, hat Herr Griessbach das aus seiner Arbeit entstandene DMM4020_Reader_v2 Programm in eine Socket basierte Version für das BC_Measurement Protokoll umprogrammiert.

Während und nach der Softwareentwicklung jedes Moduls wurden die Kommunikationsschnittstellen auf Einhaltung des entsprechenden Protokolls getestet. Nach Fertigstellung

des letzten Prozesses, dem MainController, wurde das Gesamtsystem mit einer Reihe an Testläufen und Zellzyklierungen erprobt und getestet.

6.2. Bewertung

Die Anforderungen an die Stromregelung des Zykliersystems waren ähnlich wie bei vorheriger Arbeit von Herrn Geist [14]. So sollte das System einen konstanten Zellstrom von einstellbaren 1μA bis 1mA mit einer Genauigkeit von 1μA regeln können. Bei eingeschwungener Regelung wurde das Ziel mit den Onboard Messgliedern aus Amperemeter und Strom-ADC mit 2μA Genauigkeit fast erreicht. Mit dem DMM4020_Reader und einem externen Tektronix Messgerät werden eine Genauigkeit von ca. 1μA erreicht.

Die drei Potentiometer zur Steuerung der RGB-LED in der Kamera werden mit dem Programm BC_Potentiometer_SP angesprochen. Allerdings ist zu spät aufgefallen, dass mit den Potentiometern und einem Spannungsteiler eine lineare LED-Treiber Eingangsspannung nicht erzeugt werden kann. Daher wurde dieser Weg nicht weiter gegangen. Das Potentiometer Programm ist zwar fertig, wurde jedoch nie getestet.

Die Temperaturdatenerfassung funktioniert nur im Zusammenhang mit den externen Tektronix Messgeräten für Strom- und Spannungsmessung zuverlässig. Wenn die beiden Onboard-ADC und der Temperatursensor über SPI ausgelesen werden sollen, dann wird die gesamte SPI Kommunikation auf der MISO Leitung gestört und eine Zyklierung ist gar nicht mehr möglich. Das liegt daran, dass der LM74 Temperatursensor permanent wandelt und einen Wert auf den Bus schickt. Dies stört die Kommunikation der ADC. Der Temperatursensor hat zwar eine Möglichkeit, über einen Befehl aus einem Continuous Mode in einen Shutdown Mode versetzt zu werden und umgekehrt, jedoch vereint er die MISO und MOSI Leitung zu einer SI/O Leitung. So muss im Halbduplex-Betrieb dieser 16 Bit Shutdown Befehl in sein Eingangsregister geschrieben werden. Auf der Messplatine sind jedoch MISO und MOSI Leitung bewusst voneinander getrennt.

Die Kommunikations Protokolle BCMP, BCAP und BCRP der modularen BatCycle Software wurden nach der Entwicklung auf jegliche Sende-und Empfangsmöglichkeiten mit Hilfe der Frame Debugger getestet und erprobt. Alles funktioniert der BatCycle Protokoll Spezifikation entsprechend ordnungsgemäß.

Für die Regelalgorithmen von Strom und Spannung wurde ein konservativer Weg eingeschlagen. Priorität war es, dass der Digitalregler zuverlässig funktioniert, sonst würde keine Zyklierung möglich sein. Daher wurde der bewährte Reglertyp wie vom bestehenden Zykliersystem gewählt. Dieser ist ein Zweipunktregler, der auch mit einer nichtäquidistanten Abtastung stabil regeln kann. Ein Nachteil des Reglertyps ist, dass es Oszillation um den Sollwert gibt.

6.3. Ausblick

Im Nachhinein ist der Temperatursensor falsch ausgesucht. Es wäre daher sinnvoller, einen anderen Temperatursensor zu verwenden. Über einen One-Wire Sensor wäre es z.B. komfortabler, da dieser nur zwei Leitungen benötigt: eine Masseleitung und eine Datenleitung, die gleichzeitig auch als Spannungsversorgung fungieren kann.

Jede neue Kennlinien-Korrektur erfordert eine Neukompilierung des VoltageController Programms. Für Folgeversionen könnte dieses benutzerfreundlicher gestaltet werden, indem die Korrekturparameter in ein Initfile gespeichert und vom VoltageController Prozess eingelesen werden. So gibt es zu jeder Messplatine nur Unterschiede in den Initfiles und nicht in den kompilierten Programmen.

Die digitale Regelung der cC-, cV- und cCcV-Funktion funktioniert mit einem Zweipunktregler, der mit quasi äquidistanten Intervallen arbeitet. Diese Regelschrittintervalle werden über eine Nanosleep Funktion realisiert. Diese greift in den Scheduler ein und legt den Prozess für mindestens 100ms schlafen. Je nach anstehenden Aufgaben des Schedulers, kann der Prozess auch etwas später aktiv werden. Diese Lösung verursacht also ein geringes Jittern bezüglich der Obergrenze der Abtastperiode.
Der Zweipunktregler könnte noch durch einen digitalen PI Regler mit Rekursionsformel ausgetauscht werden. Damit würde eine genaue und schnelle Regelung möglich sein. Da jedoch öfter mal kleine Hänger des Betriebssystems auftreten, kann dies die Äquidistanz des PI-Reglers verschlechtern. Die Äquidistanz lässt sich aber verbessern, indem die bestehenden Timer mittels Sleep Funktionen mit einem Software Interrupt, bzw. ein Timer Signal des Betriebssystems ausgetauscht werden. Da nach einer Beobachtung der Messserver die Timer Signale trotz hoher Systemauslastung relativ zuverlässig auf 100ms genau ausgelöst werden, ist es ein Versuch Wert, diese Idee zu implementieren. Die Änderungen betreffen dann nur das MainController Programm.

Dies könnte noch durch einen digitalen PI Regler mit Rekursionsformel und genauerer Äquidistanz verbessert werden. Damit würde eine genaue und schnelle Regelung möglich sein. Die Äquidistanz lässt sich über einen Software Interrupt, bzw. ein Timer Signal des Betriebssystems verbessern. Im Vergleich zu der Lösung sleep Funktion

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] *LM7910CT Datasheet*. Fairchild Semiconductor, 2011. – URL https://www.fairchildsemi.com/datasheets/LM/LM7910.pdf.. – Zugriffsdatum: 16.07.2015

[2] *Cortex-A7 MPCore Technical Reference Manual*. ARM, 2012

[3] *AD5270 Datasheet*. Analog Devices, 2013. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD5270_5271.pdf.. – Zugriffsdatum: 16.08.2015

[4] *AD5680 Datasheet*. Analog Devices, 2014. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD5680.pdf.. – Zugriffsdatum: 14.07.2015

[5] *LM2940T-10.0 Datasheet*. Texas Instruments, 2014. – URL http://www.ti.com/lit/ds/symlink/lm2940-n.pdf.. – Zugriffsdatum: 16.07.2015

[6] *OPA743 Datasheet*. Texas Instruments, 2014. – URL http://www.ti.com/lit/ds/symlink/opa743.pdf.. – Zugriffsdatum: 15.07.2015

[7] *TLV217 Datasheet*. Texas Instruments, 2014

[8] *AD7691 Datasheet*. Analog Devices, 2015. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD7691.pdf.. – Zugriffsdatum: 14.07.2015

[9] *AD8422 Datasheet*. Analog Devices, 2015. – URL http://www.analog.com/media/en/technical-documentation/data-sheets/AD8422.pdf.. – Zugriffsdatum: 14.07.2015

[10] *LM74 Datasheet*. Texas Instruments, 2015. – URL http://www.ti.com/lit/ds/symlink/lm74.pdf.. – Zugriffsdatum: 16.08.2015

[11] *TPS799 Datasheet*. Texas Instruments, 2015. – URL http://www.ti.com/lit/ds/symlink/tps799.pdf.. – Zugriffsdatum: 17.07.2015

[12] 2014, Energieverbrauch: *UMWELTBUNDESAMT*. 09.05.14. – URL https://www.umweltbundesamt.de/daten/energiebereitstellung-verbrauch/energieverbrauch-nach-energietraegern-sektoren. – Zugriffsdatum: 26.03.2016

[13] BUNDESREGIERUNG, DIE: *Nationaler Entwicklungsplan Elektromobilität*. 01.08.2009. – URL https://www.bmbf.de/files/nationaler_entwicklungsplan_elektromobilitaet.pdf. – Zugriffsdatum: 08.04.2016

[14] GEIST, Torsten: *Entwicklung eines embedded Einplatinensteuersystems für optische Batterieuntersuchung*. HAW, 2015

[15] GRIESSBACH, Jan: *Messaufbau mit Steuer- und Analysesoftware für die optische Zustandsbeobachtung von Lithiumbatterien*. HAW, 2014

[16] HEROLD, Helmut: *Linux/ Unix Systemprogrammierung*. Addison Wesley Verlag, 1999

[17] ITWISSEN.INFO: *Monolithische Software-Architektur*. – URL http://www.itwissen.info/definition/lexikon/Monolithische-Software-Architektur.html. – Zugriffsdatum: 14.07.2015

[18] KRAFTFAHRTBUNDESAMT: *Pressemittteilung Nr.8/201*. 01.01.16. – URL http://www.kba.de/DE/Presse/Pressemitteilungen/2016/Fahrzeugbestand/pm8_fz_bestand_pm_komplett.html?nn=716842. – Zugriffsdatum: 30.03.2016

[19] LINDNER, Marc A.: *Libconfig: A Library For Processing Structured Configuration Files*. GNU LESSER GENERAL PUBLIC LICENSE, 2015

[20] PAGE, Linux man: *Linux Sockets*. 2001. – URL http://linux.die.net/man/7/socket. – Zugriffsdatum: 21.5.2016

[21] PALLIYAGURUGE, Don Mithila M.: *Image processing for investigation of effects in Lithium battery electrodes*. HAW, 2015

[22] RIEMSCHNEIDER, Prof. Dr.-Ing. Karl-Ragmar: *Drahtlose Sensoren in den Zellen von Fahrzeugbatterien*. 2011. – URL https://www.haw-hamburg.de/fileadmin/user_upload/TI-IE/Daten/Docs/ESZ-ASP/IWKM21_Batsen.pdf. – Zugriffsdatum: 16.06.2016

[23] SCHIEPEL, Philip: *Messzelle für optoelektronische Analyse von Lithium-Ionen-Batterien*. HAW, 2014

[24] SCHNEIDER, Prof. J.: *Skript Betriebssysteme*. HAW, 2012

[25] UMWELTBUNDESAMT: *Treibhausgase*. 15.01.2015. – URL https://www.umweltbundesamt.de/themen/klima-energie/klimaschutz-energiepolitik-in-deutschland/treibhausgas-emissionen/die-treibhausgase. – Zugriffsdatum: 4.04.2016

[26] UMWELTBUNDESAMT: *Grenzwerte für Schadstoffemissionen von PKW*. 24.06.2013. – URL https://www.umweltbundesamt.de/themen/verkehr-laerm/emissionsstandards/pkw-leichte-nutzfahrzeuge. – Zugriffsdatum: 30.03.2016

[27] WIKIPEDIA: *Energieverbrauch*. 09.05.14. – URL https://de.wikipedia.org/wiki/Energieverbrauch. – Zugriffsdatum: 26.03.2016

A. Config Dateien

A.1. bc_init.cfg

```
/*****************************************************************************************
 * File:         bc_init.cfg
 * Version:      0.2
 * Date:         2015-12-26
 * Author:       Nico Rieckmann
 *
 * Discription: is a configuration file to parameterize the inner settings of battery cycling
 *              software. It will be read by MainController process.
 * Info:        Syntax Symbols for comments are '//', '#' and '/ *...* /'. Trailing simicolon
 *              is optional and whitespace is not significant.
 * References:  This configuration file uses the C lib "libconfig-1.5" from Marc A. Lindner.
 *
 * Changes:     0.2: NR added current_shunt parameter
                0.1: NR first version.
 *
 *****************************************************************************************/

version = 0.2;


inputgroup:     # Input interface
{
    sockets:
    {
        sockCurrent = "/tmp/Isock.uds";         //Pfad des Unix-Socket für Amperemeter
        sockVoltage = "/tmp/Vsock.uds";         //Pfad des Unix-Socket für Voltmeter
        sockTemp    = "/tmp/Tsock.uds";         //Pfad des Unix-Socket für Temperaturwerte
        sockDAC     = "/tmp/Dsock.uds";         //Pfad des Unix-Socket für VoltageController
        sockPoti    = "/tmp/Psock.uds";         //Pfad des Unix-Socket für Potentiometer
        sockRelay   = "/tmp/Rsock.uds";         //Pfad des Unix-Socket für RelayController
    };
};

BCHWgroup:      # parameters for batcycle hardware
{
    current_shunt = 100.0;      //Messwiderstand auf Platine im Ladestrompfad in [Ohm]
};
```

A.2. bc_param.cfg

```
/*****************************************************************************************
 * File:         bc_param.cfg
 * Version:      0.3
 * Date:         2016-04-29
 * Author:       Nico Rieckmann
 *
 * Discription: This is a configuration file to parameterize the inner settings of battery
 *              cycling software. It will be read by MainController process.
 * Info:        Syntax Symbols for comments are '//', '#' and '/ *...* /'. Trailing simicolon
 *              is optional and whitespace is not significant.
 * References:  This configuration file uses the C lib "libconfig-1.5" from Marc A. Lindner.
 *
 * Changes: 0.3: NR tidy up all and added some more
 *          0.2: NR added impedance and mode_time parameters
 *          0.1: NR first version.
```

```
16   *
17   ********************************************************************************************/
18
19   version = 0.3;
20
21
22   batterygroup:
23   {
24       type        = "Generic battery";        //Test Batterie
25       impedance   = 320.0;                    //Batterie Innenwiderstand in [Ohm]
26   }
27
28
29   limitgroup:         # Grenzwerte
30   {
31       max_voltage             = 4.00;         //obere Spannungsgrenze in [V]
32       min_voltage             = 2.00;         //untere Spannungsgrenze in [V]
33       max_temp                = 60.0;         //obere Temperaturgrenze in °[C]
34       min_temp                = -10.0;        //untere Temperaturgrenze in °[C]
35       max_current_charge      = 1000.0;       //maximaler Ladestrom in [µA]
36       max_current_discharge   = 1000.0;       //maximaler Entladestrom in [µA]
37   }
38
39
40   cyclegroup:         # Zyklierungs Parameter
41   {
42       //"mode":       Three dis-/charging modes are available: cCcV (const. Current const. Voltage),
43       //             cC (const. Current) or cV (const. Volatage).
44       //"cycles":     Number of battery cycling iterations: int value or "0" for an infinite loop.
45       //"mode_time":  Countdown to end the mode: int value or "0" for no timing limit.
46       //             The activation of mode_time is depending on modes:
47       //                 1. "cV":    mode_time begins after currentfall nominal current.
48       //                 2. "cC":    mode_time begins at start of this mode.
49       //                 3. "cCcV":  mode_time is only for latter cV phase possible.
50
51       startphase:     # Startephase soll Zelle auf folgende Werte bringen
52       {
53           mode                = "cC";         //Ladeverfahren
54           mode_time           = 1;            //Zeitliche Modus Begrenzung in [s]
55           threshold_voltage   = 3.16;         //Schwellspannung, Ladeschlussspannung in [V]
56           nominal_current     = 1.0;          //Sollstrom in [µA]
57           convalescence       = 10;           //Erholungszeit in [s]
58           led_R               = 10.68;        //Led-Rot-Wert in [%]
59           led_G               = 90.50;        //Led-Gelb-Wert in [%]
60           led_B               = 100.0;        //Led-Blau-Wert in [%]
61       }
62
63
64       cyclephase:     # Zyklierungsphase
65       {
66           cyclelist = (
67           {                           //Entladung mit konst. Strom
68               mode                = "cCcV";       //konst Strom Entladung
69               mode_time           = 600;          //Zeitliche Modus Begrenzung in [s]
70               threshold_voltage   = 2.9;          //Schwellspannung, Entladeschlussspannung in [V]
71               nominal_current     = -500.0;       //Sollstrom in [µA]
72               convalescence       = 300;          //Erholungszeit in [s]
73               led_R               = 0.0;          //Led-Rot-Wert in [%]
74               led_G               = 0.0;          //Led-Gelb-Wert in [%]
75               led_B               = 0.0;          //Led-Blau-Wert in [%]
76           },
77           {                           //Ladung mit konst. Strom und konst. Spannung
78               mode                = "cCcV";       //Ladeverfahren
79               mode_time           = 600;          //Zeitliche Modus Begrenzung in [s]
80               threshold_voltage   = 3.60;         //Schwellspannung, Ladeschlussspannung in [V]
81               nominal_current     = 500.0;        //Sollstrom in [µA]
82               convalescence       = 300;          //Erholungszeit in [s]
83               led_R               = 0.0;          //Led-Rot-Wert in [%]
84               led_G               = 0.0;          //Led-Gelb-Wert in [%]
85               led_B               = 0.0;          //Led-Blau-Wert in [%]
86           }
87           );
88
89           cycles  = 2;                //Anzahl an Wiederholungen der Cyclelist
90       }
91   };
```

B. Matlab Skripte

B.1. BatCycle Messdaten Auswertung

B.1.1. plottDataFiles.m

```
clear all;
close all;
strXlabel = 'time in [s]';
factor = 1;

%open multiple files with GUI is enabled
[FileName,PathName] = uigetfile('*.*','Select a current trendfile to plot', 'MultiSelect', 'on');

if iscellstr(FileName) %when multiple files selected
    len = size(FileName);
    len = len(2);
    for i = 1 : len
        filePath = sprintf('%s', PathName, char(FileName(i)));
        factor = 1;
        if strfind(char(FileName(i)), 'I')      %get units and names from files
            strTitle = 'cycling current';
            strYlabel = 'current in [µA]';
            factor = 1e6;
        elseif strfind(char(FileName(i)), 'V')
            strTitle = 'cycling voltage';
            strYlabel = 'voltage in [V]';
        elseif strfind(char(FileName(i)), 'T')
            strTitle = 'measured temperature';
            strYlabel = 'temperature in °[C]';
        end
        [t, value] = readFile(filePath);       %fetch vektors

        for k=2 : length(t)          %normalize timestamps
            t(k) = t(k) - t(1);
        end
        t(1) = 0;

        %Plot in Matlab
        figure(1);
        subplot(len,1,i); plot(t, value*factor);
        xlim([t(1) t(length(t))]); grid on;
        title(strTitle, 'Fontsize', 18);
        xlabel(strXlabel, 'Fontsize', 18);
        ylabel(strYlabel, 'Fontsize', 18);
    end
else                    %when a single file selected
    filePath = sprintf('%s', PathName, char(FileName));
    if strfind(char(FileName), 'I')     %get units and names from files
        strTitle = 'cycling current';
        strYlabel = 'current in [µA]';
        factor = 1e6;
    elseif strfind(char(FileName), 'V')
        strTitle = 'cycling voltage';
        strYlabel = 'voltage in [V]';
    elseif strfind(char(FileName), 'T')
        strTitle = 'measured temperature';
        strYlabel = 'temperature in °[C]';
    end
    [t, value] = readFile(filePath);       %fetch vektors
    for k=2 : length(t)          %normalize timestamps
        t(k) = t(k) - t(1);
    end
    t(1) = 0;

```

```
60      %Plot in Matlab
61      figure(1);
62      plot(t, value*factor); set(gca,'Fontsize',18);
63      xlim([0 t(length(t))]); grid on;
64      title(strTitle, 'Fontsize', 18);
65      xlabel(strXlabel, 'Fontsize', 18);
66      ylabel(strYlabel, 'Fontsize', 18);
67  end
68
69
70  % format long e
71  % current(1);
72  %
73  % x=0:1:100;
```

B.1.2. readFile.m

```
1   function [ timestamp, value ] = readFile(fname)
2   %readFile function reads Datarows from BatCycle File
3   %   Detailed explanation goes here
4
5       fileID = fopen(fname, 'r');
6
7       i = 1;
8       tline = fgets(fileID);
9       while ischar(tline)
10          if tline(1) == '#'
11              disp(tline);
12          else
13              timestamp(i) = sscanf(tline, '%ff');value(i) = sscanf(tline, 'i = i + 1;endtline = fgets(fileID);endfclose(fileID);end
```

B.2. LinTest Messdaten Auswertung

B.2.1. plottDataFiles.m

```
1   clear all;
2   close all;
3
4   %initialization
5   startRow = 3;
6   endRow = 0;
7   startCol = 1;
8   endCol = 7;
9   rangeRow = 11;
10  COL_ADC_DMM = 7;
11  COL_ADC_VAL = 6;
12  COL_ADC_DEC = 5;
13  COL_TIMESTAMP = 4;
14  COL_DAC_DMM = 3;
15  COL_DAC_DEC = 2;
16  COL_DAC_NOM = 1;
17
18  %open file
19  [FileName,PathName] = uigetfile('*.csv','Select the CSV-File to Plot');  %GUI um Datei auszuwählen
20  filePath=[PathName FileName];  %Zusammensetzen von Pfad und Filename
21  fileID = fopen(filePath, 'r');
22
23  %get titel and unit strings
24  endRow = getRowofEOF(fileID);
25  fseek(fileID, 0, 'bof'); %set file pointer to beginning of file
26  strTitel = fgets(fileID);
27  strUnits = fgets(fileID);
28  %strUnits(length(strUnits)) = '#';
29  [index, strDACnom] = getUnit(strUnits, 1);
30  [index, strDACdec] = getUnit(strUnits, index+1);
31  [index, strDAC_DMM] = getUnit(strUnits, index+1);
32  [index, strTimestamp] = getUnit(strUnits, index+1);
33  [index, strADCdec] = getUnit(strUnits, index+1);
```

```
34  [index, strADCval] = getUnit(strUnits, index+1);
35  [index, strADCDMM] = getUnit(strUnits, index+1);
36
37  %array initializations for better runtime
38  ADCvalDMM = startRow : endRow;
39  DoutADCdec = startRow : endRow;
40  DoutADCval = startRow : endRow;
41  mainLineValDMM = startRow : (endRow-startRow)/rangeRow : endRow;
42  mainLineValADC = startRow : (endRow-startRow)/rangeRow : endRow;
43  DMM_ADC_diff_abs = startRow : (endRow-startRow)/rangeRow : endRow;
44  DMM_ADC_diff_rel = startRow : (endRow-startRow)/rangeRow : endRow;
45  DMM_DAC_diff_abs = startRow : (endRow-startRow)/rangeRow : endRow;
46  DMM_DAC_diff_rel = startRow : (endRow-startRow)/rangeRow : endRow;
47  timestamp = startRow : (endRow-startRow)/rangeRow : endRow;
48  DACvalDMM = startRow : (endRow-startRow)/rangeRow : endRow;
49  DinDACVal = startRow : (endRow-startRow)/rangeRow : endRow;
50
51
52  %get decade factor
53  if strfind(strTitel, 'ADC_I')
54      factor = 1;%10^-6;
55      fac = 10^6;
56      [token, remain] = strtok(strADCDMM, '[');   %change '.. [A]' in '.. [µA]'
57      strADCDMM = [token '[µA]'];
58      [token, remain] = strtok(strADCval, '[');   %change '.. [A]' in '.. [µA]'
59      strADCval = [token '[µA]'];
60      strTitel = 'ADC_I';
61      strUnitADCval = '[µA]';
62  elseif strfind(strTitel, 'ADC_V')        %for case ADC_V and some else
63      fac = 1;
64      factor = 1;
65      strTitel = 'ADC_V';
66      strUnitADCval = '[V]';
67  end
68  m = 1;
69  for i = 0 : rangeRow : endRow-rangeRow
70      %read value block with one main and measure line from csv file
71      data = dlmread(filePath, ';', [i+startRow-1 startCol-1 i+startRow+rangeRow-2 endCol-1]);
72
73      %fetch values in repeated mesure lines
74      for k = 1 : 1 : rangeRow
75          ADCvalDMM(i+k) = data(k, COL_ADC_DMM);
76          DoutADCval(i+k) = data(k, COL_ADC_VAL);
77          DoutADCdec(i+k) = data(k, COL_ADC_DEC);
78      end
79
80      %fetch dac values in main measure line
81      DACvalDMM(m) = data(1, COL_DAC_DMM);
82      DinDACVal(m) = data(1, COL_DAC_NOM);
83
84      %fetch only values in main measure line
85      mainLineValDMM(m) = data(1, COL_ADC_DMM);
86      mainLineValADC(m) = data(1, COL_ADC_VAL);
87
88      %calculate ADC differences
89      DMM_ADC_diff_abs(m) = abs((mainLineValDMM(m) - mainLineValADC(m)*factor));
90      DMM_ADC_diff_rel(m) = abs((mainLineValDMM(m) - mainLineValADC(m)*factor) / mainLineValDMM(m));
91      timestamp(m) = data(1, COL_TIMESTAMP);
92
93      %calculate DAC differences
94      DMM_DAC_diff_abs(m) = (DACvalDMM(m) - DinDACVal(m));
95      DMM_DAC_diff_rel(m) = abs((DACvalDMM(m) - DinDACVal(m)) / DACvalDMM(m));
96
97      m = m + 1;
98  end
99
100 figure(1);      %plot ADCval(DMMval)
101 plot(ADCvalDMM*fac, DoutADCval*fac, 'r.', mainLineValDMM*fac, mainLineValADC*fac); grid on;
102 title([strTitel ' characteristics'], 'Fontsize', 18); set(gca,'Fontsize',14);
103 xlabel(strADCDMM, 'Fontsize', 18); ylabel(strADCval, 'Fontsize', 18);
104 legend([int2str(rangeRow-1) ' measures per step'], 'mainline steps');
105
106 figure(2);      %plot differences between ADCval - DMMval
107 subplot(2,1,1); plot(DinDACVal, DMM_ADC_diff_abs*fac); grid on;
108 title(['Absolute deviation ' strTitel], 'Fontsize', 18); set(gca,'Fontsize',14);
109 xlabel(strADCDMM, 'Fontsize', 18); ylabel(['absolute deviation ' strUnitADCval], 'Fontsize', 18);
110 subplot(2,1,2); plot(DinDACVal, DMM_ADC_diff_rel); grid on;
111 title(['Relative deviation ' strTitel], 'Fontsize', 18); set(gca,'Fontsize',14);
112 xlabel(strADCDMM, 'Fontsize', 18); ylabel('rel deviation [%]', 'Fontsize', 18);
113
114 % figure(2);      %plot ADCdec(DMMval)
115 % plot(ADCvalDMM, DoutADCdec); grid on;
116 % title(strTitel);
117 % xlabel('ImessDMM'); ylabel(strADCdec);
118
```

```
119 figure(3);   %plot differences between DinDACVal - DACvalDMM
120 subplot(2,1,1); plot(DinDACVal, DMM_DAC_diff_abs); grid on;
121 title('gain error/deviation DAC'); set(gca,'Fontsize',14);
122 xlabel(strDACnom); ylabel('deviation [V]');  ylim([-0.02 DMM_DAC_diff_abs(1)]);
123 % subplot(3,1,2); plot(DinDACVal, abs(DMM_DAC_diff_abs)); grid on;
124 % title('Absolute deviation DAC'); set(gca,'Fontsize',14);
125 % xlabel(strDACnom); ylabel('abs deviation [V]'); ylim([-0.02 DMM_DAC_diff_abs(1)]);
126 subplot(2,1,2); plot(DinDACVal, DMM_DAC_diff_rel); grid on;
127 title('Relative deviation DAC', 'Fontsize', 18); set(gca,'Fontsize',14);
128 xlabel(strDACnom, 'Fontsize', 18); ylabel('rel deviation [%]', 'Fontsize', 18);
129 ylim([-0.5 DMM_DAC_diff_rel(1)]);
130
131 figure(4);      %plot only mainlines DAC characteristics
132 plot(DACvalDMM, DinDACVal); grid on;
133 title('DAC characteristic', 'Fontsize', 18); set(gca,'Fontsize',14);
134 xlabel(strDAC_DMM, 'Fontsize', 18); ylabel(strDACnom, 'Fontsize', 18);
135
136 % format long e
137
138 %release all
139 fclose(fileID);
```

B.2.2. getRowofEOF.m

```
1  function [ endline ] = getRowofEOF(fileID)
2  %This function returns the last line number in a file
3      endline = 0;
4
5      tline = fgets(fileID);
6      while ischar(tline)
7          endline = endline + 1;
8          tline = fgets(fileID);
9      end
10 end
11
12
13 % fid = fopen(filePath);
14 % pos = 0
15 % fseek(fid, pos, 'bof');
16 % if ~feof(fid)
17 %     pos = pos + 1;
18 % end
```

B.2.3. getUnit.m

```
1  function [ index unit ] = getUnit(strUnits, startIndex)
2      n = 1;
3      lastIndex = length(strUnits);
4      %lastIndex = strfind(strUnits, '\n');
5      for i = startIndex : 1 : lastIndex
6          if strUnits(i) == ';' || i == lastIndex
7              index = i;
8              for j = startIndex : 1 : index-1
9                  unit(n) = strUnits(j);
10                 n = n + 1;
11             end
12             break;
13         end
14     end
15 end
```

C. BatCycle Quellcodes

C.1. BCMP_Server

C.1.1. BCMP_ServerMain.c

```
1  /************************************************************
2   * Server process for measurement control.
3   *
4   * File:    BCMP_ServerMain/BCMP_ServerMain.c
5   * Version: 0.4
6   * Date:    2016-02-18
7   * Author:  Jan Griessbach, Nico Rieckmann
8   *
9   * Changes: 0.4: JG, NR added SIGPIPE suppression and changed write()
10  *               in send() with MSG_NOSIGNAL flag into BC_sendFrame().
11  *          0.3: NR added termination by master client and usage
12  *               of main function arguments.
13  *          0.2: JG, NR added continuous mode and BCMP command cases.
14  *          0.1: JG, NR first version
15  *
16  ************************************************************/
17
18  #include "BCMP_Server.h"
19
20  /* global variables */
21  volatile sig_atomic_t runFlag = true;
22  volatile sig_atomic_t timerFlag = false;
23
24  /* signal handlers */
25  void sigEnd(int sign) {
26      runFlag = false;
27  }
28
29  void timerTickHandler(int sign) {
30      timerFlag = true;
31  }
32
33  int main(int argc, char* argv[]) {
34      int                 i, j;                       //Iterations
35      int                 readyClients, retval;       //function return values
36      fd_set              tempset;                    //temporary descriptor set
37      struct socket_fds   sock;                       //socket descriptors
38      int                 idx;                        //free index in conn_fd[]
39      char                *sockFilePath;              //socketfile path on filesystem
40      struct timeval      timeout;                    //timeout for select()
41      struct sockaddr_un  address;                    //socket struct
42      MPFrame_t           frame;                      //measurementProtocoll frame
43      sigset_t            signalset;                  //signalset to bundle signals
44      timer_t             timer_id;                   //timer for continuous modus
45      struct itimerspec   timer;
46      struct sigevent     signalevent;
47      struct sigaction    signalaction;
48
49      printf("#%s start\n", argv[0]);
50      errno = 0;      //reset error code
51
52      sockFilePath = UDS_FILE;
53      sock.arrSize = MAX_CLIENTS_DEFAULT;
54      //evaluate main function arguments
55      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
56          switch(retval) {
57          case 'p':       //path of the socketfile
58              sockFilePath = optarg;
59              #ifdef DEBUG
```

```
60              printf(".p argument: %s\n", optarg);
61              #endif
62              break;
63          case 'n':       //number of maximal clients
64              sock.arrSize = atoi(optarg);
65              #ifdef DEBUG
66              printf(".n argument: %d\n", atoi(optarg));
67              #endif
68              break;
69          case 'h':       //help text
70              usage(argv[0]);
71              exit(EXIT_SUCCESS);
72          default: ;
73  /*              usage(argv[0]);
74              exit(EXIT_FAILURE);*/
75          }
76      }
77      optind = 1; //after getopt finished, optint is -1
78
79      //initialize Hardware
80      retval = init_hardware(argc, argv);
81      if(retval < 0) {
82          perror("can not initialize Hardware");
83          reset_hardware();
84          return EXIT_FAILURE;
85      }
86
87      umask(0);       //set process permissions (don't limit), later for socket access
88
89      initConnectionSet(&sock);   //initialize the set of descriptors
90
91      //create socket
92      sock.sock_fd = socket(AF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0);
93      if(sock.sock_fd < 0) {
94          perror("Server: Failure at socket call");
95          return EXIT_FAILURE;
96      }
97
98      unlink(sockFilePath);    //deletes an already existing socket file
99
100     //init structs and variables
101     address.sun_family = AF_LOCAL;       //Unix-Domain Socket
102     strcpy(address.sun_path, sockFilePath);
103
104     //bind socket with file
105     if(bind(sock.sock_fd, (struct sockaddr*) &address, sizeof (address)) != 0) {
106         perror("Server: can not bind socket");
107         return EXIT_FAILURE;
108     }
109
110     //refresh socket list
111     if(listen(sock.sock_fd, sock.arrSize) < 0) {
112         perror("Server: listen failure");
113         return EXIT_FAILURE;
114     }
115
116     // setup and start timer
117     sigfillset(&signalset);     //fill set with all signals
118     memset(&timer, 0x00, sizeof(timer));     //init structs with zeros
119     memset(&signalevent, 0x00, sizeof(signalevent));
120     memset(&signalaction, 0x00, sizeof(signalaction));
121     signalevent.sigev_notify  = SIGEV_SIGNAL;
122     signalevent.sigev_signo   = SIGUSR1;        //register user defined signal
123     signalaction.sa_flags     = 0;//SA_RESTART;     //0 for linux behaviour, SA_RESTART for BSD OS
124     signalaction.sa_handler   = &timerTickHandler;      //register signalhandler
125     signalaction.sa_mask      = signalset;      //add all signals to signalmask
126
127     sigaction(SIGUSR1, &signalaction, NULL);    //register signalhandler for SIGUSR1 signal
128     if(timer_create(CLOCK_REALTIME, &signalevent, &timer_id) < 0) {
129         fprintf(stderr, "Error in timer_create(): Can't create timer.\n");
130         return EXIT_FAILURE;
131     }
132
133     signal(SIGINT, sigEnd);         //register signalhandler for SIGINT signal
134     signal(SIGPIPE, SIG_IGN);       //ignore global a SIGPIPE signal if socket access crashed
135
136     fprintf(stdout, "Server is listening for incomming messages...\n");
137     updateConnectionSet(&sock);
138     while(runFlag) {
139         //check if already connected clients are ready
140         timeout.tv_sec = TIMEOUT_SEC;
141         timeout.tv_usec = TIMEOUT_USEC;
142         tempset = sock.readset;
143         readyClients = select(sock.max_fd + 1, &tempset, NULL, NULL, &timeout);
144         if (readyClients == 0) {        //if 0 clients are ready
```

```
145             #ifdef DEBUG
146             fprintf(stdout, ".select() time out. no more messages in socket\n");
147             #endif
148         }
149         else if (readyClients < 0 && errno != EINTR) {  //if error occurred in select()
150             perror("Error occurred in select()");
151         }
152         if(timerFlag) {
153             #ifdef DEBUG
154             printf(".timertick:\n");
155             #endif
156             BCMP_initializeFrame(&frame, BCMP_RES_VALUE);   //prepare value frame
157             BC_setArgumentAsUint64(&frame, BCMP_RES_VALUE_OFFSET_TIMESTAMP, getTimestamp());    //put timestamp into frame
158             BC_setArgumentAsDouble(&frame, BCMP_RES_VALUE_OFFSET_VALUE, getValue());    //put value into frame
159             //send to all connected clients
160             for(i=0; i<sock.arrSize; i++) {
161                 if(sock.conn_fd[i] >= 0)
162                     sendToClient(&frame, &sock, i);
163             }
164             timerFlag = false;
165             continue;
166         }
167 
168         if (readyClients > 0) {  //if any clients are ready
169             for (j=0; j<sock.arrSize; j++) {      //iterate all connected socket descriptors
170                 //check if current descriptor was set by select() (is ready for read)
171                 if (sock.conn_fd[j] >= 0 && FD_ISSET(sock.conn_fd[j], &tempset)) {
172                     retval = BC_recvFrame(sock.conn_fd[j], &frame);    //read data from socket, nonblocking
173                     if(retval <= 0) {           //if error occurred e.g. current connection loss.
174                         close(sock.conn_fd[j]);
175                         sock.conn_fd[j] = -1;
176                         updateConnectionSet(&sock);
177                         fprintf(stdout, "Error %d occurred in BC_recvFrame() or client terminated: %d\n", retval, errno);
178                         fprintf(stdout, "Client %d deleted\n", j);
179                     }
180                     else if (retval > 0) { //if an frame received
181                         #ifdef DEBUG
182                         BC_dumpFrameData(&frame);
183                         #endif
184 
185                         //if client is not master and a master exists or master doesn't exist and client didn't send a get control
186                         if((sock.conn_fd[j] != sock.master_fd && sock.master_fd >= 0) ||
187                            (sock.master_fd < 0 && frame.fields.command != BCMP_REQ_GETCONTROL)) {
188                             //prepare denial frame
189                             BCMP_initializeFrame(&frame, BCMP_RES_CONTROL);
190                             BC_setArgumentAsUint8(&frame, BCMP_RES_CONTROL_OFFSET_CONTROL, BCMP_RES_CONTROL_ARGVAL_CONTROL_DENIAL);
191                             sendToClient(&frame, &sock, j);      //send denial frame to client
192                             fprintf(stdout, "send an denial frame\n");
193                             #ifdef DEBUG
194                             BC_dumpFrameData(&frame);
195                             #endif
196                             continue;
197                         }
198                         switch(frame.fields.command) {
199                         case BCMP_REQ_GETVALUE:      //received a get value frame
200                             //prepare value frame
201                             BCMP_initializeFrame(&frame, BCMP_RES_VALUE);
202                             BC_setArgumentAsUint64(&frame, BCMP_RES_VALUE_OFFSET_TIMESTAMP, getTimestamp()); //put timestamp into frame
203                             BC_setArgumentAsDouble(&frame, BCMP_RES_VALUE_OFFSET_VALUE, getValue());    //put value into frame
204                             for(i=0; i<sock.arrSize; i++) {  //send to all connected clients
205                                 if(sock.conn_fd[i] >= 0)
206                                     sendToClient(&frame, &sock, i);
207                             }
208                             break;
209                         case BCMP_REQ_STARTCONTINUOUS:       //received a start continuous frame
210                             //start timer with received interval from client
211                             setTimer(timer_id, &timer, BC_getArgumentAsUint32(&frame, BCMP_REQ_STARTCONTINUOUS_OFFSET_INTERVAL));
212                             break;
213                         case BCMP_REQ_STOPCONTINUOUS:        //received a stop continuous frame
214                             setTimer(timer_id, &timer, 0);      //0 stops timer
215                             break;
216                         case BCMP_REQ_GETCONTROL:        //received a get control frame
217                             sock.master_fd = sock.conn_fd[j];      //save client with control
218                             //prepare grant frame
219                             BCMP_initializeFrame(&frame, BCMP_RES_CONTROL);
220                             BC_setArgumentAsUint8(&frame, BCMP_RES_CONTROL_OFFSET_CONTROL, BCMP_RES_CONTROL_ARGVAL_CONTROL_GRANT);
221                             sendToClient(&frame, &sock, j);      //send frame to client
222                             break;
223                         case BCMP_REQ_RETCONTROL:        //received a return control frame
224                             sock.master_fd = -1;     //reset master client
225                             //prepare release frame
226                             BCMP_initializeFrame(&frame, BCMP_RES_CONTROL);
227                             BC_setArgumentAsUint8(&frame, BCMP_RES_CONTROL_OFFSET_CONTROL, BCMP_RES_CONTROL_ARGVAL_CONTROL_RELEASE);
228                             sendToClient(&frame, &sock, j);      //send frame to client
229                             break;
```

```
230                         case BCMP_REQ_QUIT:     //received a quit frame
231                             fprintf(stdout, "Server was quitted by authorized master client\n");
232                             runFlag = false;
233                             break;
234                         default: printf("Unknown command in frame\n"); //received something else
235                         }
236                     }
237                 }
238             }
239         }
240
241         //check for a new incomming connection
242         if (FD_ISSET(sock.sock_fd, &tempset)) {
243             int peer_fd;
244
245             peer_fd = accept4(sock.sock_fd, NULL, NULL, SOCK_NONBLOCK); //accept new client, non-blocking mode
246             if (peer_fd < 0) {
247                 perror("Error occurred in accept()");
248             }
249             else {
250                 idx = getnextfreeIndex(sock.conn_fd, sock.arrSize);
251                 if (idx >= 0) {  //check if not more than MAX_CLIENTS are connected
252                     sock.conn_fd[idx] = peer_fd;
253                     updateConnectionSet(&sock);
254
255                     fprintf(stdout, "Client %d connected\n", idx);
256
257                 }
258                 else {
259                     close(peer_fd);
260                     fprintf(stdout, "Limit of %d connected clients reached\n", sock.arrSize);
261                 }
262             }
263         }
264     }
265
266     //quit properly all used resources
267     reset_hardware();
268     for(i=0; i<sock.arrSize; i++) {     //close all used descriptors
269         int fd = sock.conn_fd[i];
270
271         if(fd > 0) {
272             shutdown(fd, SHUT_RDWR);
273             close(fd);
274         }
275     }
276     close(sock.sock_fd);
277     unlink(sockFilePath);               //delete the used socket file
278     free(sock.conn_fd);
279
280     printf("#%s end\n", argv[0]);
281     return EXIT_SUCCESS;
282 }
```

C.1.2. BCMP_Server.c

```
1  /*************************************************************
2   * Server process for measurement control.
3   *
4   * File:    BCMP_ServerMain/BCMP_Server.c
5   * Version: 0.4
6   * Date:    2016-02-18
7   * Author:  Jan Griessbach, Nico Rieckmann
8   *
9   * Changes: 0.4: JG, NR added SIGPIPE suppression and changed write()
10  *               in send() with MSG_NOSIGNAL flag into BC_sendFrame().
11  *          0.3: NR added termination by master client and usage
12  *               of main function arguments.
13  *          0.2: JG, NR added continuous mode and BCMP command cases.
14  *          0.1: JG, NR first version
15  *
16  *************************************************************/
17
18 #include "BCMP_Server.h"
19
20 int getnextfreeIndex(int *fdArr, int len) {
21     int i;
22
23     for(i=0; i<len; i++) {
```

```
24          if(fdArr[i] < 0)
25              return i;
26      }
27      return -1;
28  }
29
30  void updateConnectionSet(struct socket_fds* sock) {
31      int i, masterfd = -1;
32
33      sock->max_fd = sock->sock_fd;  //set fd for incomming connections to maxfd first
34      FD_ZERO(&sock->readset);           //clear the descriptor set
35      FD_SET(sock->sock_fd, &sock->readset);     //insert file descriptors
36      for(i=0; i<sock->arrSize; i++) {
37          int fd = sock->conn_fd[i];
38
39          if(fd >= 0){
40              FD_SET(fd, &sock->readset);               //add descriptor to readset
41              sock->max_fd = (fd > sock->max_fd) ? fd : sock->max_fd;   //maxfd must be biggest descriptor
42              if(fd == sock->master_fd)       //check for existing master client
43                  masterfd = sock->master_fd;
44          }
45      }
46      sock->master_fd = masterfd; //set master client if exists, else -1
47  }
48
49  int initConnectionSet(struct socket_fds *sock) {
50      int i;
51
52      sock->conn_fd = malloc(sock->arrSize * sizeof(int));
53      if(sock->conn_fd == NULL)
54          return -1;
55      for(i=0; i<sock->arrSize; i++)    //init descriptor array
56          sock->conn_fd[i] = -1;
57      sock->master_fd = -2;   //different initial value than -1
58      return 1;
59  }
60
61  int setTimer(timer_t id, struct itimerspec* timer, uint32_t msInterval) {
62      //starts timer when msInterval > 0. Stop timer when msInterval == 0.
63      timer->it_value.tv_sec     = msInterval / 1000;         //starttime offset
64      timer->it_value.tv_nsec    = (msInterval % 1000) * 1E6;
65      timer->it_interval.tv_sec  = msInterval / 1000;         //interval time
66      timer->it_interval.tv_nsec = (msInterval % 1000) * 1E6;
67      return timer_settime(id, 0, timer, NULL);              //start timer
68  }
69
70  int sendToClient(MPFrame_t *frame, struct socket_fds* sock, int index) {
71      int retval;
72
73      retval = BC_sendFrame(sock->conn_fd[index], frame);
74      if(retval < 0) {
75          close(sock->conn_fd[index]);
76          sock->conn_fd[index] = -1;
77          updateConnectionSet(sock);
78          perror("Error occurred in BC_sendFrame()");
79          fprintf(stdout, "Client %d deleted due to send error\n", index);
80      }
81      return retval;
82  }
83
84  uint64_t getTimestamp(void) {   //return in msec since 1970
85      struct timeval time;
86
87      memset(&time, 0x00, sizeof(time));  //init structure
88      gettimeofday(&time, NULL);          //get time stamp
89      return ((uint64_t) time.tv_sec) * 1000 + time.tv_usec / 1000;   //sec + msec
90  }
```

C.1.3. BCMP_Server.h

```
1   /************************************************************
2    * Server process for measurement control.
3    *
4    * File:    BCMP_ServerMain/BCMP_Server.h
5    * Version: 0.4
6    * Date:    2016-02-18
7    * Author:  Jan Griessbach, Nico Rieckmann
8    *
9    * Changes: 0.4: JG, NR added SIGPIPE suppression and changed write()
```

```
10   *              in send() with MSG_NOSIGNAL flag into BC_sendFrame().
11   *         0.3: NR added termination by master client and usage
12   *              of main function arguments.
13   *         0.2: JG, NR added continuous mode and BCMP command cases.
14   *         0.1: JG, NR first version
15   *
16   ************************************************************/
17
18   #ifndef BCMP_SERVER_H_
19   #define BCMP_SERVER_H_
20
21   /* uds_server.c */
22   #include <stdlib.h>
23   #include <stdio.h>
24   #include <errno.h>
25   #include <unistd.h>
26   //for sockets
27   #include <sys/types.h>
28   #include <sys/stat.h>   //for umask
29   #define __USE_GNU       //for accept4 function
30   #include <sys/socket.h>
31   #include <sys/un.h>
32   #include <netinet/in.h>
33   #include <arpa/inet.h>
34   #include <string.h>
35   #include <sys/select.h>
36   //for signalhandler and timer
37   #include <signal.h>
38   #include <time.h>
39   //for boolean
40   #include <stdbool.h>
41   //for timestamp
42   #include <sys/time.h>
43   //for client server communication
44   #include "BC_Protocol/BC_MeasurementProtocol.h"
45   #include "HW_Implementation/BC_HW_Implementation.h"
46
47   //#define DEBUG
48   #ifdef DEBUG
49       #define TIMEOUT_SEC         1
50       #define TIMEOUT_USEC        0
51   #else
52       #define TIMEOUT_SEC         0
53       #define TIMEOUT_USEC        10000
54   #endif
55
56   //socket file descriptor struct
57   struct socket_fds {
58       int     sock_fd;                //descriptor for incomming connections
59       int     master_fd;              //master client descriptor
60       int     *conn_fd;               //dynamic client descriptor array
61       int     arrSize;                //dynamic client array size
62       int     max_fd;                 //maximal descriptor value
63       fd_set  readset;                //set of descriptors
64   };
65
66   /* prototypes */
67   int      initConnectionSet(struct socket_fds *sock);
68   void     updateConnectionSet(struct socket_fds *sock);
69   int      getnextfreeIndex(int *fdArr, int len);
70   int      setTimer(timer_t id, struct itimerspec* timer, uint32_t msInterval);
71   int      sendToClient(MPFrame_t *frame, struct socket_fds* sock, int index);
72   uint64_t getTimestamp(void);
73   double   getValue(void);
74
75   /* functions with different implementations */
76   extern int    init_hardware(int argc, char* argv[]);
77   extern double getValue(void);
78   extern void   reset_hardware(void);
79   extern void   usage(char *progname);
80
81   #endif /* BCMP_SERVER_H_ */
```

C.2. BCAP_Server

C.2.1. BCAP_ServerMain.c

```
1  /************************************************************
2   * Server process for actuator control
3   *
4   * File:    BCAP_ServerMain/BCAP_ServerMain.c
5   * Version: 0.1
6   * Date:    2016-02-28
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version.
10  *               Covered from BCMP_ServerMain.c
11  *
12  ************************************************************/
13
14 #include "BCAP_Server.h"
15
16 /* global variables */
17 volatile sig_atomic_t runFlag = true;
18
19 /* signal handlers */
20 void sigEnd(int sign) {
21     runFlag = false;
22 }
23
24 int main(int argc, char* argv[]) {
25     int                 i, j;                       //Iterations
26     int                 readyClients, retval;       //function return values
27     fd_set              tempset;                    //temporary descriptor set
28     struct socket_fds   sock;                       //socket descriptors
29     int                 idx;                        //free index in conn_fd[]
30     char                *sockFilePath;              //socketfile path on filesystem
31     struct timeval      timeout;                    //timeout for select()
32     struct sockaddr_un  address;                    //socket struct
33     BCFrame_t           frame;                      //Protocol frame
34
35     printf("#%s start\n", argv[0]);
36     errno = 0;          //reset error code
37
38     sockFilePath = UDS_FILE;
39     sock.arrSize = MAX_CLIENTS_DEFAULT;
40     //evaluate main function arguments
41     while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
42         switch(retval) {
43         case 'p':       //path of the socketfile
44             sockFilePath = optarg;
45             #ifdef DEBUG
46             printf(".p argument: %s\n", optarg);
47             #endif
48             break;
49         case 'n':       //number of maximal clients
50             sock.arrSize = atoi(optarg);
51             #ifdef DEBUG
52             printf(".n argument: %d\n", atoi(optarg));
53             #endif
54             break;
55         case 'h':       //help text
56         default:
57             usage(argv[0]);
58             exit(EXIT_FAILURE);
59         }
60     }
61     optind = 1; //after getopt finished, optint is -1
62
63     umask(0);       //set process permissions (don't limit), later for socket access
64     initConnectionSet(&sock);   //initialize the set of descriptors
65
66     //create socket
67     sock.sock_fd = socket(AF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0);
68     if(sock.sock_fd < 0) {
69         perror("Server: Failure at socket call");
70         return EXIT_FAILURE;
71     }
72
73     unlink(sockFilePath);    //deletes an already existing socket file
74
75     //init structs and variables
76     address.sun_family = AF_LOCAL;       //Unix-Domain Socket
77     strcpy(address.sun_path, sockFilePath);
```

```
78
79      //bind socket with file
80      if(bind(sock.sock_fd, (struct sockaddr*) &address, sizeof (address)) != 0) {
81          perror("Server: can not bind socket");
82          return EXIT_FAILURE;
83      }
84
85      //refresh socket list
86      if(listen(sock.sock_fd, sock.arrSize) < 0) {
87          perror("Server: listen failure");
88          return EXIT_FAILURE;
89      }
90
91      //initialize Hardware
92      retval = init_hardware(argc, argv);
93      if(retval < 0) {
94          perror("Can not initialize hardware");
95          reset_hardware();
96          return EXIT_FAILURE;
97      }
98
99      signal(SIGINT, sigEnd);         //register signalhandler for SIGINT signal
100     signal(SIGPIPE, SIG_IGN);       //ignore global a SIGPIPE signal if socket access crashed
101
102     fprintf(stdout, "Server is listening for incomming messages...\n");
103     updateConnectionSet(&sock);
104     while(runFlag) {
105         //check if already connected clients are ready
106         timeout.tv_sec = TIMEOUT_SEC;
107         timeout.tv_usec = TIMEOUT_USEC;
108         tempset = sock.readset;
109         readyClients = select(sock.max_fd + 1, &tempset, NULL, NULL, &timeout);
110         if (readyClients == 0) {        //if 0 clients are ready
111             #ifdef DEBUG
112             fprintf(stdout, ".select() time out. no more messages in socket\n");
113             #endif
114         }
115         else if (readyClients < 0 && errno != EINTR) {  //if error occurred in select()
116             perror("Error occurred in select()");
117         }
118
119         if (readyClients > 0) {  //if any clients are ready
120             for (j=0; j<sock.arrSize; j++) {      //iterate all connected socket descriptors
121                 //check if current descriptor was set by select() (is ready for read)
122                 if (sock.conn_fd[j] >= 0 && FD_ISSET(sock.conn_fd[j], &tempset)) {
123                     retval = BC_recvFrame(sock.conn_fd[j], &frame);    //read data from socket, nonblocking
124                     if(retval <= 0) {           //if error occurred e.g. current connection loss.
125                         close(sock.conn_fd[j]);
126                         sock.conn_fd[j] = -1;
127                         updateConnectionSet(&sock);
128                         fprintf(stdout, "Error %d occurred in BC_recvFrame() or client terminated: %d\n", retval, errno);
129                         fprintf(stdout, "Client %d deleted\n", j);
130                     }
131                     else if (retval > 0) { //if an frame received
132                         #ifdef DEBUG
133                         BC_dumpFrameData(&frame);
134                         #endif
135
136                         retval = behavioral((BCFrame_t*) &frame);
137                         if(retval < 0) {
138                             runFlag = false;
139                             if(retval == -1)
140                                 fprintf(stderr, "Error occured in behavioral()\n");
141                             else if(retval == -2)
142                                 fprintf(stdout, "Server was quitted by client\n");
143                         }
144
145                     }
146                 }
147             }
148         }
149
150         //check for a new incomming connection
151         if(FD_ISSET(sock.sock_fd, &tempset)) {
152             int peer_fd;
153
154             peer_fd = accept4(sock.sock_fd, NULL, NULL, SOCK_NONBLOCK); //accept new client, non-blocking mode
155             if (peer_fd < 0) {
156                 perror("Error occurred in accept()");
157             }
158             else {
159                 idx = getnextfreeIndex(sock.conn_fd, sock.arrSize);
160                 if (idx >= 0) {  //check if not more than MAX_CLIENTS are connected
161                     sock.conn_fd[idx] = peer_fd;
162                     updateConnectionSet(&sock);
```

```
163                     fprintf(stdout, "Client %d connected\n", idx);
164                 }
165                 else {
166                     close(peer_fd);
167                     fprintf(stdout, "Limit of %d connected clients reached\n", sock.arrSize);
168                 }
169             }
170         }
171     }
172
173     //quit properly all used resources
174     reset_hardware();
175     for(i=0; i< sock.arrSize; i++) {    //close all used descriptors
176         int fd = sock.conn_fd[i];
177
178         if(fd > 0) {
179             shutdown(fd, SHUT_RDWR);
180             close(fd);
181         }
182     }
183     close(sock.sock_fd);
184     unlink(sockFilePath);               //delete the used socket file
185     free(sock.conn_fd);
186
187     printf("#%s end\n", argv[0]);
188     return EXIT_SUCCESS;
189 }
```

C.2.2. BCAP_Server.c

```
1  /*************************************************************
2   * Server process for actuator control
3   *
4   * File:    BCAP_ServerMain/BCAP_Server.c
5   * Version: 0.1
6   * Date:    2016-02-28
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version.
10  *               Covered from BCMP_ServerMain.c
11  *
12  *************************************************************/
13
14 #include "BCAP_Server.h"
15
16 int getnextfreeIndex(int *fdArr, int len) {
17     int i;
18
19     for(i=0; i<len; i++) {
20         if(fdArr[i] < 0)
21             return i;
22     }
23     return -1;
24 }
25
26 void updateConnectionSet(struct socket_fds* sock) {
27     int i, masterfd = -1;
28
29     sock->max_fd = sock->sock_fd;  //set fd for incomming connections to maxfd first
30     FD_ZERO(&sock->readset);           //clear the descriptor set
31     FD_SET(sock->sock_fd, &sock->readset);     //insert file descriptors
32     for(i=0; i<sock->arrSize; i++) {
33         int fd = sock->conn_fd[i];
34
35         if(fd >= 0){
36             FD_SET(fd, &sock->readset);                //add descriptor to readset
37             sock->max_fd = (fd > sock->max_fd) ? fd : sock->max_fd;   //maxfd must be biggest descriptor
38             if(fd == sock->master_fd)       //check for existing master client
39                 masterfd = sock->master_fd;
40         }
41     }
42     sock->master_fd = masterfd; //set master client if exists, else -1
43 }
44
45 int initConnectionSet(struct socket_fds *sock) {
46     int i;
47
48     sock->conn_fd = malloc(sock->arrSize * sizeof(int));
49     if(sock->conn_fd == NULL)
```

```
50          return -1;
51      for(i=0; i<sock->arrSize; i++)    //init descriptor array
52          sock->conn_fd[i] = -1;
53      sock->master_fd = -2;   //different initial value than -1
54      return 1;
55  }
```

C.2.3. BCAP_Server.h

```
1   /*************************************************************
2    * Server process for actuator control
3    *
4    * File:    BCAP_ServerMain/BCAP_Server.h
5    * Version: 0.1
6    * Date:    2016-02-28
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *               Covered from BCMP_ServerMain.c
11   *
12   *************************************************************/
13
14  #ifndef BCAP_SERVER_H_
15  #define BCAP_SERVER_H_
16
17  /* uds_server.c */
18  #include <stdlib.h>
19  #include <stdio.h>
20  #include <errno.h>
21  #include <unistd.h>
22  //for sockets
23  #include <sys/types.h>
24  #include <sys/stat.h>   //for umask
25  #define __USE_GNU       //for accept4 function
26  #include <sys/socket.h>
27  #include <sys/un.h>
28  #include <netinet/in.h>
29  #include <arpa/inet.h>
30  #include <string.h>
31  #include <sys/select.h>
32  //for signalhandler and timer
33  #include <signal.h>
34  #include <time.h>
35  //for boolean
36  #include <stdbool.h>
37  //for timestamp
38  #include <sys/time.h>
39  //for client server communication
40  #include "HW_Implementation/BC_HW_Implementation.h"
41
42  //#define DEBUG
43  #ifdef DEBUG
44      #define TIMEOUT_SEC         1           //1 sec
45      #define TIMEOUT_USEC        0
46  #else
47      #define TIMEOUT_SEC         0
48      #define TIMEOUT_USEC        10000       //10 msec
49  #endif
50
51  //socket file descriptor struct
52  struct socket_fds {
53      int     sock_fd;                //descriptor for incomming connections
54      int     master_fd;              //master client descriptor
55      int     *conn_fd;               //dynamic client descriptor array
56      int     arrSize;                //dynamic client array size
57      int     max_fd;                 //maximal descriptor value
58      fd_set  readset;                //set of descriptors
59  };
60
61  /* prototypes */
62  int     initConnectionSet(struct socket_fds *sock);
63  void    updateConnectionSet(struct socket_fds *sock);
64  int     getnextfreeIndex(int *fdArr, int len);
65
66  /* functions with different implementations */
67  extern int    init_hardware(int argc, char* argv[]);
68  extern void   reset_hardware(void);
69  extern void   usage(char *progname);
70  extern int    behavioral(BCFrame_t *frame);
```

```
71
72  #endif /* BCAP_SERVER_H_ */
```

C.3. BC_Protocol

C.3.1. BC_Protocol.c

```
1   /*************************************************************
2    * Basic functions for all BatCycle protocols.
3    *
4    * File:    BC_Protocol.c
5    * Version: 0.5
6    * Date:    2016-02-12
7    * Author:  Jan Griessbach, Nico Rieckmann
8    *
9    * Changes: 0.5: NR changed write() to send() fkt into BC_sendFrame(),
10   *               to suppress a SIGPIPE throw if connection broke.
11   *          0.4: NR fix BC_dumpFrameData function.
12   *          0.3: NR changed BC_recvFrame and BC_dumpFrameData.
13   *          0.2: NR added set- and get- functions for uint8_t
14   *               and MAKROS for frame termination error.
15   *          0.1: first version.
16   *
17   *************************************************************/
18
19  #include <stdio.h>
20  #include <unistd.h>
21  #include <math.h>
22  #include <string.h>
23  #include <stdint.h>
24  #include <stdbool.h>
25  #include <sys/types.h>
26  #include <sys/socket.h>
27  #include "BC_Protocol.h"
28
29  void BC_initializeFrame(BCFrame_t *f, uint8_t command, uint8_t argLen)
30  {
31    /* set command and argument length */
32    f->fields.command = command;
33    f->fields.argLen  = argLen;
34    /* clear argument buffer */
35    memset(f->fields.argBuffer, 0x00, f->fields.argLen);
36    /* write frame end character */
37    GET_BC_END_CHARACTER(*f) = BC_FRAME_END_CHAR;
38  }
39
40  uint8_t BC_getArgumentAsUint8(BCFrame_t *f, unsigned int offset)
41  {
42    /* check argument size */
43    if(f->fields.argLen < (sizeof(uint8_t) + offset))
44      return 0;
45    return *((uint8_t*)(f->fields.argBuffer + offset));
46  }
47
48  uint32_t BC_getArgumentAsUint32(BCFrame_t *f, unsigned int offset)
49  {
50    /* check argument size */
51    if(f->fields.argLen < (sizeof(uint32_t) + offset))
52      return 0;
53    return *((uint32_t*)(f->fields.argBuffer + offset));
54  }
55
56  uint64_t BC_getArgumentAsUint64(BCFrame_t *f, unsigned int offset)
57  {
58    /* check argument size */
59    if(f->fields.argLen < (sizeof(uint64_t) + offset))
60      return 0;
61    return *((uint64_t*)(f->fields.argBuffer + offset));
62  }
63
64  float BC_getArgumentAsFloat(BCFrame_t *f, unsigned int offset) {
65    /* check argument size */
66    if(f->fields.argLen < (sizeof(float) + offset))
67      return NAN;
68    return *((float*)(f->fields.argBuffer + offset));
```

```
69  }
70
71  double BC_getArgumentAsDouble(BCFrame_t *f, unsigned int offset)
72  {
73    /* check argument size */
74    if(f->fields.argLen < (sizeof(double) + offset))
75      return NAN;
76    return *((double*)(f->fields.argBuffer + offset));
77  }
78
79  void BC_setArgumentAsUint8(BCFrame_t *f, unsigned int offset, uint8_t argument)
80  {
81    /* check argument size */
82    if(f->fields.argLen < (sizeof(uint8_t) + offset))
83      return;
84     *((uint8_t*)(f->fields.argBuffer + offset)) = argument;
85  }
86
87  void BC_setArgumentAsUint32(BCFrame_t *f, unsigned int offset, uint32_t argument)
88  {
89    /* check argument size */
90    if(f->fields.argLen < (sizeof(uint32_t) + offset))
91      return;
92     *((uint32_t*)(f->fields.argBuffer + offset)) = argument;
93  }
94
95  void BC_setArgumentAsUint64(BCFrame_t *f, unsigned int offset, uint64_t argument)
96  {
97    /* check argument size */
98    if(f->fields.argLen < (sizeof(uint64_t) + offset))
99      return;
100    *((uint64_t*)(f->fields.argBuffer + offset)) = argument;
101 }
102
103 void BC_setArgumentAsFloat(BCFrame_t *f, unsigned int offset, float argument) {
104   /* check argument size */
105   if(f->fields.argLen < (sizeof(float) + offset))
106     return;
107   *((float*)(f->fields.argBuffer + offset)) = argument;
108 }
109
110 void BC_setArgumentAsDouble(BCFrame_t *f, unsigned int offset, double argument)
111 {
112   /* check argument size */
113   if(f->fields.argLen < (sizeof(double) + offset))
114     return;
115   *((double*)(f->fields.argBuffer + offset)) = argument;
116 }
117
118 void BC_dumpFrameData(BCFrame_t *f)
119 {
120   fprintf(stderr, "Frame dump:\nAddress: %p;\t Command: 0x%X;\t ArgLen: %u;\n", f, f->fields.command, f->fields.argLen);
121   if(f->fields.argLen > 0){
122     int i, offset = 0;
123     fprintf(stderr, "Arguments:");
124     for(i=0; i<f->fields.argLen; i++) {
125       if(i%8 == 0) {
126         fprintf(stderr, "\n0x%02x:\t0x", offset);
127         offset += 8;
128       }
129       fprintf(stderr, " %02x", f->fields.argBuffer[i]);
130     }
131     fprintf(stderr,"\n");
132   }
133 }
134
135 int BC_sendFrame(int socketfd, const BCFrame_t *f)
136 {
137   //MSG_NOSIGNAL suppress a SIGPIPE throw
138   return (int) send(socketfd, f->buffer, GET_BC_FRAME_LENGTH(*f), MSG_NOSIGNAL);
139 }
140
141 int BC_recvFrame(int socketfd, BCFrame_t *f)
142 {
143   int retval;
144
145   //retval = read(socketfd, f->buffer, 2);
146   retval = recv(socketfd, f->buffer, 2, 0);
147   if(retval<=0)
148     return retval;
149   if(f->fields.argLen > BC_MAX_ARG_SIZE)
150     return BC_FRAME_SIZE_ERROR;
151   if(f->fields.argLen == 0) {
152     //retval = read(socketfd, f->buffer + 2, 1);  //read BC_FRAME_END_CHAR
153     retval = recv(socketfd, f->buffer + 2, 1, 0);  //read BC_FRAME_END_CHAR
```

```
154     }
155     else {
156         //read argLen many chars + BC_FRAME_END_CHAR
157         //retval = read(socketfd, f->fields.argBuffer, f->fields.argLen + 1);
158         retval = recv(socketfd, f->fields.argBuffer, f->fields.argLen + 1, 0);
159     }
160     if(retval > 0 && !CHECK_BC_END_CHARACTER(*f))     //if message has no BC_FRAME_END_CHAR
161       return BC_FRAME_TERMINATION_ERROR;
162     return retval;
163   }
```

C.3.2. BC_Protocol.h

```
1   /*************************************************************
2    * Basic functions for all BatCycle protocols.
3    *
4    * File:    BC_Protocol.h
5    * Version: 0.5
6    * Date:    2016-02-12
7    * Author:  Jan Griessbach, Nico Rieckmann
8    *
9    * Changes: 0.5: NR changed write() to send() fkt into BC_sendFrame(),
10   *               to suppress a SIGPIPE throw if connection broke.
11   *          0.4: NR fix BC_dumpFrameData function.
12   *          0.3: NR changed BC_recvFrame and BC_dumpFrameData.
13   *          0.2: NR added set- and get- functions for uint8_t
14   *               and MAKROS for Frame termination error.
15   *          0.1: first version.
16   *
17   *************************************************************/
18
19  #ifndef BC_PROTOCOL_H_
20  #define BC_PROTOCOL_H_
21
22  #include <stdio.h>
23  #include <unistd.h>
24  #include <math.h>
25  #include <string.h>
26  #include <stdint.h>
27  #include <stdbool.h>
28  #include <sys/types.h>
29  #include <sys/socket.h>
30
31  /* Maximum frame size */
32  #define BC_MAX_ARG_SIZE 20
33  #define BC_MAX_FRAME_SIZE (BC_MAX_ARG_SIZE + 3*sizeof(uint8_t))
34  /* End character */
35  #define BC_FRAME_END_CHAR '\n'
36
37  /* Error constants */
38  #define BC_FRAME_SIZE_ERROR             -2
39  #define BC_FRAME_TERMINATION_ERROR      -3
40
41  /* Macros */
42  #define GET_BC_END_CHARACTER(x)   ((x).buffer[2+(x).fields.argLen])
43  #define GET_BC_FRAME_LENGTH(x)    (sizeof(uint8_t)*(3+(x).fields.argLen))
44  #define CHECK_BC_END_CHARACTER(x) ((GET_BC_END_CHARACTER(x) == BC_FRAME_END_CHAR)? true : false)
45
46
47  /* Data types */
48  typedef union union_bc_frame {
49    unsigned char buffer[BC_MAX_FRAME_SIZE];
50    struct {
51      uint8_t command;
52      uint8_t argLen;
53      uint8_t argBuffer[BC_MAX_ARG_SIZE];
54    } fields;
55  } BCFrame_t;
56
57
58  /* function prototypes */
59  void     BC_initializeFrame(BCFrame_t *f, uint8_t command, uint8_t argLen);
60
61  uint8_t  BC_getArgumentAsUint8(BCFrame_t *f, unsigned int offset);
62  uint32_t BC_getArgumentAsUint32(BCFrame_t *f, unsigned int offset);
63  uint64_t BC_getArgumentAsUint64(BCFrame_t *f, unsigned int offset);
64  float    BC_getArgumentAsFloat(BCFrame_t *f, unsigned int offset);
65  double   BC_getArgumentAsDouble(BCFrame_t *f, unsigned int offset);
66  //float*   BC_getArgumentAsFloatArray(BCFrame_t *f, unsigned int offset);   //TODO kann weg
```

```
67
68  void     BC_setArgumentAsUint8(BCFrame_t *f, unsigned int offset, uint8_t argument);
69  void     BC_setArgumentAsUint32(BCFrame_t *f, unsigned int offset, uint32_t argument);
70  void     BC_setArgumentAsUint64(BCFrame_t *f, unsigned int offset, uint64_t argument);
71  void     BC_setArgumentAsFloat(BCFrame_t *f, unsigned int offset, float argument);
72  void     BC_setArgumentAsDouble(BCFrame_t *f, unsigned int offset, double argument);
73  //void     BC_getArgumentAsFloatArray(BCFrame_t *f, unsigned int offset, float* argumen); //TODO kann weg
74
75  void     BC_dumpFrameData(BCFrame_t *f);
76  int      BC_sendFrame(int socketfd, const BCFrame_t *f);
77  int      BC_recvFrame(int socketfd, BCFrame_t *f);
78
79  #endif /* BC_PROTOCOL_H_ */
```

C.3.3. BC_ActuatorProtocol.c

```
1   /*************************************************************
2    * Functions and definitions for the actuator protocol.
3    *
4    * File:    BC_ActuatorProtocol.c
5    * Version: 0.2
6    * Date:    2016-04-14
7    * Author:  Jan Griessbach
8    *
9    * Changes: 0.2: NR added command for ´LEDs and renamed from VRefProtocol
10   *               in ActuatorProtocol.
11   *          0.1: first version
12   *
13   *************************************************************/
14
15  #include "BC_ActuatorProtocol.h"
16
17  void BCAP_initializeFrame(APFrame_t *f, APCommand_t command)
18  {
19    BC_initializeFrame(f, (uint8_t) command, BCAP_getDefaultArgLen(command));
20  }
21
22  uint8_t BCAP_getDefaultArgLen(APCommand_t command)
23  {
24    switch(command) {
25      case BCAP_REQ_SETDAC: return 0x08;
26      case BCAP_REQ_SETLED: return 0x0C;
27      case BCAP_REQ_QUIT:   return 0x00;
28      default:              return 0;
29    }
30  }
```

C.3.4. BC_ActuatorProtocol.h

```
1   /*************************************************************
2    * Functions and definitions for the actuator protocol.
3    *
4    * File:    BC_ActuatorProtocol.h
5    * Version: 0.2
6    * Date:    2016-04-14
7    * Author:  Jan Griessbach
8    *
9    * Changes: 0.2: NR added command for ´LEDs and renamed from VRefProtocol
10   *               in ActuatorProtocol.
11   *          0.1: first version
12   *
13   *************************************************************/
14
15  #ifndef BC_ACTUATORPROTOCOL_H_
16  #define BC_ACTUATORPROTOCOL_H_
17
18  #include "BC_Protocol.h"
19
20  /* Data types */
21  typedef BCFrame_t APFrame_t;
22
23  typedef enum enum_bcap_command {
24    BCAP_REQ_SETDAC    = 0x01,
```

```
25    BCAP_REQ_SETLED    = 0x02,
26    BCAP_REQ_QUIT      = 0x7F
27  } APCommand_t;
28
29  /* argument offsets */
30  #define BCAP_REQ_SETDAC_OFFSET_VALUE     0x00
31  #define BCAP_REQ_LED_RED_OFFSET_VALUE    0x00
32  #define BCAP_REQ_LED_GREEN_OFFSET_VALUE  (sizeof(float))
33  #define BCAP_REQ_LED_BLUE_OFFSET_VALUE   (2*sizeof(float))
34
35  /* function prototypes */
36  void     BCAP_initializeFrame(APFrame_t *f, APCommand_t command);
37  uint8_t  BCAP_getDefaultArgLen(APCommand_t command);
38
39  #endif /* BC_ACTUATORPROTOCOL_H_ */
```

C.3.5. BC_MeasurementProtocol.c

```
1   /*****************************************************************
2    * Functions and definitions for the Measurement protocol.
3    *
4    * File:    BC_MeasurementProtocol.c
5    * Version: 0.2
6    * Date:    2016-02-12
7    * Author:  Jan Griessbach, Nico Rieckmann
8    *
9    * Changes: 0.2: NR added MAKROS for Control response argument values.
10   *          0.1: JG first version
11   *
12   *****************************************************************/
13
14  #include "BC_MeasurementProtocol.h"
15
16  void BCMP_initializeFrame(MPFrame_t *f, MPCommand_t command)
17  {
18    BC_initializeFrame(f, (uint8_t) command, BCMP_getDefaultArgLen(command));
19  }
20
21  uint8_t BCMP_getDefaultArgLen(MPCommand_t command)
22  {
23    switch(command) {
24      case BCMP_REQ_GETVALUE:         return 0x00;
25      case BCMP_REQ_STARTCONTINUOUS:  return 0x04;
26      case BCMP_REQ_STOPCONTINUOUS:   return 0x00;
27      case BCMP_REQ_GETCONTROL:       return 0x00;
28      case BCMP_REQ_RETCONTROL:       return 0x00;
29      case BCMP_REQ_QUIT:             return 0x00;
30      case BCMP_RES_VALUE:            return 0x10;
31      case BCMP_RES_CONTROL:          return 0x01;
32      default:                        return 0;
33    }
34  }
```

C.3.6. BC_MeasurementProtocol.h

```
1   /*****************************************************************
2    * Functions and definitions for the Measurement protocol.
3    *
4    * File:    BC_MeasurementProtocol.h
5    * Version: 0.2
6    * Date:    2016-02-12
7    * Author:  Jan Griessbach, Nico Rieckmann
8    *
9    * Changes: 0.2: NR added MAKROS for Control response argument values.
10   *          0.1: JG first version
11   *
12   *****************************************************************/
13
14  #ifndef BC_MEASUREMENTPROTOCOL_H_
15  #define BC_MEASUREMENTPROTOCOL_H_
16
17  #include "BC_Protocol.h"
18
```

```
19  /* Data types */
20  typedef BCFrame_t MPFrame_t;
21
22  typedef enum enum_bcmp_command {
23    BCMP_REQ_GETVALUE          = 0x01,
24    BCMP_REQ_STARTCONTINUOUS = 0x02,
25    BCMP_REQ_STOPCONTINUOUS   = 0x03,
26    BCMP_REQ_GETCONTROL        = 0x04,
27    BCMP_REQ_RETCONTROL        = 0x05,
28    BCMP_REQ_QUIT              = 0x7F,
29    BCMP_RES_VALUE             = 0x81,
30    BCMP_RES_CONTROL           = 0x84
31  } MPCommand_t;
32
33  /* argument offsets */
34  #define BCMP_REQ_STARTCONTINUOUS_OFFSET_INTERVAL 0x00
35  #define BCMP_RES_VALUE_OFFSET_TIMESTAMP          0x00
36  #define BCMP_RES_VALUE_OFFSET_VALUE              (sizeof(uint64_t))
37  #define BCMP_RES_CONTROL_OFFSET_CONTROL          0x00
38
39  /* Control response argument values */
40  #define BCMP_RES_CONTROL_ARGVAL_CONTROL_DENIAL   0x00
41  #define BCMP_RES_CONTROL_ARGVAL_CONTROL_GRANT    0x01
42  #define BCMP_RES_CONTROL_ARGVAL_CONTROL_RELEASE  0x02
43
44  /* function prototypes */
45  void     BCMP_initializeFrame(MPFrame_t *f, MPCommand_t command);
46  uint8_t  BCMP_getDefaultArgLen(MPCommand_t command);
47
48  #endif /* BC_MEASUREMENTPROTOCOL_H_ */
```

C.3.7. BC_RelayProtocol.c

```
1   /*****************************************************************
2    * Functions and definitions for the Relay protocol.
3    *
4    * File:    BC_RelayProtocol.c
5    * Version: 0.2
6    * Date:    2016-02-28
7    * Author:  Jan Griessbach
8    *
9    * Changes: 0.2: NR added RPArgument_t enum type.
10   *          0.1: first version.
11   *
12   *****************************************************************/
13
14
15  #include "BC_RelayProtocol.h"
16
17
18  void BCRP_initializeFrame(RPFrame_t *f, RPCommand_t command)
19  {
20    BC_initializeFrame(f, (uint8_t) command, BCRP_getDefaultArgLen(command));
21  }
22
23  uint8_t BCRP_getDefaultArgLen(RPCommand_t command)
24  {
25    switch(command) {
26      case BCRP_REQ_SETREGISTER: return 0x01;
27      case BCRP_REQ_SETRELAY:    return 0x01;
28      case BCRP_REQ_DELRELAY:    return 0x01;
29      case BCRP_REQ_TOGRELAY:    return 0x01;
30      case BCRP_REQ_QUIT:        return 0x00;
31      default:                   return 0;
32    }
33  }
```

C.3.8. BC_RelayProtocol.h

```
1   /*****************************************************************
2    * Functions and definitions for the Relay protocol.
3    *
```

```
4    * File:    BC_RelayProtocol.h
5    * Version: 0.2
6    * Date:    2016-02-28
7    * Author:  Jan Griessbach
8    *
9    * Changes: 0.2: NR added RPArgument_t enum type.
10   *          0.1: first version.
11   *
12   ************************************************************/
13
14   #ifndef BC_RELAYPROTOCOL_H_
15   #define BC_RELAYPROTOCOL_H_
16
17   #include "BC_Protocol.h"
18
19   /* Data types */
20   typedef BCFrame_t RPFrame_t;
21
22   typedef enum enum_bcrp_command {
23     BCRP_REQ_SETREGISTER = 0x01,
24     BCRP_REQ_SETRELAY    = 0x02,
25     BCRP_REQ_DELRELAY    = 0x03,
26     BCRP_REQ_TOGRELAY    = 0x04,
27     BCRP_REQ_QUIT        = 0x7F
28   } RPCommand_t;
29
30   typedef enum enum_bcrp_argument {
31     BCRP_RELAY_VN  = 0x01,
32     BCRP_RELAY_VP  = 0x02,
33     BCRP_RELAY_I   = 0x04,
34     BCRP_RELAY_ALL = BCRP_RELAY_VN | BCRP_RELAY_VP | BCRP_RELAY_I
35   } RPArgument_t;
36
37   /* argument offsets */
38   #define BCRP_REQ_SETREGISTER_OFFSET_PATTERN 0x00
39   #define BCRP_REQ_SETRELAY_OFFSET_RELAYNR    0x00
40   #define BCRP_REQ_DELRELAY_OFFSET_RELAYNR    0x00
41   #define BCRP_REQ_TOGRELAY_OFFSET_RELAYNR    0x00
42
43   /* function prototypes */
44   void     BCRP_initializeFrame(RPFrame_t *f, RPCommand_t command);
45   uint8_t  BCRP_getDefaultArgLen(RPCommand_t command);
46
47   #endif /* BC_RELAYPROTOCOL_H_ */
```

C.4. BC_HW_Lib

C.4.1. BC_HW_Initialization.c

```
1    /************************************************************
2    * GPIO and SPI initialization functions for the Batcycle hardware.
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_Initialization.c
7    * Version: 0.3
8    * Date:    2016-03-09
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.3: NR added case handling of an existing uninitialized
12   *               semaphore in BCHW_initializeAccessBarrier() function.
13   *          0.2: NR added unnamed semaphores for multi process
14   *               access on GPIO and SPI pins.
15   *          0.1: NR first version
16   *
17   ************************************************************/
18
19   #include "BC_HW_Initialization.h"
20
21   sem_t         *sem_SPI = NULL;
22   sem_t         *sem_Relays = NULL;
23   sig_atomic_t serverFlag = false;
24   //sig_atomic_t spiInitFlag = false;
25
26   /** This function maps shared memory into own space and initialize a un-
```

```
27      named semaphore to handle access on SPI and GPIO from various processes.
28      The first process who calls this function creates the memfile on filesystem
29      and initialize a semaphore. All other processes becomes clients on the semaphore.
30      @param1: semaphore pointer what is the access barrier
31      @param2: init value of semaphore
32      @param3: file pointer for shared memory object
33      return: 1 on success and -1 on failure
34  */
35  int BCHW_initializeAccessBarrier(sem_t **semaphore, unsigned int initval, char *memfile) {
36      int fd;
37      void *addr_ptr;
38      int retval;
39
40      /*when this process calls at first of all others this function, he will be the server
41        who initialize and destroy the semaphores at program end*/
42      fd = shm_open(memfile, O_RDWR | O_CREAT | O_EXCL, MODE);
43      if(fd > 0 && errno != EEXIST) {
44          serverFlag = true;
45          //truncate shared memory object
46          retval = ftruncate(fd, sizeof(sem_t*));
47          if(retval < 0) {
48              perror("Failure in ftruncate()");
49              shm_unlink(memfile);
50              close(fd);
51              return -1;
52          }
53          //map the shared memory object into own process memory space
54          addr_ptr = mmap(0, sizeof(sem_t*), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
55          if(addr_ptr < 0) {
56              perror("Failure in mmap()");
57              shm_unlink(memfile);
58              close(fd);
59              return -1;
60          }
61          //initialize semaphore by server
62          close(fd);
63          *semaphore = (sem_t*) addr_ptr;
64          retval = sem_init(*semaphore, PSHARED, initval);   //PSHARED for share between processes
65          if(retval < 0) {
66              perror("Failure in sem_init()");
67              shm_unlink(memfile);
68              return -1;
69          }
70      }
71      //when an other process called this function first, this process must be a client on the shared semaphores
72      else {
73          fd = shm_open(memfile, O_RDWR, 0);
74          if(fd < 0) {
75              perror("Failure in shm_open()");
76              return -1;
77          }
78          //map the shared memory object into own process memory space
79          addr_ptr = mmap(0, sizeof(sem_t*), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
80          if(addr_ptr < 0) {
81              perror("Failure in mmap()");
82              close(fd);
83              return -1;
84          }
85          //initialize semaphore by server
86          close(fd);
87          *semaphore = (sem_t*) addr_ptr;
88
89          /*semaphore test to prevent a possible segmentation fault.
90          only when the shared memfile wasn't unlinked by previous program execution*/
91          if(sem_trywait(*semaphore) < 0) {
92              perror("Failure at semaphore");
93              shm_unlink(memfile);
94              return -1;
95              /*if(errno == EINVAL) {
96                  perror("Error occurred by an invalid semaphore address");
97                  shm_unlink(memfile);
98                  return -1;
99              }  */
100         }
101         sem_post(*semaphore);
102     }
103     return 1;   //when success
104 }
105
106 /** This function demaps shared memory in own space. If the current process had called
107     BCHW_initializeAccessBarrier at first from all parallel processes, he is the
108     only one who can destroy the semphore.
109     @param1: semaphore pointer what is the access barrier
110     @param2: file pointer for shared memory object
111     return: 1 on success and -1 on failure
```

```
112 */
113 int BCHW_releaseAccessBarrier(sem_t **semaphore, char *memfile) {
114     int retval  = 1;
115 
116     if(serverFlag == true) {
117         //this region is only for the first process who sets the serverFlag
118         retval |= sem_destroy(*semaphore);
119         if(retval < 0)
120             perror("Failure in sem_destroy()");
121     }
122 
123     //unmap bounded memory
124     retval |= munmap((void*) *semaphore, sizeof(sem_t*));
125     if(retval < 0)
126         perror("Failure in munmap()");
127 
128     //remove shared memory file only by process who sets the serverFlag
129     if(serverFlag == true) {
130         retval |= shm_unlink(memfile);
131         if(retval < 0)
132             perror("Failure in shm_unlink()");
133     }
134     return retval;
135 }
136 
137 int BCHW_initializeAll(void) {
138     //initialize the library to allocate BCM 2835 registers by open "/dev/mem"
139     if(bcm2835_init() <= 0)
140         return -1;
141 
142     return 1;
143 }
144 
145 void BCHW_initializeSPI(void) {
146     if(serverFlag) {
147         bcm2835_spi_begin();        //init interface
148         bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);    //sends MSB first
149         bcm2835_spi_setDataMode(BCM2835_SPI_MODE1); //data detection by falling edge of sclk
150         bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_65536);   //set clock to 3.8kHz
151         //bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_64);   //set clock to 3.9MHz
152         //bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);   //set clock to 976,56kHz
153         bcm2835_spi_chipSelect(BCM2835_SPI_CS0);            //set CS0 to chipselect pin
154         bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);    //activity level of CS pin is low, idle high
155     }
156 }
157 
158 void BCHW_releaseSPI(void) {
159     /*SPI0 pins P1-19 (MOSI), P1-21 (MISO), P1-23 (CLK), P1-24 (CE0),
160     P1-26 (CE1) are returned to their default INPUT behaviour*/
161     if(serverFlag == true)
162         bcm2835_spi_end();
163 }
164 
165 int BCHW_releaseAll(void) {
166     //Close the library to deallocate BCM 2835 registers and close /dev/mem
167     if(bcm2835_close() <= 0)
168         return -1;
169     return 1;
170     /*//process may only destroy a semaphore he has also used.
171     sem_getvalue(sem_Relays, &val);
172     if(val >= 0)
173         BCHW_releaseAccessBarrier(&sem_Relays, SHARED_MEM_FILE_RELAYS);
174     sem_getvalue(sem_SPI, &val);
175     if(val >= 0)
176         BCHW_releaseAccessBarrier(&sem_SPI, SHARED_MEM_FILE_SPI);
177     return 1;*/
178 }
```

C.4.2. BC_HW_Initialization.h

```
1 /*************************************************************
2  * GPIO and SPI initialization functions for the Batcycle hardware.
3  *
4  * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5  *
6  * File:    BC_HW_Initialization.h
7  * Version: 0.2
8  * Date:    2016-03-03
9  * Author:  Nico Rieckmann
```

```
10   *
11   * Changes: 0.2: NR added unnamed semaphores for multi process
12   *               access on GPIO and SPI pins.
13   *          0.1: NR first version
14   *
15   ************************************************************/
16
17  #ifndef BC_HW_Initialization_H_
18  #define BC_HW_Initialization_H_
19
20  #include <stdio.h>
21  #include <stdlib.h>
22  #include <bcm2835.h>
23  #include <unistd.h>
24  #include <stdbool.h>    //for boolean
25  #include <sys/types.h>
26  #include <sys/mman.h>
27  #include <semaphore.h>
28  #include <sys/stat.h> /* For mode constants */
29  #include <fcntl.h> /* For shm_open constants */
30  #include <signal.h>
31  #include <errno.h>
32  //#include "BC_HW_global.h"
33
34  /* GPIO access semaphores */
35  #define SHARED_MEM_FILE_SPI     "shmem_spi"
36  #define SHARED_MEM_FILE_RELAYS  "shmem_gio"
37  #define SEM_INITVAL_SPI         1
38  #define SEM_INITVAL_RELAYS      1
39  #define MODE                    (S_IRWXU | S_IRGRP | S_IXGRP | S_IROTH | S_IXOTH)
40  //#define MODE                    (S_IRWXU | S_IRWXG | S_IRWXO)
41  #define PSHARED                 1
42
43  /* global variables definition*/
44  extern sem_t        *sem_SPI;      //semaphore for spi pin access barrier
45  extern sem_t        *sem_Relays;   //semaphore for GPIO access barrier
46  extern sig_atomic_t serverFlag;    //identify flag, semaphore-server becomes the process whos started first
47
48  /* GPIO and SPI Initialization*/
49  int  BCHW_initializeAccessBarrier(sem_t **semaphore, unsigned int startval, char *memfile);
50  int  BCHW_releaseAccessBarrier(sem_t **semaphore, char *memfile);
51  int  BCHW_initializeAll(void);
52  void BCHW_initializeSPI(void);
53  void BCHW_releaseSPI(void);
54  int  BCHW_releaseAll(void);
55
56
57  #endif // BC_HW_Initialization_H_
```

C.4.3. BC_HW_ADC.c

```
1   /************************************************************
2    * ADC functions for the AD7691(16bit) ADC on Batcycle hardware.
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_ADC.c
7    * Version: 0.3
8    * Date:    2016-03-03
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.3: NR added unnamed semaphores for multi process
12   *               access on GPIO and SPI pins.
13   *          0.2: NR added sign bit shift up from bit17 to bit31 for
14   *               negative range and fixed the misinterpretation problem.
15   *          0.1: NR first version
16   *
17   ************************************************************/
18
19  #include "BC_HW_ADC.h"
20
21  int BCHW_initializeADCgpio(uint8_t chipsel, uint8_t convpin) {
22      int retval;
23
24      //Hardware Initializations
25      retval = BCHW_initializeAll();
26      if(retval < 0)
27          return -1;
28
```

```
29      //allocate global semaphore into this process
30      retval = BCHW_initializeAccessBarrier(&sem_SPI, SEM_INITVAL_SPI, SHARED_MEM_FILE_SPI);
31      if(retval < 0)
32          return -1;
33
34      //enter critical region
35      sem_wait(sem_SPI);          //decrement sem_SPI
36
37      BCHW_initializeSPI();
38      //set GPIO pins to output and CSpins to HIGH
39      bcm2835_gpio_fsel(chipsel, BCM2835_GPIO_FSEL_OUTP);    //chipselect pin to output
40      bcm2835_gpio_fsel(convpin, BCM2835_GPIO_FSEL_OUTP);    //convert pin to output
41      bcm2835_gpio_write(chipsel, HIGH);      //set HIGH
42      bcm2835_gpio_write(convpin, HIGH);      //set HIGH
43      //exit critical region
44      sem_post(sem_SPI);      //increment sem_SPI
45      return 1;
46  }
47
48  int BCHW_releaseADCgpio(void) {
49      int retval = 1;
50
51      BCHW_releaseSPI();
52      retval |= BCHW_releaseAll();
53      retval |= BCHW_releaseAccessBarrier(&sem_SPI, SHARED_MEM_FILE_SPI);
54      return retval;
55  }
56
57  /** This function reads an value out of adc.
58      @param1: RP2 GPIOpin for chipselect of spi-member
59      @param2: convertpin from adc which is connected to RPI2 GPIO
60      return: readed adc value
61  */
62  int32_t BCHW_readFromADC(uint8_t chipsel, uint8_t convpin) {
63      int i;
64      uint8_t dummy = 0;
65      int32_t temp = 0;
66      union inbuffer_u buf;
67      double doub;            //kann weg
68
69      //Hardware Initializations
70      buf.data = 0;     //reset union
71      //enter critical region
72      sem_wait(sem_SPI);          //decrement sem_SPI
73      //activate ADC
74      bcm2835_gpio_write(convpin, HIGH);      //starts adc acquisition
75
76      bcm2835_delay(WAITING_TIME_US);            //wait at least Â3,7µs
77      bcm2835_gpio_write(chipsel, LOW);       //pulls chipselect pin to low
78      //receive data
79      for( i=2; i>=0; i--) {
80          buf.byte[i] = bcm2835_spi_transfer(dummy);
81      }
82      //deactivate ADC
83      bcm2835_gpio_write(chipsel, HIGH);      //pulls chipselect pin to high
84      bcm2835_gpio_write(convpin, LOW);       //adc acquisition ends
85      //exit critical region
86      sem_post(sem_SPI);      //increment sem_SPI
87
88      //bcm2835 is in little-Endian, adc sends MSB first, bit0 is LSB and bit17 is signbit
89      buf.data = buf.data >> 6; //shift left to usefully 18bit
90      if(buf.data & ADC_SIGNBIT_MASK) { //check sign in bit17 (if negative value)
91          buf.data &= ~ADC_SIGNBIT_MASK;    //delete sign in bit17
92          //buf.data |= 0x80000000;     //zahl negativ machen
93          //buf.data = -buf.data;   //TODO falsch, da ADC Wert bereits im 2er Komplement ist.
94
95          temp = -ADC_RANGE;  //converting 2ers complement value from 18bit to 32bit integer
96          temp |= buf.data;
97          buf.data = temp;
98      }
99      doub = BCHW_parseToVoltage(buf.data);
100     return buf.data;
101 }
102
103 double BCHW_parseToCurrent(uint32_t value) {
104     //return (double) (((value - 65536.0) / 52427)) * 1000; //formel mit Uref=2.5V
105     //return (double) ((value - 54613.0) / 43691) * 1000;  //formel mit Uref=3V
106     return (double) ((value - ADC_I_ZERO_OFFSET) * ADC_I_GRADIENT) / FAC_MICRO;
107 }
108
109 double BCHW_parseToVoltage(int32_t value) {
110      //return (double) ( value * (2.5*2) ) / 131071;   //D=U_in/(2.5V/2^17) Thorstens formel
111      //return (double) 2 * ( value * (2.5 / 131072));   //D=U_in/(2.5V/2^17)
112      //return (double) ( value * (ADC_VREF / ADC_RANGE) / ADC_V_GAIN);   //D=U_in/(2.5V/2^17)   //TODO neues Uref 3.0V statt 2.5V
113      return (double) ((value * ADC_ULSB) / ADC_V_GAIN);   //D=U_in/(2.5V/2^17)  //Uref 3.0V instead 2.5V
```

```
114 }
```

C.4.4. BC_HW_ADC.h

```
1  /************************************************************
2   * ADC functions for the AD7691(16bit) ADC on Batcycle hardware.
3   *
4   * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5   *
6   * File:    BC_HW_ADC.h
7   * Version: 0.3
8   * Date:    2016-03-03
9   * Author:  Nico Rieckmann
10  *
11  * Changes: 0.3: NR added unnamed semaphores for multi process
12  *               access on GPIO and SPI pins.
13  *          0.2: NR added sign bit shift up from bit17 to bit31 for
14  *               negative range and fixed the misinterpretation problem.
15  *          0.1: NR first version
16  *
17  ************************************************************/
18
19 #ifndef BC_HW_ADC_H_
20 #define BC_HW_ADC_H_
21
22 #include <stdio.h>
23 #include <stdlib.h>
24 #include "BC_HW_Initialization.h"
25
26 /* general ADC constants */
27 #define WAITING_TIME_US     30      //time in Âµs to wait after adc acquisition start
28 #define ADC_SIGNBIT_MASK    0x00020000          //bit mask of sign bit to read data out of adc
29 #define ADC_VREF            3.0                 //Uref voltage
30 #define ADC_RANGE           131072              //17bit ADC with 2^17
31 #define ADC_FULL_SCALE      (ADC_RANGE - 1)             //highest decimal value of adc
32 #define ADC_LOW_SCALE       0                           //lowest decimal value of adc
33 #define ADC_ULSB            ((double) (ADC_VREF / ADC_RANGE))      //Ulsb
34
35 /* ADC_V constants */
36 #define ADC_V_GAIN          0.5             //Gain of Instr.Amp which is connected at ADC_V input
37
38 /* ADC_I constants */
39 //#define ADC_I_GRADIENT    (1000.0/52427)        //1mA/ADC_I decimal) factor. ADC_I output is 52427 at 1mA current (Uref=2.5V)
40 #define ADC_I_GRADIENT      (1000.0/43691)        //1mA/ADC_I decimal) factor. ADC_I output is 43691 at 1mA current (Uref=3V)
41 #define ADC_I_ZERO_OFFSET   ((int) (1.25 / ADC_ULSB) + 0.5)   //decimal value of the 1.25V Instr.Amp offset to lift the point of origin
42 #define FAC_MICRO           1E6                 //conversion factor from Â[µA] to [A]
43
44 /* SPI Chipselects (CS) */
45 #define CS_PIN_ADC_I        RPI_BPLUS_GPIO_J8_11    // SPI Chipselect (CS) at GPIO27 (pin11)
46 #define CS_PIN_ADC_V        RPI_BPLUS_GPIO_J8_13    // CS for voltage ADC at GPIO17 (pin13)
47
48 /* ADC's convert trigger */
49 #define ADC_CONV_PIN        RPI_BPLUS_GPIO_J8_16    // Convert trigger is at GPIO23 (pin16)
50
51 /* transfer buffer struct */
52 union inbuffer_u {
53     int32_t data;
54     uint8_t byte[4];
55 };
56
57 /* function prototypes */
58 int     BCHW_initializeADCgpio(uint8_t chipsel, uint8_t convpin);
59 int     BCHW_releaseADCgpio(void);
60 int32_t BCHW_readFromADC(uint8_t chipsel, uint8_t convpin);
61 double  BCHW_parseToCurrent(uint32_t value);
62 double  BCHW_parseToVoltage(int32_t value);
63
64 #endif /* BC_HW_ADC_H_ */
```

C.4.5. BC_HW_DAC.c

```
1  /*************************************************************
2   * DAC functions for the AD5680(16bit) DAC on Batcycle hardware.
3   *
4   * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5   *
6   * File:    BC_HW_DAC.c
7   * Version: 0.2
8   * Date:    2016-03-03
9   * Author:  Nico Rieckmann
10  *
11  * Changes: 0.2: NR added unnamed semaphores for multi process
12  *               access on GPIO and SPI pins.
13  *          0.1: NR first version
14  *
15  *************************************************************/
16
17 #include "BC_HW_DAC.h"
18
19 int BCHW_initializeDACgpio(uint8_t chipsel) {
20     int retval;
21
22     //Hardware Initializations
23     retval = BCHW_initializeAll();
24     if(retval < 0)
25         return -1;
26
27     //allocate global semaphore into this process
28     retval = BCHW_initializeAccessBarrier(&sem_SPI, SEM_INITVAL_SPI, SHARED_MEM_FILE_SPI);
29     if(retval < 0)
30         return -1;
31
32     //enter critical region
33     sem_wait(sem_SPI);          //decrement sem_SPI
34
35     BCHW_initializeSPI();
36     //set GPIO pins to output and CSpins to HIGH
37     bcm2835_gpio_fsel(chipsel, BCM2835_GPIO_FSEL_OUTP);    //CS_PIN_DAC_I to output
38     bcm2835_gpio_write(chipsel, HIGH);     //set HIGH
39     //exit critical region
40     sem_post(sem_SPI);      //increment sem_SPI
41     return 1;
42 }
43
44 int BCHW_releaseDACgpio(void) {
45     int retval = 1;
46
47     BCHW_releaseSPI();
48     retval |= BCHW_releaseAll();
49     retval |= BCHW_releaseAccessBarrier(&sem_SPI, SHARED_MEM_FILE_SPI);
50     return retval;
51 }
52
53 /** This function send an value to the 16bit DAC.
54     @param1: value from D=0...131071 (Uout=D/DAC_GAIN*Ulsb)  Ulsb=2,5V/2^16=38µV
55     @param2: RPI2 GPIOpin for chipselect of dac in spi-bus
56     return: void because if an transmission failure occured, data will not repeated.
57 */
58 void BCHW_sendToDAC(uint8_t chipsel, uint32_t value) {
59     int i;
60     union outbuffer_u buf;
61
62     //prepare value before sending
63     value = value/DAC_GAIN; //value must divide by 2, cause amp on hardware multiply by 2
64     if((value < DAC_LOW_SCALE) || (value > DAC_FULL_SCALE))
65         return;
66
67     buf.data = value << 4;    //bit0 (LSB) until bit3 will masked out
68     buf.data &= DAC_MASK;       //mask "Tabu-Bits" out
69
70     //enter critical region
71     sem_wait(sem_SPI);          //decrement sem_SPI
72     //activate DAC
73     bcm2835_gpio_write(chipsel, LOW);
74     //transmit data
75     for( i=2; i>=0; i--) {
76         uint8_t retval = 0;
77
78         retval = bcm2835_spi_transfer(buf.byte[i]);
79         if(retval < 0)                  //if transmitting failure occurred  //TODO vllt sicherheitsabfrage wieder entfernen
80             fprintf(stderr, "Error occurred in setDACval()");
81     }
82     //deactivate DAC
83     bcm2835_gpio_write(chipsel, HIGH);   //pulls chipselect pin to high
84
85     //exit critical region
```

```
86      sem_post(sem_SPI);      //increment sem_SPI
87  }
88
89  int32_t  BCHW_parseToDACvalue(double voltage) {
90      //gain error correction
91      return (int32_t) ((voltage - DAC_GAIN_CORRECTION(voltage)) / (DAC_VREF / DAC_RANGE)) + 0.5;
92
93      //without GainError correction
94      //return (int32_t) (voltage / (DAC_VREF / DAC_RANGE)) + 0.5;
95  }
```

C.4.6. BC_HW_DAC.h

```
1   /************************************************************
2    * DAC functions for the AD5680(16bit) DAC on Batcycle hardware.
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_DAC.h
7    * Version: 0.2
8    * Date:    2016-03-03
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.2: NR added unnamed semaphores for multi process
12   *               access on GPIO and SPI pins.
13   *          0.1: NR first version
14   *
15   ************************************************************/
16
17  #ifndef BC_HW_DAC_H_
18  #define BC_HW_DAC_H_
19
20  #include <stdio.h>
21  #include <stdlib.h>
22  #include "BC_HW_Initialization.h"
23
24  /* DAC constants */
25  #define DAC_MASK         0x000FFFF0
26  #define DAC_GAIN         2
27  #define DAC_VREF         3.0             //reference voltage
28  #define DAC_RANGE        65536           //16 bit DAC with 2^16
29  #define DAC_FULL_SCALE   (DAC_RANGE - 1)
30  #define DAC_LOW_SCALE    0
31  #define DAC_ULSB         (DAC_VREF*DAC_GAIN/DAC_RANGE)
32
33  /* Gain error correction */                          //measured with DAC LinearityTet
34  #define ERROR_GRADIENT               -0.00218        //m = (y2-y1)/(x2-x1)
35  #define ZERO_OFFSET                  0.005363        //b = offset when x=0
36  #define DAC_GAIN_CORRECTION(X)       (ERROR_GRADIENT * X + ZERO_OFFSET)   //Gain error correction with y=m*x+b
37
38  /* SPI Chipselects (CS) */
39  #define CS_PIN_DAC       RPI_BPLUS_GPIO_J8_15    //CS for DAC at GPIO22 (pin15)
40
41  /* transfer buffer struct */
42  union outbuffer_u {
43      uint32_t data;
44      uint8_t byte[3];
45  };
46
47  /* function prototypes */
48  int      BCHW_initializeDACgpio(uint8_t chipsel);
49  int      BCHW_releaseDACgpio(void);
50  void     BCHW_sendToDAC(uint8_t chipsel, uint32_t value);
51  //uint32_t BCHW_parseToCurrent(double value);
52  int32_t  BCHW_parseToDACvalue(double voltage);
53
54  #endif /* BC_HW_DAC_H_ */
```

C.4.7. BC_HW_Relay.c

```
1   /************************************************************
2    * Relay functions for the charging circuit breakers on Batcycle hardware.
```

```
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_Relay.c
7    * Version: 0.4
8    * Date:    2016-03-07
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.4: NR implemented BCHW_setRelayPattern() function.
12   *          0.3: NR added unnamed semaphores for multi process
13   *               access on GPIO and SPI pins.
14   *          0.2: add functions: BCHW_setRelayPattern(),
15   *               BCHW_getRelayPosition(), BCHW_toggleRelay().
16   *          0.1: NR first version
17   *
18   ************************************************************/
19
20   #include "BC_HW_Relay.h"
21
22   int BCHW_initializeRelayGPIO(void) {
23       int retval;
24
25       //hardware initialization
26       retval = BCHW_initializeAll();
27       if(retval < 0)
28           return -1;
29
30       //allocate global semaphore into this process
31       retval = BCHW_initializeAccessBarrier(&sem_Relays, SEM_INITVAL_RELAYS, SHARED_MEM_FILE_RELAYS);
32       if(retval < 0)
33           return -1;
34
35       //enter critical region
36       sem_wait(sem_Relays);   //decrement semaphore and engage pins
37       bcm2835_gpio_fsel(PIN_RELAYS_VN, BCM2835_GPIO_FSEL_OUTP);
38       bcm2835_gpio_fsel(PIN_RELAYS_VP, BCM2835_GPIO_FSEL_OUTP);
39       bcm2835_gpio_fsel(PIN_RELAYS_I, BCM2835_GPIO_FSEL_OUTP);
40       //BCHW_setRelayPattern(0x00);
41       //sem_post(sem_Relays);   //left region, increment semaphore
42       return 1;
43   }
44
45   int BCHW_releaseRelayGPIO(void) {
46       int retval = 1;
47
48       if(serverFlag)
49           BCHW_setRelayPattern(0x00);
50
51       //leave critical region
52       sem_post(sem_Relays);      //increment semaphore and give up gpio pins
53       retval |= BCHW_releaseAll();
54       retval |= BCHW_releaseAccessBarrier(&sem_Relays, SHARED_MEM_FILE_RELAYS);
55       return retval;
56   }
57
58   void BCHW_setRelayPattern(uint8_t gpioPattern) {
59       uint32_t value = 0;
60
61       value |= (((gpioPattern & 0x1) > 0) << PIN_RELAYS_VN);
62       value |= (((gpioPattern & 0x2) > 0) << PIN_RELAYS_VP);
63       value |= (((gpioPattern & 0x4) > 0) << PIN_RELAYS_I);
64       #ifdef DEBUG
65       printf("value: %x\n", value);
66       #endif
67       bcm2835_gpio_write_mask(value, PIN_MASK_ALL_RELAYS);
68   }
69
70   void BCHW_openRelay(uint32_t gpioPin) {
71       bcm2835_gpio_write(gpioPin, LOW);       //open relay
72       bcm2835_delayMicroseconds(RELAY_SWITCH_TIME_MS);   //setting time
73   }
74
75   void BCHW_closeRelay(uint32_t gpioPin) {
76       bcm2835_gpio_write(gpioPin, HIGH);      //close relay
77       bcm2835_delayMicroseconds(RELAY_SWITCH_TIME_MS);   //setting time
78   }
79
80   uint8_t BCHW_getRelayPosition(uint32_t gpioPin) {
81       if(bcm2835_gpio_lev(gpioPin) == HIGH)       //if gpioPin is High, relay is closed
82           return 0x1;
83       else if(bcm2835_gpio_lev(gpioPin) == LOW)   //if gpioPin is LOW, relay is opened
84           return 0x0;
85       else
86           return -1;
87   }
```

```
88
89  void BCHW_toggleRelay(uint32_t gpioPin) {
90      if(bcm2835_gpio_lev(gpioPin) == HIGH) { //if Relay closed
91          bcm2835_gpio_write(gpioPin, LOW);  //open Relay
92          bcm2835_delayMicroseconds(RELAY_SWITCH_TIME_MS);   //setting time
93      }
94      else if(bcm2835_gpio_lev(gpioPin) == LOW){  //if Relay open
95          bcm2835_gpio_write(gpioPin, HIGH);  //close Relay
96          bcm2835_delayMicroseconds(RELAY_SWITCH_TIME_MS);   //setting time
97      }
98  }
```

C.4.8. BC_HW_Relay.h

```
1   /************************************************************
2    * Relay functions for the charging circuit breakers on Batcycle hardware
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_Relay.h
7    * Version: 0.4
8    * Date:    2016-03-07
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.4: NR implemented BCHW_setRelayPattern() function.
12   *          0.3: NR added unnamed semaphores for multi process
13   *               access on GPIO and SPI pins.
14   *          0.2: add functions: BCHW_setRelayPattern(),
15   *               BCHW_getRelayPosition(), BCHW_toggleRelay().
16   *          0.1: NR first version
17   *
18   ************************************************************/
19
20  #ifndef BC_HW_RELAY_H_
21  #define BC_HW_RELAY_H_
22
23  #include <stdio.h>
24  #include <stdlib.h>
25  #include "BC_HW_Initialization.h"
26
27  /* constants */
28  #define RELAY_SWITCH_TIME_MS        200         //time after relay closed in msec
29
30  /* GPIO pins */
31  #define PIN_RELAYS_VN    RPI_BPLUS_GPIO_J8_36    //for relay at battery minus at GPIO16 (pin36)
32  #define PIN_RELAYS_VP    RPI_BPLUS_GPIO_J8_38    //for relay at battery plus at GPIO20 (pin38)
33  #define PIN_RELAYS_I     RPI_BPLUS_GPIO_J8_40    //for relay in current path at GPIO21 (pin40)
34  //mask of all relays on batcycle hardware
35  #define PIN_MASK_ALL_RELAYS  ((1<<PIN_RELAYS_VN) | (1<<PIN_RELAYS_VP) | (1<<PIN_RELAYS_I))
36
37  /* function prototypes */
38  int      BCHW_initializeRelayGPIO(void);
39  int      BCHW_releaseRelayGPIO(void);
40
41  void     BCHW_setRelayPattern(uint8_t gpioPattern);
42  void     BCHW_openRelay(uint32_t gpioPin);
43  void     BCHW_closeRelay(uint32_t gpioPin);
44  uint8_t  BCHW_getRelayPosition(uint32_t gpioPin);
45  void     BCHW_toggleRelay(uint32_t gpioPin);
46
47  #endif /* BC_HW_RELAY_H_ */
```

C.4.9. BC_HW_Temperature.c

```
1   /************************************************************
2    * Functions for the temperature sensor LM74(12bit) on Batcycle hardware.
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_Temperature.c
7    * Version: 0.1
8    * Date:    2016-04-30
```

```
 * Author:  Nico Rieckmann
 *
 * Changes: 0.1: NR first version
 *
 ************************************************************/

#include "BC_HW_Temperature.h"

int BCHW_initializeTempGPIO(uint8_t chipsel) {
    int retval;

    //Hardware Initializations
    retval = BCHW_initializeAll();
    if(retval < 0)
        return -1;

    //allocate global semaphore into this process
    retval = BCHW_initializeAccessBarrier(&sem_SPI, SEM_INITVAL_SPI, SHARED_MEM_FILE_SPI);
    if(retval < 0)
        return -1;

    //enter critical region
    sem_wait(sem_SPI);          //decrement sem_SPI

    BCHW_initializeSPI();
    //set GPIO pins to output and CSpins to HIGH
    bcm2835_gpio_fsel(chipsel, BCM2835_GPIO_FSEL_OUTP);    //chipselect pin to output
    bcm2835_gpio_write(chipsel, HIGH);     //set HIGH

    //exit critical region
    sem_post(sem_SPI);      //increment sem_SPI
    return 1;
}

int BCHW_releaseTempGPIO(void) {
    int retval = 1;

    BCHW_releaseSPI();
    retval |= BCHW_releaseAll();
    retval |= BCHW_releaseAccessBarrier(&sem_SPI, SHARED_MEM_FILE_SPI);
    return retval;
}

double BCHW_getTempValue(uint8_t chipsel) {
    int i;
    union inbuffer_u inbuf;
    union outbuffer_u outbuf;

    inbuf.data = 0;     //initialize buffers
    outbuf.data = MODE_CONTINOUS;

    //enter critical region
    sem_wait(sem_SPI);          //decrement sem_SPI
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);     //change data detection to raising edge of sclk
    bcm2835_gpio_write(chipsel, LOW);       //chipselect pin to low


    /*for( i=1; i>=0; i--){
        inbuf.byte[i] = bcm2835_spi_transfer(dummy);
    }
    bcm2835_spi_writenb((char*) &outbuf.byte, 2);   //activate sensor in continuous mode */

    for(i=3; i>=0; i--) {
        inbuf.byte[i] = bcm2835_spi_transfer(outbuf.byte[i]);
    }

    /*outbuf.byte = MODE_SHUTDOWN;        //deactivate sensor for better power consumption
    for( i=1; i>=0; i--){
    //TODO hier für muss MOSI und MISO am RPI zusammen geschlossen sein,
    was aber wegen der anderen Teilnehmer niocht gut ist
    inbuf.byte[i] = bcm2835_spi_transfer(0);
    }
    bcm2835_spi_writenb(&outbuf.byte, 2);   */

    bcm2835_gpio_write(chipsel, HIGH);       //chipselect pin to high
    BCHW_initializeSPI();   //reinitialize spi settings

    //exit critical region
    sem_post(sem_SPI);      //increment sem_SPI

    inbuf.data = inbuf.data >> 16;
    if(inbuf.data & TEMP_CONVFLAG) {    //when conversion flag was set
         inbuf.data = inbuf.data >> 3;   //shift left to useful 12 bit
         return inbuf.data * TEMP_LSB;
    }
```

```
94      else
95          fprintf(stderr, "Error occured: incomplete temperature conversion\n");
96      return -1000;
97  }
98
99  /*double BCHW_parseToCelsius(int16_t value) {
100     return value * TEMP_LSB;
101 }*/
```

C.4.10. BC_HW_Temperature.h

```
1   /************************************************************
2    * Functions for the temperature sensor LM74(12bit) on Batcycle hardware.
3    *
4    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
5    *
6    * File:    BC_HW_Temperature.h
7    * Version: 0.1
8    * Date:    2016-04-30
9    * Author:  Nico Rieckmann
10   *
11   * Changes: 0.1: NR first version
12   *
13   ************************************************************/
14
15  #ifndef BC_HW_TEMPERATURE_H_
16  #define BC_HW_TEMPERATURE_H_
17
18  #include <stdio.h>
19  #include <stdlib.h>
20  #include "BC_HW_Initialization.h"
21
22  /* sensor input register states */
23  #define MODE_CONTINOUS      0x00000000      //continious mode in configuration register
24  #define MODE_SHUTDOWN       0x000000FF      //sleep mode in configuration register
25
26  /* general constants */
27  #define TEMP_LSB            0.0625      //temperature LSB value in °C
28  #define TEMP_CONVFLAG       0x0004      //conversion flag mask bit
29  //#define TEMP_SIGNBIT_MASK   0x0004        //bit mask of sign bit to read data out of sensor
30
31  /* SPI Chipselects (CS) */
32  #define CS_PIN_TEMP         RPI_BPLUS_GPIO_J8_32    // CS for temperature sensor at GPIO12 (pin32)
33
34  /* transfer buffer structs */
35  union inbuffer_u {
36      int32_t data;
37      uint8_t byte[4];
38  };
39
40  union outbuffer_u {
41      int32_t data;
42      uint8_t byte[4];
43  };
44
45  /* function prototypes */
46  int     BCHW_initializeTempGPIO(uint8_t chipsel);
47  int     BCHW_releaseTempGPIO(void);
48  double  BCHW_getTempValue(uint8_t chipsel);
49  //double  BCHW_parseToCelsius(int16_t value);
50  //double  BCHW_parseToFahrenheit(int16_t value);
51
52  #endif /* BC_HW_TEMPERATURE_H_ */
```

C.4.11. BC_HW_Potentiometer.c

```
1   /************************************************************
2    * Functions for the digital potentiometer AD5270(10bit)/ AD5271(8bit)
3    * on the Batcycle hardware to trigger the RGB Led's.
4    *
5    * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
6    *
```

```
7    * File:    BC_HW_Potentiometer.c
8    * Version: 0.1
9    * Date:    2016-02-18
10   * Author:  Nico Rieckmann
11   *
12   * Changes: 0.1: NR first version
13   *
14   ************************************************************/
15
16   #include "BC_HW_Potentiometer.h"
17
18   int BCHW_initializePotiGPIO(uint8_t chipsel) {
19       int retval, i;
20       union outbuffer_u buf;
21
22       //Hardware Initializations
23       retval = BCHW_initializeAll();
24       if(retval < 0)
25           return -1;
26
27       //allocate global semaphore into this process
28       retval = BCHW_initializeAccessBarrier(&sem_SPI, SEM_INITVAL_SPI, SHARED_MEM_FILE_SPI);      //TODO evtl. extra poti semaphore
           anlegen
29       if(retval < 0)
30           return -1;
31
32       //enter critical region
33       sem_wait(sem_SPI);          //decrement sem_SPI                          //TODO evtl. extra poti semaphore anlegen
34
35       BCHW_initializeSPI();
36       //set GPIO pins to output and CSpins to HIGH
37       bcm2835_gpio_fsel(chipsel, BCM2835_GPIO_FSEL_OUTP);    //CS_PIN_LED to output
38       bcm2835_gpio_write(chipsel, HIGH);     //set HIGH
39
40       buf.data = 0x1C02;          //TODO was ist das? kann das weg?
41
42       bcm2835_gpio_write(chipsel, LOW);
43       bcm2835_delayMicroseconds(100);
44
45       for(i=1; i>=0; i--) {
46           bcm2835_spi_transfer(buf.byte[i]);
47       }
48
49       bcm2835_delayMicroseconds(100);
50       bcm2835_gpio_write(chipsel, HIGH);
51
52       //exit critical region
53       sem_post(sem_SPI);      //increment sem_SPI                               //TODO evtl. extra poti semaphore anlegen
54       return 1;
55   }
56
57   int BCHW_releasePotiGPIO(void) {
58       int retval = 1;
59
60       BCHW_releaseSPI();
61       retval |= BCHW_releaseAll();
62       retval |= BCHW_releaseAccessBarrier(&sem_SPI, SHARED_MEM_FILE_SPI);         //TODO evtl. extra poti semaphore anlegen
63       return retval;
64   }
65
66   void BCHW_sendToPoti(uint8_t chipsel, uint16_t data) {
67       int i;
68       union outbuffer_u buf;
69
70       //paste send command into bus word
71       buf.data |= 0x0400; //see poti datasheet
72
73       //enter critical region
74       sem_wait(sem_SPI);          //decrement sem_SPI
75
76       bcm2835_gpio_write(chipsel, LOW);
77       bcm2835_delayMicroseconds(100);
78
79       for( i=1; i>=0; i--) {
80           bcm2835_spi_transfer(buf.byte[i]);
81       }
82
83       bcm2835_delayMicroseconds(100);
84       bcm2835_gpio_write(chipsel, HIGH);
85
86       //exit critical region
87       sem_post(sem_SPI);      //increment sem_SPI
88   }
```

C.4.12. BC_HW_Potentiometer.h

```
/*************************************************************
 * Functions for the digital potentiometer AD5270(10bit)/ AD5271(8bit)
 * on the Batcycle hardware to trigger the RGB Led's.
 *
 * BCM2835 docu: http://www.airspayce.com/mikem/bcm2835/group__spi.html
 *
 * File:    BC_HW_Potentiometer.c
 * Version: 0.1
 * Date:    2016-02-18
 * Author:  Nico Rieckmann
 *
 * Changes: 0.1: NR first version
 *
 *************************************************************/

#ifndef BC_HW_POTENTIOMETER_H_
#define BC_HW_POTENTIOMETER_H_

#include <stdio.h>
#include <stdlib.h>
#include "BC_HW_Initialization.h"

/* constants */
#define AD5271      1           //for 8bit poti
#define AD5270      2           //for 10bit poti
#define POTI        AD5271

#if POTI == AD5271
    #define POTI_RANGE  256         //2^8
#elif POTI == AD5270
    #define POTI_RANGE  1024        //2^10
#endif // POTI
#define POTI_FULL_SCALE     (POTI_RANGE - 1)

//Chipselects SPI
#define CS_PIN_LED_R        RPI_BPLUS_GPIO_J8_33    //CS for red LED at GPIO13 (pin33)
#define CS_PIN_LED_G        RPI_BPLUS_GPIO_J8_35    //CS for green LED at GPIO19 (pin35)
#define CS_PIN_LED_B        RPI_BPLUS_GPIO_J8_37    //CS for blue LED at GPIO26 (pin37)

#define MAX_MEMBER          3

#define COMMAND_SEND        0x0400      //send command for control bit register

/* transfer buffer struct */
union outbuffer_u {
    uint16_t data;
    uint8_t byte[2];
};

/* function prototypes */
int     BCHW_initializePotiGPIO(uint8_t chipsel);
int     BCHW_releasePotiGPIO(void);
void    BCHW_sendToPoti(uint8_t chipsel, uint16_t data);

#endif // BC_HW_POTENTIOMETER_H_
```

C.5. BC_DMM4020Reader_SP

C.5.1. tektronix.c

```
/*************************************************************
 * hw_implementation part for Tektronix DMM4020
 *
 * File:    tektronix.c
 * Version: 0.1
 * Date:    2016-02-25
 * Author:  Jan Griessbach
 *
 * Changes: 0.1: JG first version
 *
 *************************************************************/

```

```
13  #include "tektronix.h"
14  #include "BC_HW_Implementation.h"
15  #include <stdio.h>
16  #include <stdlib.h>
17  #include <unistd.h>
18  #include <termios.h>
19  #include <string.h>
20  #include <fcntl.h>
21  #include <math.h>
22  #include <errno.h>
23  #include <sys/select.h>
24
25  static char *modes[] = { "\nVDC\n",
26                   "\nVAC\n",
27                   "\nADC\n",
28                   "\nAAC\n",
29                   "\nVACDC\n",
30                   "\nAACDC\n",
31                   "\nOHMS\n"
32                 };
33
34  static char* unitStr;
35
36  static hw_tektronix_t hw_opts;
37
38  int init_hardware(int argc, char* argv[])
39  {
40    int              opt;
41    int              mandatoryOptions = 1;
42    int              mode             = 0;
43
44    /* evaluate commandline options */
45    while( (opt = getopt(argc, argv, OPT_STRING)) != -1 ) {
46      switch( opt ) {
47        case 'p':
48        case 'n':
49          break;
50        case 'd': hw_opts.devicename = optarg;
51                  mandatoryOptions--;
52                  break;
53        case 'm': mode = atoi(optarg);
54                  break;
55        case 'h':
56        default:  usage(argv[0]);
57                  exit(EXIT_FAILURE);
58      }
59    }
60
61    /* limit checks */
62    if( (mandatoryOptions > 0)
63      ||(mode < 0)
64      ||(mode > MAXMODE)
65    ) {
66      usage(argv[0]);
67      exit(EXIT_FAILURE);
68    }
69
70    /* open serial port */
71    if( (hw_opts.ser_fd = open(hw_opts.devicename, O_RDWR)) < 0 ) {
72      fprintf(stderr, "Can't open serial port.\n");
73      return -1;
74    }
75
76    /* initialize structures */
77    memset(&hw_opts.current_settings, 0x00, sizeof(hw_opts.current_settings));
78    memset(&hw_opts.current_settings, 0x00, sizeof(hw_opts.old_settings));
79    hw_opts.current_settings.c_cflag     = CLOCAL | CS8 | CREAD;
80    hw_opts.current_settings.c_iflag     = 0;
81    hw_opts.current_settings.c_oflag     = 0;
82    hw_opts.current_settings.c_lflag     = 0;
83    hw_opts.current_settings.c_cc[VMIN]  = MINCHARACTERS;
84    cfsetispeed(&hw_opts.current_settings, B9600);
85    cfsetospeed(&hw_opts.current_settings, B9600);
86
87    /* save current serial port settings and apply new settings */
88    tcgetattr(hw_opts.ser_fd, &hw_opts.old_settings);
89    tcsetattr(hw_opts.ser_fd, TCSANOW, &hw_opts.current_settings);
90
91    /* apply mode, rate and format (suffix) settings */
92    write(hw_opts.ser_fd, modes[mode], strlen(modes[mode]));
93    write(hw_opts.ser_fd, RATESTRING,  strlen(RATESTRING));
94    write(hw_opts.ser_fd, WITHOUTMODE, strlen(WITHOUTMODE));
95    tcflush(hw_opts.ser_fd, TCOFLUSH);
96    sleep(MODESETTIME);
97    switch(mode) {
```

```
98    case 0:
99    case 1:
100   case 4:   unitStr = "V";
101             break;
102   case 2:
103   case 3:
104   case 5:   unitStr = "A";
105             break;
106   case 6:   unitStr = "Ohm";
107             break;
108   }
109
110   return 0;
111 }
112
113 void  reset_hardware( void ){
114   /* restore serial port settings */
115   tcsetattr(hw_opts.ser_fd, TCSANOW, &hw_opts.old_settings);
116   /* close serial port */
117   close(hw_opts.ser_fd);
118 }
119
120 void usage(char *progname) {
121     fprintf(stderr, "\n\tUSAGE:\t %s -d dev [-p sock] [-n clients] [-m mode] [-h help]\n"
122                     "\n\tOPTIONS: -d\tSerial port device\n"
123                     "\t\t -p\tSocketfile; will be overwritten if it exists.\n"
124                     "\t\t\tDefault is '%s'.\n"
125                     "\t\t -n\tMax clients; the default limit is %u clients.\n\n"
126                     "\t\t mode:\t 0: volts DC\n"
127                     "\t\t\t 1: volts AC\n"
128                     "\t\t\t 2: amperes DC\n"
129                     "\t\t\t 3: amperes AC\n"
130                     "\t\t\t 4: volts true RMS\n"
131                     "\t\t\t 5: amperes true RMS\n"
132                     "\t\t\t 6: ohms\n"
133                     "\t\t -h\tHelp; view help text.\n"
134                     "\n\tDESCRIPTION:\tServer process which is a Tektronix DMM4020 Reader.\n\n"
135                     , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
136 }
137
138 double getValue( void )
139 {
140   double value = NAN;
141   char   readagain = 1;
142   struct timeval    readtimeout;
143
144   tcflush(hw_opts.ser_fd, TCIFLUSH);
145   write(hw_opts.ser_fd, QUERYSTRING, strlen(QUERYSTRING));
146   tcflush(hw_opts.ser_fd, TCOFLUSH);
147
148   while(readagain) {
149     int    retval, bufcnt = 0;
150     char   buffer[BUFFSIZE];
151     fd_set readset;
152
153     /* initialize filedescriptor sets and timeout structure */
154     FD_ZERO(&readset);
155     FD_SET(hw_opts.ser_fd, &readset);
156     readtimeout.tv_sec  = 0L;
157     readtimeout.tv_usec = READTIMEOUTUSEC;
158
159     /* wait for input data */
160     retval = select(hw_opts.ser_fd+1, &readset, NULL, NULL, &readtimeout);
161
162     if( retval < 0 ) {
163       if( errno != EINTR ) {
164         fprintf(stderr, "# ERROR: %s\n", strerror(errno));
165         value     = NAN;
166         readagain = 0;
167       }
168       break;
169     }
170     if( retval == 0 ) {
171       fprintf(stderr, "# Multimeter did not respond in time\n");
172       value     = NAN;
173       readagain = 0;
174       break;
175     }
176
177     /* read a line */
178     do {
179       read(hw_opts.ser_fd, buffer+bufcnt, 1);
180     } while((buffer[bufcnt++] != '\n'));
181     buffer[bufcnt] = '\0';
182
```

```
183       switch(buffer[0]) {
184       /* -/+ indicate a value => write output to logfile and console */
185       case '-':
186       case '+': value     = atof(buffer);
187                 readagain = 0;
188
189                 fprintf(stdout, "Server sends value: %lf [%s]\n", value, unitStr);
190                 break;
191       /* = indicates prompt => do nothing */
192       case '=': break;
193       /* !/? indicate error => put a message */
194       case '?':
195       case '!': value     = NAN;
196                 readagain = 0;
197       default:  break;
198       }
199     }
200     return value;
201   }
```

C.5.2. tektronix.h

```
1   /*************************************************************
2    * hw_implementation part for Tektronix DMM4020
3    *
4    * File:    tektronix.h
5    * Version: 0.1
6    * Date:    2016-02-25
7    * Author:  Jan Griessbach
8    *
9    * Changes: 0.1: JG first version
10   *
11   *************************************************************/
12
13  #include <termios.h>
14
15  #define QUERYSTRING        "VAL1?\n"
16  #define RATESTRING         "RATE S\n"
17  #define WITHOUTMODE        "FORMAT 1\n"
18  #define BUFFSIZE           1024
19  #define READTIMEOUTUSEC    250000L /* 250ms */
20  #define MODESETTIME        5
21  #define MINCHARACTERS      1
22
23  #define MAXMODE 6
24
25  int    init_hardware(int argc, char* argv[]);
26  double getValue(void);
27  void   reset_hardware(void);
28  void   usage(char *progname);
29
30  typedef struct struct_hw_tektronix {
31    char           *devicename;
32    int            ser_fd;
33    struct termios current_settings;
34    struct termios old_settings;
35  } hw_tektronix_t;
```

C.5.3. BC_HW_Implementation.h

```
1   /*************************************************************
2    * Server process for Tektronix DMM4020 Reader
3    *
4    * File:    BC_DMM4020Reader_SP/BC_HW_Implementation.h
5    * Version: 0.2
6    * Date:    2016-02-22
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.2: JG added main argument handover by getopt().
10   *          0.1: JG first version.
11   *
12   *************************************************************/
```

```
13
14 #ifndef BC_HW_IMPLEMENTATION_H_
15 #define BC_HW_IMPLEMENTATION_H_
16
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <unistd.h>
20
21 //this defines are depending by implementation
22 #define MAX_CLIENTS_DEFAULT 10                  //max number of connected clients
23 #define UDS_FILE            "/tmp/DMM4020sock.uds"    //default path
24 #define OPT_STRING          "p:n:d:m:h"
25
26 #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.6. BC_Amperemeter_SP

C.6.1. BC_HW_ADC_I.c

```
1  /*****************************************************************
2   * Server process for current measurement
3   *
4   * File:    BC_Amperemeter_SP/BC_HW_ADC_I.c
5   * Version: 0.3
6   * Date:    2016-03-03
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.3: NR modified init_hardware() for new version of BC_HW_ADC.h
10  *          0.2: NR added main argument handover by getopt().
11  *          0.1: NR first version.
12  *
13  *****************************************************************/
14
15 #include <stdio.h>
16 #include <stdlib.h>
17 #include <unistd.h>
18 #include "BC_HW_Lib/BC_HW_ADC.h"
19 #include "BC_HW_Implementation.h"
20
21 void usage(char *progname) {
22     fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
23                     "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
24                     "\t\t\tDefault is '%s'.\n"
25                     "\t\t -n\tMax clients; default limit is %u clients.\n"
26                     "\t\t -h\tHelp; view help text.\n"
27                     "\n\tDESCRIPTION:\tServer process for current measurement.\n\n"
28                     , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
29 }
30
31 int init_hardware(int argc, char* argv[]) {
32     int retval;
33
34     //evaluate main function arguments
35     while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
36         switch(retval) {
37         case 'p':
38         case 'n':
39         case 'h':           //show help description
40             break;
41         default:
42             usage(argv[0]);
43             exit(EXIT_FAILURE);
44         }
45     }
46     retval = BCHW_initializeADCgpio(CS_PIN_ADC_I, ADC_CONV_PIN);
47     if(retval < 0)
48         return -1;
49
50     //start first adc convolution */
51     retval = BCHW_readFromADC(CS_PIN_ADC_I, ADC_CONV_PIN);
52
53     return 1;
54 }
55
56 double getValue(void) {
```

```
57      int temp;
58      double val;
59
60      temp = BCHW_readFromADC(CS_PIN_ADC_I, ADC_CONV_PIN);
61      val = BCHW_parseToCurrent(temp);
62      fprintf(stdout, "Server sends value: %lf [uA]\n", val * FAC_MICRO);
63      return val;
64  }
65
66  void reset_hardware(void) {
67      if(BCHW_releaseADCgpio() < 0)
68          fprintf(stderr, "Error occurred in reset_hardware()\n");
69  }
```

C.6.2. BC_HW_Implementation.h

```
1   /*************************************************************
2    * Server process for current measurement
3    *
4    * File:    BC_Amperemeter_SP/BC_HW_Implementation.h
5    * Version: 0.2
6    * Date:    2016-02-22
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.2: JG, NR added main argument handover by getopt().
10   *          0.1: NR first version.
11   *
12   *************************************************************/
13
14  #ifndef BC_HW_IMPLEMENTATION_H_
15  #define BC_HW_IMPLEMENTATION_H_
16
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <unistd.h>
20
21  //this defines are depending by implementation
22  #define MAX_CLIENTS_DEFAULT 10                  //max number of connected clients
23  #define UDS_FILE            "/tmp/Isock.uds"    //default path
24  #define OPT_STRING          "p:n:h"
25
26  #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.7. BC_Voltmeter_SP

C.7.1. BC_HW_ADC_V.c

```
1   /*************************************************************
2    * Server process for voltage measurement
3    *
4    * File:    BC_Voltmeter_SP/BC_HW_ADC_V.c
5    * Version: 0.2
6    * Date:    2016-02-22
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.2: JG, NR added main argument handover by getopt().
10   *          0.1: NR first version.
11   *
12   *************************************************************/
13
14  #include <stdio.h>
15  #include <stdlib.h>
16  #include <unistd.h>
17  #include "BC_HW_Lib/BC_HW_ADC.h"
18  #include "BC_HW_Implementation.h"
19
20  void usage(char *progname) {
21      fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
```

```
22                      "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
23                      "\t\t\tDefault is '%s'.\n"
24                      "\t\t -n\tMax clients; default limit is %u clients.\n"
25                      "\t\t -h\tHelp; view help text.\n"
26                      "\n\tDESCRIPTION:\tServer process for voltage measurement.\n\n"
27                      , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
28  }
29
30  int init_hardware(int argc, char* argv[]) {
31      int retval;
32
33      //evaluate main function arguments
34      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
35          switch(retval) {
36          case 'p':
37          case 'n':
38          case 'h':           //show help description
39              break;
40          default:
41              usage(argv[0]);
42              exit(EXIT_FAILURE);
43          }
44      }
45      //Hardware Initializations
46      retval = BCHW_initializeADCgpio(CS_PIN_ADC_V, ADC_CONV_PIN);
47      if(retval < 0)
48          return -1;
49
50      //start first adc convolution
51      retval = BCHW_readFromADC(CS_PIN_ADC_V, ADC_CONV_PIN);
52
53      return 1;
54  }
55
56  double getValue(void) {
57      int temp;
58      double val;
59
60      temp = BCHW_readFromADC(CS_PIN_ADC_V, ADC_CONV_PIN);
61      val = BCHW_parseToVoltage(temp);
62      fprintf(stdout, "Server sends value: %lf [V]\n", val);
63      return val;
64  }
65
66  void reset_hardware(void) {
67      if(BCHW_releaseADCgpio() < 0)
68          fprintf(stderr, "Error occurred in reset_hardware()\n");
69  }
```

C.7.2. BC_HW_Implementation.h

```
1   /*****************************************************************
2    * Server process for voltage measurement
3    *
4    * File:    BC_Voltmeter_SP/BC_HW_Implementation.h
5    * Version: 0.2
6    * Date:    2016-02-22
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.2: JG, NR added main argument handover by getopt().
10   *          0.1: NR first version.
11   *
12   *****************************************************************/
13
14  #ifndef BC_HW_IMPLEMENTATION_H_
15  #define BC_HW_IMPLEMENTATION_H_
16
17  #include <stdio.h>
18  #include <stdlib.h>
19  #include <unistd.h>
20
21  //this defines are depending by implementation
22  #define MAX_CLIENTS_DEFAULT 10                  //max number of connected clients
23  #define UDS_FILE            "/tmp/Vsock.uds"    //default path
24  #define OPT_STRING          "p:n:h"
25
26  #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.8. BC_Temperature_SP

C.8.1. BC_HW_TEMP.c

```
/************************************************************
 * Server process for current measurement
 *
 * File:    BC_Temperature_SP/BC_HW_TEMP.c
 * Version: 0.3
 * Date:    2016-04-30
 * Author:  Nico Rieckmann
 *
 * Changes: 0.1: NR first version.
 *
 ************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "BC_HW_Lib/BC_HW_Temperature.h"
#include "BC_HW_Implementation.h"

void usage(char *progname) {
    fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
                    "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
                    "\t\t\tDefault is '%s'.\n"
                    "\t\t -n\tMax clients; default limit is %u clients.\n"
                    "\t\t -h\tHelp; view help text.\n"
                    "\n\tDESCRIPTION:\tServer process for temperature measurement.\n\n"
                    , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
}

int init_hardware(int argc, char* argv[]) {
    int retval;

    //evaluate main function arguments
    while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
        switch(retval) {
        case 'p':
        case 'n':
        case 'h':           //show help description
            break;
        default:
            usage(argv[0]);
            exit(EXIT_FAILURE);
        }
    }
    retval = BCHW_initializeTempGPIO(CS_PIN_TEMP);
    if(retval < 0)
        return -1;

    //test first temperature reading*/
    retval = BCHW_getTempValue(CS_PIN_TEMP);

    return 1;
}

double getValue(void) {
    //int16_t temp;
    double val;

    val = BCHW_getTempValue(CS_PIN_TEMP);
    //val = BCHW_parseToCelsius(temp);
    fprintf(stdout, "Server sends value: %lf Â°[C]\n", val);
    return val;
}

void reset_hardware(void) {
    if(BCHW_releaseTempGPIO() < 0)
        fprintf(stderr, "Error occurred in reset_hardware()\n");
}
```

C.8.2. BC_HW_Implementation.h

```
/*************************************************************
 * Server process for current measurement
 *
 * File:    BC_Temperature_SP/BC_HW_Implementation.h
 * Version: 0.2
 * Date:    2016-02-22
 * Author:  Nico Rieckmann
 *
 * Changes: 0.2: JG, NR added main argument handover by getopt().
 *          0.1: NR first version.
 *
 *************************************************************/

#ifndef BC_HW_IMPLEMENTATION_H_
#define BC_HW_IMPLEMENTATION_H_

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

//this defines are depending by implementation
#define MAX_CLIENTS_DEFAULT 10                  //max number of connected clients
#define UDS_FILE            "/tmp/Tsock.uds"    //default path
#define OPT_STRING          "p:n:h"

#endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.9. BC_VoltageController_SP

C.9.1. BC_HW_VoltageController.c

```
/*************************************************************
 * Extern functions for VoltageController
 *
 * File:    BC_VoltageController/BC_HW_VoltageController.c
 * Version: 0.1
 * Date:    2016-03-07
 * Author:  Nico Rieckmann
 *
 * Changes: 0.1: NR first version.
 *
 *************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "BC_HW_Lib/BC_HW_DAC.h"
#include "BC_HW_Implementation.h"
#include "../BC_Protocol/BC_ActuatorProtocol.h"

void usage(char *progname) {
    fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
                    "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
                    "\t\t\tDefault is '%s'.\n"
                    "\t\t -n\tMax clients; default limit is %u clients.\n"
                    "\t\t -h\tHelp; view help text.\n"
                    "\n\tDESCRIPTION:\tServer process to control the DAC voltage.\n\n"
                    , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
}

int init_hardware(int argc, char* argv[]) {
    int retval;

    //evaluate main function arguments
    while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
        switch(retval) {
        case 'p':
        case 'n':
        case 'h':           //show help description
            break;
        default:
            exit(EXIT_FAILURE);
        }
    }
    //Hardware Initializations
```

```
45      retval = BCHW_initializeDACgpio(CS_PIN_DAC);
46      if(retval < 0)
47          return -1;
48
49      return 1;
50  }
51
52  int behavioral(BCFrame_t *inputframe) {
53      double inputArg;
54      APCommand_t command = 0x00;
55
56      command = inputframe->fields.command;
57      inputArg = BC_getArgumentAsDouble(inputframe, BCAP_REQ_SETDAC_OFFSET_VALUE);
58      switch(command) {
59          case BCAP_REQ_SETDAC:      //received a "set value" frame
60              BCHW_sendToDAC(CS_PIN_DAC, BCHW_parseToDACvalue(inputArg));
61              fprintf(stdout, "Server received value: %lf [V]\n", inputArg);
62              break;
63          case BCAP_REQ_QUIT:      //received a quit frame
64              return -2;
65          default:
66              fprintf(stdout, "Unknown command in frame\n"); //received something else
67      }
68      return 0;
69  }
70
71  void reset_hardware(void) {
72      if(BCHW_releaseDACgpio() < 0)
73          fprintf(stderr, "Error occurred in BCHW_releaseAll()\n");
74  }
```

C.9.2. BC_HW_Implementation.h

```
1   /*****************************************************************
2    * Extern functions for VoltageController
3    *
4    * File:    BC_VoltageController/BC_HW_Implementation.h
5    * Version: 0.1
6    * Date:    2016-03-07
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *
11   *****************************************************************/
12
13  #ifndef BC_HW_IMPLEMENTATION_H_
14  #define BC_HW_IMPLEMENTATION_H_
15
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <unistd.h>
19  #include "../BC_Protocol/BC_ActuatorProtocol.h"
20
21  //this defines are depending by implementation
22  #define MAX_CLIENTS_DEFAULT 1                   //max number of connected clients
23  #define UDS_FILE            "/tmp/Dsock.uds"    //default path
24  #define OPT_STRING          "p:n:h"
25
26  #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.10. BC_Potentiometer_SP

C.10.1. BC_HW_Potentiometer.c

```
1   /*****************************************************************
2    * Extern functions for potentiometers
3    *
4    * File:    BC_Potentiometer_SP/BC_HW_Potentiometer.c
```

```
5    * Version: 0.1
6    * Date:    2016-03-07
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *
11   ************************************************************/
12
13   #include <stdio.h>
14   #include <stdlib.h>
15   #include <unistd.h>
16   #include "BC_HW_Lib/BC_HW_Potentiometer.h"
17   #include "BC_HW_Implementation.h"
18   #include "../BC_Protocol/BC_ActuatorProtocol.h"
19
20   void usage(char *progname) {
21       fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
22                       "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
23                       "\t\t\tDefault is '%s'.\n"
24                       "\t\t -n\tMax clients; default limit is %u clients.\n"
25                       "\t\t -h\tHelp; view help text.\n"
26                       "\n\tDESCRIPTION:\tServer process to control digital potentiometer.\n\n"
27                       , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
28   }
29
30   int init_hardware(int argc, char* argv[]) {
31       int retval;
32
33       //evaluate main function arguments
34       while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
35           switch(retval) {
36           case 'p':
37           case 'n':
38           case 'h':           //show help description
39               break;
40           default:
41               exit(EXIT_FAILURE);
42           }
43       }
44       //Hardware Initializations
45       retval = BCHW_initializePotiGPIO(CS_PIN_LED_R);
46       retval = BCHW_initializePotiGPIO(CS_PIN_LED_G);
47       retval = BCHW_initializePotiGPIO(CS_PIN_LED_B);
48       if(retval < 0)
49           return -1;
50
51       return 1;
52   }
53
54   int behavioral(BCFrame_t *inputframe) {
55       float inputArg[MAX_MEMBER];
56       APCommand_t command = 0x00;
57       uint16_t potival[MAX_MEMBER];
58       int i;
59
60       command = inputframe->fields.command;
61       inputArg[0] = BC_getArgumentAsFloat(inputframe, BCAP_REQ_LED_RED_OFFSET_VALUE);
62       inputArg[1] = BC_getArgumentAsFloat(inputframe, BCAP_REQ_LED_GREEN_OFFSET_VALUE);
63       inputArg[2] = BC_getArgumentAsFloat(inputframe, BCAP_REQ_LED_BLUE_OFFSET_VALUE);
64
65       fprintf(stdout, "Server received: red = %lf\tgreen = %lf\tblue = %lf\n"
66               ,inputArg[0], inputArg[1], inputArg[2]);
67
68       //normalize inputArg from [0 - 100%] in [0.0 - 1.0] * POTI_FS, and 0.5 for int round
69       for(i=0; i<MAX_MEMBER; i++) {
70           potival[i] = (uint16_t) ((POTI_FULL_SCALE * (inputArg[i] / 100.0)) + 0.5);
71       }
72
73       //inputArg = (double) BC_getArgumentAsUint64(inputframe, BCRP_REQ_SETVALUE_OFFSET_VALUE);
74       switch(command) {
75           case BCAP_REQ_SETLED:      //received a "set value" frame
76               BCHW_sendToPoti(CS_PIN_LED_R, potival[0]);
77               BCHW_sendToPoti(CS_PIN_LED_G, potival[1]);
78               BCHW_sendToPoti(CS_PIN_LED_B, potival[2]);
79               break;
80           case BCAP_REQ_QUIT:      //received a quit frame
81               return -2;
82           default:
83               fprintf(stderr, "Unknown command in frame\n"); //received something else
84       }
85       return 1;
86   }
87
88   void reset_hardware(void) {
89       if(BCHW_releasePotiGPIO() < 0)
```

```
90          fprintf(stderr, "Error occurred in BCHW_releaseAll()\n");
91  }
```

C.10.2. BC_HW_Implementation.h

```
1   /*************************************************************
2    * Extern functions for potentiometers
3    *
4    * File:    BC_Potentiometer_SP/BC_HW_Implementation.h
5    * Version: 0.1
6    * Date:    2016-03-07
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *
11   *************************************************************/
12
13  #ifndef BC_HW_IMPLEMENTATION_H_
14  #define BC_HW_IMPLEMENTATION_H_
15
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <unistd.h>
19  #include "../BC_Protocol/BC_ActuatorProtocol.h"
20
21  //this defines are depending by implementation
22  #define MAX_CLIENTS_DEFAULT 1                   //max number of connected clients
23  #define UDS_FILE            "/tmp/Psock.uds"    //default path
24  #define OPT_STRING          "p:n:h"
25
26  #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.11. BC_RelayController_SP

C.11.1. BC_HW_RelayController.c

```
1   /*************************************************************
2    * Extern functions for relay controlling
3    *
4    * File:    BC_RelayController/BC_HW_RelayController.c
5    * Version: 0.1
6    * Date:    2016-02-28
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *
11   *************************************************************/
12
13  #include <stdio.h>
14  #include <stdlib.h>
15  #include <unistd.h>
16  #include "BC_HW_Lib/BC_HW_Relay.h"
17  #include "BC_HW_Implementation.h"
18  #include "../BC_Protocol/BC_RelayProtocol.h"
19
20  void usage(char *progname) {
21      fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-n clients] [-h help]\n"
22                      "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
23                      "\t\t\tDefault is '%s'.\n"
24                      "\t\t -n\tMax clients; default limit is %u clients.\n"
25                      "\t\t -h\tHelp; view help text.\n"
26                      "\n\tDESCRIPTION:\tServer process to access the relays.\n\n"
27                      , progname, UDS_FILE, MAX_CLIENTS_DEFAULT);
28  }
29
30  int init_hardware(int argc, char* argv[]) {
31      int retval;
32
```

```
33      //evaluate main function arguments
34      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
35          switch(retval) {
36          case 'p':
37          case 'n':
38          case 'h':           //show help description
39              break;
40          default:
41              usage(argv[0]);
42              exit(EXIT_FAILURE);
43          }
44      }
45      //Hardware Initializations
46      retval = BCHW_initializeRelayGPIO();
47      if(retval < 0)
48          return -1;
49
50      BCHW_openRelay(PIN_RELAYS_VN);
51      BCHW_openRelay(PIN_RELAYS_VP);
52      BCHW_openRelay(PIN_RELAYS_I);
53
54      return 1;
55  }
56
57  int behavioral(BCFrame_t *inputframe) {
58      RPArgument_t inputArg = 0x00;
59      RPCommand_t command = 0x00;
60      char* stateStr[] = {STR_OPEN, STR_CLOSE};
61      int8_t retval = 0;
62
63      command = inputframe->fields.command;
64      inputArg = BC_getArgumentAsUint8(inputframe, 0);
65      switch(command) {
66          case BCRP_REQ_SETREGISTER:      //received a "set register" frame
67              //plausibility check of argument value
68              if(inputArg <= (BCRP_RELAY_VN | BCRP_RELAY_VP | BCRP_RELAY_I))
69                  BCHW_setRelayPattern(inputArg);
70              else
71                  retval = -1;
72              break;
73          case BCRP_REQ_SETRELAY:       //received a "set relay" frame
74              if(inputArg == BCRP_RELAY_VN)
75                  BCHW_closeRelay(PIN_RELAYS_VN);
76              else if(inputArg == BCRP_RELAY_VP)
77                  BCHW_closeRelay(PIN_RELAYS_VP);
78              else if(inputArg == BCRP_RELAY_I)
79                  BCHW_closeRelay(PIN_RELAYS_I);
80              else
81                  retval = -1;
82              break;
83          case BCRP_REQ_DELRELAY:       //received a "delete relay" frame
84              if(inputArg == BCRP_RELAY_VN)
85                  BCHW_openRelay(PIN_RELAYS_VN);
86              else if(inputArg == BCRP_RELAY_VP)
87                  BCHW_openRelay(PIN_RELAYS_VP);
88              else if(inputArg == BCRP_RELAY_I)
89                  BCHW_openRelay(PIN_RELAYS_I);
90              else
91                  retval = -1;
92              break;
93          case BCRP_REQ_TOGRELAY:      //received a "toggel relay" frame
94              if(inputArg == BCRP_RELAY_VN)
95                  BCHW_toggleRelay(PIN_RELAYS_VN);
96              else if(inputArg == BCRP_RELAY_VP)
97                  BCHW_toggleRelay(PIN_RELAYS_VP);
98              else if(inputArg == BCRP_RELAY_I)
99                  BCHW_toggleRelay(PIN_RELAYS_I);
100             else
101                 retval = -1;
102             break;
103         case BCRP_REQ_QUIT:      //received a quit frame
104             return -2;
105         default:
106             fprintf(stdout, "Unknown command in frame\n"); //received something else
107             return 0;
108     }
109     if(retval < 0) {
110         fprintf(stdout, "Unknown argument for this command\n");
111         return 0;
112     }
113     //print relay states
114     retval = BCHW_getRelayPosition(PIN_RELAYS_VN);
115     fprintf(stdout, "relay_VN: %s\t", stateStr[retval]);
116     retval = BCHW_getRelayPosition(PIN_RELAYS_VP);
117     fprintf(stdout, "relay_VP: %s\t", stateStr[retval]);
```

```
118     retval = BCHW_getRelayPosition(PIN_RELAYS_I);
119     fprintf(stdout, "relay_I: %s\n", stateStr[retval]);
120     return 1;
121 }
122
123 void reset_hardware(void) {
124     if(BCHW_releaseRelayGPIO() < 0)
125         fprintf(stderr, "Error occurred in BCHW_releaseAll()\n");
126 }
```

C.11.2. BC_HW_Implementation.h

```
1  /*****************************************************************
2   * Extern functions for potentiometers
3   *
4   * File:    BC_Potentiometer_SP/BC_HW_Implementation.h
5   * Version: 0.1
6   * Date:    2016-03-07
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version.
10  *
11  *****************************************************************/
12
13 #ifndef BC_HW_IMPLEMENTATION_H_
14 #define BC_HW_IMPLEMENTATION_H_
15
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <unistd.h>
19 #include "../BC_Protocol/BC_ActuatorProtocol.h"
20
21 //this defines are depending by implementation
22 #define MAX_CLIENTS_DEFAULT 1                   //max number of connected clients
23 #define UDS_FILE            "/tmp/Psock.uds"    //default path
24 #define OPT_STRING          "p:n:h"
25
26 #endif /* BC_HW_IMPLEMENTATION_H_ */
```

C.12. BC_MainController_CP

C.12.1. BC_ClientMain.c

```
1  /*****************************************************************
2   * Client process to control the measurement and actuator processes
3   * from modular batcycle process family
4   *
5   * File:    BC_ClientMain.c
6   * Version: 0.2
7   * Date:    2016-04-29
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.2: NR added DAC and battery voltageAlignment function
11  *          0.1: NR first version
12  *
13  *****************************************************************/
14
15 #include "BC_Client.h"
16 #include "BC_Regulation.h"
17
18 /* global variables */
19 volatile sig_atomic_t runFlag = true;
20 volatile sig_atomic_t timerFlag = false;
21
22 /* signal handlers */
23 void sigEnd_handler(int sign) {
24     runFlag = false;
25 }
```

```
26
27  int main(int argc, char* argv[]) {
28      config_t cfgInit, cfgParam;         //config file descriptors
29      config_setting_t *cycleList;        //pointer to cycle list
30      BCParamTree_t bcp;                  //batcycle parameter tree sturct
31      BCMValSet_t bcm;                    //batcycle measure values struct
32      BCAValSet_t bca;                    //batcycle acting values struct
33      int sock_fd[MAX_SERVERS];           //socket descriptor array
34      //int mandatoryOptions = 0;           //getopt necessaries
35      int i, retval = 0;                  //function return value
36      int listLen;                        //length of cycle list in param file
37      int prev = 0;    //previous cycling iterations from aborted program execution
38
39      printf("#%s start\n", argv[0]);
40
41      //initialize struct types
42      memset(&bca, 0x00, sizeof(BCAValSet_t));
43      memset(&bcm, 0x00, sizeof(BCMValSet_t));
44      memset(&bcp, 0x00, sizeof(BCParamTree_t));
45      bcm.period.tv_nsec = MEASURE_PERIOD_NS;     //set quasi equidistant sampling period
46      //bcm.period.tv_sec = 1;                        //TODO kann weg
47      errno = 0;      //reset error code
48
49      bcp.initfile.filepath = FILE_INI;
50      bcp.paramfile.filepath = FILE_PARAM;
51      //evaluate main function arguments
52      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
53          switch(retval) {
54          case 'i':       //path of init file
55              bcp.initfile.filepath = optarg;
56              //mandatoryOptions--;
57              break;
58          case 'p':       //path of parameter file
59              bcp.paramfile.filepath = optarg;
60              //mandatoryOptions--;
61              break;
62          case 'h':       //help text
63          default:
64              usage(argv[0]);
65              exit(EXIT_FAILURE);
66          }
67      }
68
69      /*if(mandatoryOptions > 0) {
70          usage(argv[0]);
71          exit(EXIT_FAILURE);
72      }*/
73
74      //initialize bcp struct with initfile values
75      bcp.initfile.fptr = openConfigFile(&cfgInit, bcp.initfile.filepath);
76      if(bcp.initfile.fptr == NULL)
77          exit(EXIT_FAILURE);
78      if(readInitFile(&cfgInit, &bcp) < 0) {
79          //fprintf(stderr, "Failure occurred in readInitFile()\n");
80          closeConfigFile(&cfgInit, bcp.initfile.fptr);
81          sleep(10);
82          exit(EXIT_FAILURE);
83      }
84
85      //initialize bcp struct with paramfile values
86      bcp.paramfile.fptr = openConfigFile(&cfgParam, bcp.paramfile.filepath);
87      if(bcp.paramfile.fptr == NULL) {
88          closeConfigFile(&cfgParam, bcp.initfile.fptr);
89          exit(EXIT_FAILURE);
90      }
91      if(readParamFile(&cfgParam, &bcp, &cycleList) < 0) {
92          //fprintf(stderr, "Failure occurred in readParamFile()\n");
93          closeConfigFile(&cfgInit, bcp.initfile.fptr);
94          closeConfigFile(&cfgParam, bcp.paramfile.fptr);
95          sleep(10);
96          exit(EXIT_FAILURE);
97      }
98
99      //socket initialization
100     for(i=0; i<MAX_SERVERS; i++) {
101         retval = connectToSock(&sock_fd[i], bcp.sockpaths[i]);
102         if(retval < 0)
103             fprintf(stderr, "Failure at connect to socket '%s': %s\n", bcp.sockpaths[i], strerror(errno));
104         else
105             printf(".Client is connected on '%s'\n", bcp.sockpaths[i]);
106     }
107
108     //try to register as master client on servers
109     retval = BCMP_getControl(sock_fd[SERV_CURR]);
110     if(retval < 0) {
```

```
111         fprintf(stderr, "Can't register as master client on '%s'\n", bcp.sockpaths[SERV_CURR]);
112         runFlag = false;
113     }
114     retval = BCMP_getControl(sock_fd[SERV_VOLT]);
115     if(retval < 0) {
116         fprintf(stderr, "Can't register as master client on '%s'\n", bcp.sockpaths[SERV_VOLT]);
117         runFlag = false;
118     }
119     retval = BCMP_getControl(sock_fd[SERV_TEMP]);
120     if(retval < 0) {
121         fprintf(stderr, "Can't register as master client on '%s'\n", bcp.sockpaths[SERV_TEMP]);
122     }
123     retval = BCMP_startContinuousMode(sock_fd[SERV_TEMP], TEMP_PERIOD_MS);
124     if(retval < 0)
125         fprintf(stderr, "Can't set server in continuous mode on '%s'\n", bcp.sockpaths[SERV_TEMP]);
126     fprintf(stdout, "\n");
127
128     signal(SIGINT, sigEnd_handler);         //register signalhandler for SIGINT signal
129     signal(SIGPIPE, SIG_IGN);       //ignore global a SIGPIPE signal if socket access crashed
130
131 /*** cycle program begin ***/
132     while(runFlag) {    //loop to allow reconnects on sockets if an error occurred
133         retval = 0;
134         if(voltageAlignment(sock_fd, &bcm, &bca) < 0) {
135             runFlag = false;
136             break;
137         }
138
139 /*** Startphase ***/
140         fprintf(stdout, "Startphase: \n");
141
142         //adjust led light
143         bca.act_RGBlight[0] = bcp.nominals.nom_RGBlight[0];
144         bca.act_RGBlight[1] = bcp.nominals.nom_RGBlight[1];
145         bca.act_RGBlight[2] = bcp.nominals.nom_RGBlight[2];
146         BCAP_sendValueToPoti(sock_fd[SERV_POTI], bca.act_RGBlight);
147
148         //const voltage mode
149         if(strcmp(bcp.nominals.charge_mode, STR_MODE_cV) == 0)
150             retval |= cV_Regulation(sock_fd, &bcm, &bca, &bcp);
151         //const current mode
152         else if(strcmp(bcp.nominals.charge_mode, STR_MODE_cC) == 0)
153             retval |= cC_Regulation(sock_fd, &bcm, &bca, &bcp);
154         //const current const voltage mode
155         else if(strcmp(bcp.nominals.charge_mode, STR_MODE_cCcV) == 0)
156             retval |= cCcV_Regulation(sock_fd, &bcm, &bca, &bcp);
157
158         if(retval < 0) {        //if failure, cancel all
159             fprintf(stdout, "Failure occured at %s regulation: %s\n", bcp.nominals.charge_mode, strerror(errno));
160             runFlag = false;
161             break;
162         }
163
164 /*** Cyclephase ***/
165         fprintf(stdout, "Cyclephase: \n");
166
167         for(i=prev; i<bcp.cycles; i++) {  //iterate the number of cycles
168             listLen = -1;
169
170             while(runFlag && listLen) {     //iterate the parameter sets into cycle list
171                 retval |= convalescenceTime(sock_fd, &bcm, &bca, &bcp);
172
173                 listLen = getNextParamSet(cycleList, &bcp, listLen);
174                 if(listLen < 0) {
175                     fprintf(stderr, "Failure occurred during get next parameter set\n");
176                     runFlag = false;
177                 }
178
179                 //adjust led light
180                 bca.act_RGBlight[0] = bcp.nominals.nom_RGBlight[0];
181                 bca.act_RGBlight[1] = bcp.nominals.nom_RGBlight[1];
182                 bca.act_RGBlight[2] = bcp.nominals.nom_RGBlight[2];
183                 BCAP_sendValueToPoti(sock_fd[SERV_POTI], bca.act_RGBlight);
184
185                 //const voltage mode
186                 if(strcmp(bcp.nominals.charge_mode, STR_MODE_cV) == 0)
187                     retval |= cV_Regulation(sock_fd, &bcm, &bca, &bcp);
188                 //const current mode
189                 else if(strcmp(bcp.nominals.charge_mode, STR_MODE_cC) == 0)
190                     retval |= cC_Regulation(sock_fd, &bcm, &bca, &bcp);
191                 //const current const voltage mode
192                 else if(strcmp(bcp.nominals.charge_mode, STR_MODE_cCcV) == 0)
193                     retval |= cCcV_Regulation(sock_fd, &bcm, &bca, &bcp);
194
195                 if(retval < 0) {    //TODO if failure, reconnect all
```

```
196                     fprintf(stdout, "Failure occured at %s regulation: %s\n", bcp.nominals.charge_mode, strerror(errno));
197                     runFlag = false;
198                     sleep(10);
199                     break;
200                 }
201             }
202
203             if(!runFlag)        //exit point
204                 break;
205             fprintf(stdout, "%d cyclelist iterations done\n"
206                             "%d cyclelist iterations remaining\n\n", i+1, bcp.cycles-i-1);
207         }
208
209         break; //exit point of all
210     }
211
212 /*** clean up all ***/
213     BCRP_setRelays(sock_fd[SERV_REL], !BCRP_RELAY_ALL);  //open all relays
214     BCMP_stopContinuousMode(sock_fd[SERV_TEMP]);        //release used server
215     BCMP_returnControl(sock_fd[SERV_TEMP]);
216     BCMP_returnControl(sock_fd[SERV_CURR]);
217     BCMP_returnControl(sock_fd[SERV_VOLT]);
218     for(i=0; i< MAX_SERVERS; i++) {     //quit
219         /*BCFrame_t frame;   kann weg
220
221         BC_initializeFrame(&frame, BCMP_REQ_QUIT, 0);   //close servers
222         BC_sendFrame(sock_fd[i], &frame);*/
223         close(sock_fd[i]);
224     }
225     closeConfigFile(&cfgInit, bcp.initfile.fptr);
226     closeConfigFile(&cfgParam, bcp.paramfile.fptr);
227     printf("#%s end\n", argv[0]);
228     return EXIT_SUCCESS;
229 }
```

C.12.2. BC_Client.c

```
1  /*************************************************************
2   * Client process to control the measurement and actuator processes
3   * from modular batcycle process family
4   *
5   * File:    BC_Client.c
6   * Version: 0.2
7   * Date:    2016-04-29
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.2: NR added DAC and battery voltageAlignment function
11  *          0.1: NR first version
12  *
13  *************************************************************/
14
15 #include "BC_Client.h"
16
17 void usage(char *progname) {
18     fprintf(stderr, "\n\tUSAGE:\t %s [-i initFile] [-p paramFile] [-h help]\n"
19                     "\n\tOPTIONS: -i\tinitFile; will be read. Default is '%s'.\n"
20                     "\t\t -p\tparamFile; will be read. Default is '%s'.\n"
21                     "\t\t -h\thelp; view help text.\n"
22                     "\n\tDESCRIPTION:\tClient process to control the measurement and actuator \n"
23                     "\t\t\tprocesses from modular batcycle process family.\n\n"
24                     , progname, FILE_INI, FILE_PARAM);
25 }
26
27 int connectToSock(int *sock_fd, const char *sockFilePath) {
28     struct sockaddr_un address;     //address struct
29     struct timeval timeout;
30
31     if(sockFilePath == NULL) {
32         fprintf(stderr, "No sockFilePath string\n");
33         return -1;
34     }
35
36     //create socket discriptor
37     *sock_fd = socket(PF_LOCAL, SOCK_STREAM, 0);    //use unix socket in stream mode, block mode
38     if(*sock_fd <= 0) {         //if an error occurred
39         #ifdef DEBUG
40         fprintf(stderr, "Failure at socket() call for '%s'\n", sockFilePath);
41         #endif
42         close(*sock_fd);
```

```
43          *sock_fd = -1;
44          return -1;
45      }
46
47      //init structs
48      address.sun_family = AF_LOCAL;          //make them to unix sockets
49      strcpy(address.sun_path, sockFilePath);
50
51      //set socket to blocking modus with an timeout
52      timeout.tv_sec = TIMEOUT_SEC;
53      timeout.tv_usec = TIMEOUT_USEC;
54      if(setsockopt(*sock_fd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout)) < 0) {
55          #ifdef DEBUG
56          fprintf(stderr, "Failure at setsockopt() call for '%s'\n", sockFilePath);
57          #endif
58          close(*sock_fd);
59          *sock_fd = -1;
60          return -1;
61      }
62
63      //connect to socket file
64      if(connect(*sock_fd, (struct sockaddr*) &address, sizeof(address)) < 0) {
65          #ifdef DEBUG
66          fprintf(stderr, "Failure at connect to socket '%s'\n", sockFilePath);
67          #endif
68          close(*sock_fd);
69          *sock_fd = -1;
70          return -1;
71      }
72
73      return 1;
74  }
75
76  int BCMP_getControl(int sock_fd) {
77      MPFrame_t frame;
78      int retval;
79
80      //set endchar of frame
81      //BC_initializeFrame(&frame, BCMP_REQ_GETCONTROL, BCMP_getDefaultArgLen(BCMP_REQ_GETCONTROL));
82      BCMP_initializeFrame(&frame, BCMP_REQ_GETCONTROL);
83      retval = BC_sendFrame(sock_fd, &frame);
84      if(retval < 0) {
85          #ifdef DEBUG
86          perror("Error occurred at BC_sendFrame()");
87          #endif
88          close(sock_fd);
89          return -1;
90      }
91      //receive response frame
92      retval = BC_recvFrame(sock_fd, &frame);
93      if(retval <= 0) {
94          #ifdef DEBUG
95          perror("Error occurred in BC_recvFrame()");
96          #endif
97          close(sock_fd);
98          return -1;
99      }
100     retval = BC_getArgumentAsUint8(&frame, BCMP_RES_CONTROL_OFFSET_CONTROL);
101     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_GRANT)
102         return 1;
103     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_DENIAL)
104         fprintf(stderr, "Server response was an denial frame\n");
105     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_RELEASE)
106         fprintf(stderr, "Server answered on a release frame\n");
107     return -1;
108 }
109
110 int BCMP_returnControl(int sock_fd) {
111     MPFrame_t frame;
112     int retval;
113
114     //set endchar of frame
115     //BC_initializeFrame(&frame, BCMP_REQ_RETCONTROL, BCMP_getDefaultArgLen(BCMP_REQ_RETCONTROL));
116     BCMP_initializeFrame(&frame, BCMP_REQ_RETCONTROL);
117     retval = BC_sendFrame(sock_fd, &frame);
118     if(retval < 0) {
119         #ifdef DEBUG
120         perror("Error occurred at BC_sendFrame()");
121         #endif
122         close(sock_fd);
123     }
124     return retval;
125 }
126
127 int BCMP_startContinuousMode(int sock_fd, uint32_t interval_ms) {  //TODO
```

```
128     MPFrame_t frame;
129
130     BCMP_initializeFrame(&frame, BCMP_REQ_STARTCONTINUOUS);
131     BC_setArgumentAsUint32(&frame, BCMP_REQ_STARTCONTINUOUS_OFFSET_INTERVAL, interval_ms);
132     return BC_sendFrame(sock_fd, &frame);
133 }
134
135 int BCMP_stopContinuousMode(int sock_fd) {
136     MPFrame_t frame;
137
138     BCMP_initializeFrame(&frame, BCMP_REQ_STOPCONTINUOUS);
139     return BC_sendFrame(sock_fd, &frame);
140 }
141
142 int BCMP_requestValue(int sock_fd, uint64_t *timestamp, double *value) {
143     MPFrame_t frame;
144     int retval;
145
146     BCMP_initializeFrame(&frame, BCMP_REQ_GETVALUE);
147     retval = BC_sendFrame(sock_fd, &frame);
148     if(retval < 0) {
149         #ifdef DEBUG
150         perror("Error occurred at BC_sendFrame()");
151         #endif
152         return -1;
153     }
154
155     retval = BC_recvFrame(sock_fd, &frame);
156     if(retval < 0) {
157         #ifdef DEBUG
158         perror("Error occurred at BC_recvFrame()");
159         #endif
160         return -1;
161     }
162     *timestamp = BC_getArgumentAsUint64(&frame, BCMP_RES_VALUE_OFFSET_TIMESTAMP);
163     *value = BC_getArgumentAsDouble(&frame, BCMP_RES_VALUE_OFFSET_VALUE);
164     return 1;
165 }
166
167 int BCMP_getContinuousValue(int sock_fd, uint64_t *timestamp, double *value) { //TODO
168     MPFrame_t frame;
169     fd_set readset;
170     struct timeval timeout;
171     int retval = 0;
172
173     memset(&timeout, 0x00, sizeof(timeout));
174     FD_ZERO(&readset);
175     FD_SET(sock_fd, &readset);
176     while(select(sock_fd+1, &readset, NULL, NULL, &timeout) > 0) {
177         if(FD_ISSET(sock_fd, &readset)) {
178             retval = BC_recvFrame(sock_fd, &frame);
179             if(retval < 0) {
180                 #ifdef DEBUG
181                 perror("Error occurred at BC_recvFrame()");
182                 #endif
183                 return -1;
184             }
185             *timestamp = BC_getArgumentAsUint64(&frame, BCMP_RES_VALUE_OFFSET_TIMESTAMP);
186             *value = BC_getArgumentAsDouble(&frame, BCMP_RES_VALUE_OFFSET_VALUE);
187         }
188     }
189     return retval;
190 }
191
192 int BCRP_setRelays(int sock_fd, RPArgument_t pattern) {
193     RPFrame_t frame;
194     int retval;
195
196     BCRP_initializeFrame(&frame, BCRP_REQ_SETREGISTER);
197     frame.fields.argBuffer[0] = (uint8_t) pattern;
198     retval = BC_sendFrame(sock_fd, &frame);
199     if(retval < 0) {
200         #ifdef DEBUG
201         perror("Error occurred at BC_sendFrame()");
202         #endif
203         return -1;
204     }
205     return 1;
206 }
207
208 int BCAP_sendVoltage(int sock_fd, double value) {
209     APFrame_t frame;
210     int retval;
211
212     BCAP_initializeFrame(&frame, BCAP_REQ_SETDAC);
```

```
213     //*((double*) frame.fields.argBuffer) = value;          //TODO kann weg neu
214     BC_setArgumentAsDouble(&frame, BCAP_REQ_SETDAC_OFFSET_VALUE, value);
215     retval = BC_sendFrame(sock_fd, &frame);
216     if(retval < 0)
217         return -1;
218     return 1;
219 }
220
221 int BCAP_sendValueToPoti(int sock_fd, float values[MAX_LED]) {
222     APFrame_t frame;
223     int retval;
224
225     BCAP_initializeFrame(&frame, BCAP_REQ_SETLED);
226     *((float*) (frame.fields.argBuffer + BCAP_REQ_LED_RED_OFFSET_VALUE)) = values[LED_R];
227     *((float*) (frame.fields.argBuffer + BCAP_REQ_LED_GREEN_OFFSET_VALUE)) = values[LED_G];
228     *((float*) (frame.fields.argBuffer + BCAP_REQ_LED_BLUE_OFFSET_VALUE)) = values[LED_B];
229     retval = BC_sendFrame(sock_fd, &frame);
230     if(retval < 0)
231         return -1;
232     return 1;
233 }
234
235 /* signal handlers */
236 void timer_handler(int sign) {
237     timerFlag = true;
238 }
```

C.12.3. BC_Client.h

```
1  /*************************************************************
2   * Client process to control the measurement and actuator processes
3   * from modular batcycle process family
4   *
5   * File:    BC_Client.h
6   * Version: 0.2
7   * Date:    2016-04-29
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.2: NR added DAC and battery voltageAlignment function
11  *          0.1: NR first version
12  *
13  *************************************************************/
14
15 #ifndef BC_CLIENT_H_
16 #define BC_CLIENT_H_
17
18 /* uds_client.c */
19 #include <stdlib.h>
20 #include <stdio.h>
21 #include <errno.h>
22 #include <unistd.h>
23 #include <sys/socket.h>     //for sockets
24 #include <sys/un.h>
25 #include <sys/select.h>
26 #include <sys/time.h>       //for signalhandler and timer
27 #include <signal.h>
28 #include <time.h>
29 #include <stdbool.h>        //for boolean
30 //for client server communication
31 #include "BC_Protocol/BC_MeasurementProtocol.h"
32 #include "BC_Protocol/BC_RelayProtocol.h"
33 #include "BC_Protocol/BC_ActuatorProtocol.h"
34 #include "BC_Config/bc_config.h"
35
36 /* defines */
37 #define FILE_INI            "in/bc_init.cfg"       //default init file name
38 #define FILE_PARAM          "in/bc_param.cfg"      //default param file name
39
40 #define OPT_STRING          "i:p:h"     //init file: cycle param file: show help
41
42 #define TIMEOUT_SEC         1           //timeout for socket communication
43 #define TIMEOUT_USEC        0
44
45 /* global variables */
46 extern volatile sig_atomic_t runFlag;
47 extern volatile sig_atomic_t timerFlag;
48
49 /* prototypes */
50 extern void usage(char *progname);
```

```
51 int connectToSock(int *sock_fd, const char *sockFilePath);
52 int BCMP_getControl(int sock_fd);
53 int BCMP_returnControl(int sock_fd);
54 int BCMP_startContinuousMode(int sock_fd, uint32_t interval_ms);
55 int BCMP_stopContinuousMode(int sock_fd);
56 int BCMP_requestValue(int sock_fd, uint64_t *timestamp, double *value);
57 int BCMP_getContinuousValue(int sock_fd, uint64_t *timestamp, double *value);
58 int BCRP_setRelays(int sock_fd, RPArgument_t pattern);
59 int BCAP_sendVoltage(int sock_fd, double value);
60 int BCAP_sendValueToPoti(int sock_fd, float values[MAX_LED]);
61
62 #endif /* BC_CLIENT_H_ */
```

C.12.4. BC_Regulation.c

```
1  /*************************************************************
2   * Client process to control the measurement and actuator processes
3   * from modular batcycle process family
4   *
5   * File:    BC_Regulation.c
6   * Version: 0.2
7   * Date:    2016-04-29
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.2: NR added DAC and battery voltageAlignment function
11  *          0.1: NR first version
12  *
13  *************************************************************/
14
15 #include "BC_Regulation.h"
16
17 /** Function balance the DAC to the battery voltage. First it will be checked the
18  *  battery voltage, then the DAC will be adjusted on same voltage, and then all
19  *  relays will be closed.
20  */
21 int voltageAlignment(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca) {
22     int retval = 0;
23
24     //close VN and VP Relay
25     retval = BCRP_setRelays(sock_fd[SERV_REL], BCRP_RELAY_VN | BCRP_RELAY_VP);
26     usleep(RELAY_TIME_US);   //non busy waiting for relay switching time
27
28     //adjust DAC output on same voltage level as battery
29     retval |= BCMP_requestValue(sock_fd[SERV_VOLT], &bcm->voltage.timestamp, &bcm->voltage.value);
30     bca->dacVoltage = bcm->voltage.value;
31     retval |= BCAP_sendVoltage(sock_fd[SERV_DAC], bca->dacVoltage);       //send to VoltageController process
32     //usleep(500000);
33     sleep(DELAY_RC_TIMECONST);       //wait 1 sec of capacitor voltage alignment
34
35     //close I Relay
36     retval |= BCRP_setRelays(sock_fd[SERV_REL], BCRP_RELAY_ALL);
37     if(retval < 0) {
38         fprintf(stderr, "Failure occurred during initialzing phase at \n"
39                         "battery voltage adaption: %s\n", strerror(errno));
40         return -1;
41     }
42     usleep(RELAY_TIME_US);   //non busy waiting for relay switching time
43     return 1;
44 }
45
46 /** Function keeps alive the measurement sampling frequency during an convalescence
47  *  time for the battery cell.
48  */
49 int convalescenceTime(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp) {
50     time_t start, timeDiff = 0;
51     int convtime = bcp->nominals.conv_time - DELAY_RC_TIMECONST;
52     int retval = 0;
53
54     if(bcp->nominals.conv_time == 0) {        //end condition for 0s convalescence
55         fprintf(stdout, "No convalescence time for battery\n");
56         return 1;
57     }
58
59     //initialize timespec struct
60     fprintf(stdout, "Battery needs %d seconds convalescence time...\n", bcp->nominals.conv_time);
61
62     //disconnect battery cell in current line
63     if(BCRP_setRelays(sock_fd[SERV_REL], BCRP_RELAY_ALL & ~BCRP_RELAY_I) < 0) {      //open I Relay
64         fprintf(stderr, "Failure occurred at set relays: %s\n", strerror(errno));
```

```
65          return -1;
66      }
67
68      //to keep alive the measurement sampling frequency
69      start = time(NULL);
70      while(runFlag && (timeDiff < convtime)) {
71          BCMP_requestValue(sock_fd[SERV_VOLT], &bcm->voltage.timestamp, &bcm->voltage.value);
72          BCMP_requestValue(sock_fd[SERV_CURR], &bcm->current.timestamp, &bcm->current.value);
73          BCMP_getContinuousValue(sock_fd[SERV_TEMP], &bcm->temperature.timestamp, &bcm->temperature.value);
74          retval |= nanosleep(&bcm->period, NULL);                    //nonbusy waiting for one sample period
75          timeDiff = time(NULL) - start;
76      }
77      retval |= voltageAlignment(sock_fd, bcm, bca);
78      if(retval < 0) {
79          fprintf(stderr, "Failure occurred during convalescence time:\n%s\n", strerror(errno));
80          return -1;
81      }
82
83      return 1;
84  }
85
86  /** This function regulates the battery cycle period by an constant
87   *  voltage regulation method (cV). If threshold voltage would cause
88   *  an over max current, the voltage level will be adjusted lower and
89   *  after a fix time interval it will be readjusted. The battery is empty
90   *  or full, when the current is lower than nominal current. At this moment
91   *  an timer counts in "mode time" seconds down and exits this algorithm.
92   *  When mode time is 0, function exits immediately.
93   *  @param1: file descriptor of socket to a peered server
94   *  @param2: pointer of battery cycle measurement struct
95   *  @param3: pointer of battery cycle actutor struct
96   *  @param4: pointer of battery cycle parameter struct
97   *  return: if success 1, else -1
98   */
99  int cV_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp) {
100     uint64_t *tsVolt = &bcm->voltage.timestamp;
101     uint64_t *tsCurr = &bcm->current.timestamp;
102     uint64_t *tsTemp = &bcm->temperature.timestamp;
103     double *act_voltage = &bcm->voltage.value;
104     double *act_current = &bcm->current.value;
105     double *act_temp = &bcm->temperature.value;
106     double threshold = bcp->nominals.thres_voltage;
107     double nom_current = fabs(bcp->nominals.nom_current);
108     int retval = 0;
109     double approxI;             //approximation current in Â[µA] to calculate current limit
110     double resistor = bcp->hw_shunt + bcp->impedance;
111     time_t interval, intstart, start;
112     bool exit = false;
113     int mutex = 1;          //mutual exclusion flag
114
115     retval |= BCMP_requestValue(sock_fd[SERV_VOLT], tsVolt, act_voltage);
116     retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
117
118     //when actual voltage is higher than threshold voltage: discharge
119     if(*act_voltage > threshold) {
120         fprintf(stdout, "discharging in %s mode...\n", STR_MODE_cV);
121         //sign = -1;
122     }
123     //when actual voltage is lower than threshold voltage: charge
124     else if(*act_voltage < threshold) {
125         fprintf(stdout, "charging in %s mode...\n", STR_MODE_cV);
126         //sign = +1;
127     }
128     //when actual voltage is in threshold voltage: exit function
129     else {
130         fprintf(stdout, "%s mode ends. Battery cell is still in threshold range.\n\n", STR_MODE_cV);
131         return 1;
132     }
133     fprintf(stdout, "   %s:\t%.3lf [V]\n   %s:\t%.3lf [uA]\n   %s:\t\t%d [s]\n", STR_THRESVOLT, threshold,
134             STR_NOMCURR, bcp->nominals.nom_current * 1e6, STR_MODETIME, bcp->nominals.mode_time);
135
136     start = time(&intstart);      //get start timestamp in sec, copy to intstart
137     while(runFlag) {
138         BCMP_getContinuousValue(sock_fd[SERV_TEMP], tsTemp, act_temp);
139         retval |= BCMP_requestValue(sock_fd[SERV_VOLT], tsVolt, act_voltage);
140         retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
141         if(retval < 0)
142             break;
143
144         //when current lower than nominal_current and mode_time is over: exit function
145         if(fabs(*act_current) <= nom_current) {
146             //miss the start current, cause the rising current activate the termination count down
147             if((time(NULL) - start) > WARMUP_TIME) {
148                 if(!exit) {         //reinit start time for mode_time countdown
149                     start = time(NULL);
```

```
150                     exit = true;
151                 }
152                 if((time(NULL) - start) >= bcp->nominals.mode_time) {
153                     retval = 1;
154                     fprintf(stdout, "%s mode ends. Battery cell is in threshold range now.\n", STR_MODE_cV);
155                     break;
156                 }
157             }
158         }
159 
160         //when actual current is higher than maximal limits: sink current with raising approxI
161         if((*act_current > bcp->limits.max_lpos) || (*act_current < bcp->limits.max_lneg)) {
162             resistor -= resistor * 0.1;         //reduce resistor value by 10%
163             mutex = 1;
164         }
165         else {
166             //when interval is over, readjust voltage to threshold: time interval
167             interval = time(NULL) - intstart;
168             if(interval >= INTERVAL_CV) {
169                 intstart = time(NULL);
170                 resistor = bcp->hw_shunt + bcp->impedance;  //reset resistance to readjust voltage level
171                 fprintf(stdout, "readjusting voltage level...\n");
172                 mutex = 1;
173             }
174         }
175 
176         //if actual voltage not inside threshold tolerance band: regulate
177         if(*act_voltage < bcp->nominals.thres_voltage - THRESVOLT_TOL) {
178             threshold += VOLTAGE_STEP;
179             mutex = 1;
180         }
181         else if(*act_voltage > bcp->nominals.thres_voltage + THRESVOLT_TOL) {
182             threshold -= VOLTAGE_STEP;
183             mutex = 1;
184         }
185 
186         if(mutex) {
187             approxI = ((threshold - *act_voltage) / resistor);  //get approximated current
188             if(approxI > bcp->limits.max_lpos)
189                 bca->dacVoltage = *act_voltage + bcp->limits.max_lpos * resistor;
190             else if(approxI < bcp->limits.max_lneg)
191                 bca->dacVoltage = *act_voltage + bcp->limits.max_lneg * resistor;
192             else
193                 bca->dacVoltage = threshold;
194             mutex = 0;
195             #ifdef DEBUG
196             fprintf(stdout, "output voltage regulation: %f [V]\n", bca->dacVoltage);
197             #endif
198 
199             //send new voltage to VoltageController
200             retval = BCAP_sendVoltage(sock_fd[SERV_DAC], bca->dacVoltage);
201             if(retval < 0)
202                 break;
203         }
204 
205         nanosleep(&bcm->period, NULL);  //non busy waiting
206     }
207 
208     //if something wrong, throw error message
209     if(retval < 0) {
210         #ifdef DEBUG
211         fprintf(stderr, "Failure occurred in cV regulation: %s\n", strerror(errno));
212         #endif
213         return -1;
214     }
215     fprintf(stdout, "\n");
216     return 1;
217 }
218 
219 /** This function regulates the battery cycle period by an constant
220  *  current regulation method (cC). The battery is empty or full,
221  *  when their threshold voltage is reached or when the mode time is
222  *  over. When mode time is 0, the timer is disabled.
223  *  @param1: file descriptor of socket to a peered server
224  *  @param2: pointer of battery cycle measurement struct
225  *  @param3: pointer of battery cycle actutor struct
226  *  @param4: pointer of battery cycle parameter struct
227  *  return: if success 1, else -1
228  */
229 int cC_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp) {
230     uint64_t *tsVolt = &bcm->voltage.timestamp;
231     uint64_t *tsCurr = &bcm->current.timestamp;
232     uint64_t *tsTemp = &bcm->temperature.timestamp;
233     double *act_voltage = &bcm->voltage.value;
234     double *act_current = &bcm->current.value;
```

```
235     double *act_temp = &bcm->temperature.value;
236     double threshold = bcp->nominals.thres_voltage;
237     double nom_current = fabs(bcp->nominals.nom_current);
238     bool charge = false, discharge = false;
239     double diff_current = 0;
240     int sign;
241     int retval = 0;
242     time_t start, mode_time;
243
244     retval |= BCMP_requestValue(sock_fd[SERV_VOLT], tsVolt, act_voltage);
245     retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
246
247     //when actual voltage is higher than threshold voltage: discharge
248     if(*act_voltage > threshold) {
249         fprintf(stdout, "discharging in %s mode...\n", STR_MODE_cC);
250         discharge = true;
251         sign = -1;
252     }
253     //when actual voltage is lower than threshold voltage: charge
254     else if(*act_voltage < threshold) {
255         fprintf(stdout, "charging in %s mode...\n", STR_MODE_cC);
256         charge = true;
257         sign = +1;
258     }
259     //when actual voltage is in threshold voltage: exit function
260     else {
261         fprintf(stdout, "%s mode ends. Battery cell is still in threshold range.\n\n", STR_MODE_cC);
262         return 1;
263     }
264     fprintf(stdout, "   %s:\t%.3lf [V]\n   %s:\t%.3lf [uA]\n   %s:\t\t%d [s]\n", STR_THRESVOLT, threshold,
265             STR_NOMCURR, bcp->nominals.nom_current * 1e6, STR_MODETIME, bcp->nominals.mode_time);
266
267     nom_current *= sign;
268     start = time(NULL);      //get start timestamp in sec
269     while(runFlag) {
270         BCMP_getContinuousValue(sock_fd[SERV_TEMP], tsTemp, act_temp);
271         retval |= BCMP_requestValue(sock_fd[SERV_VOLT], tsVolt, act_voltage);
272         retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
273         if(retval < 0)
274             break;
275
276         //when temperature is out of limits: exit
277         //TODO
278
279         //when actual voltage is at threshold voltage: exit function
280         if(((*act_voltage >= threshold) && charge) || ((*act_voltage <= threshold) && discharge)) {
281             fprintf(stdout, "%s mode ends. Voltage limit of battery cell is reached.\n", STR_MODE_cC);
282                 break;
283         }
284
285         //when mode_time is over: exit function
286         if(bcp->nominals.mode_time > 0) {
287             mode_time = time(NULL) - start;
288             if(mode_time > bcp->nominals.mode_time) {
289                 fprintf(stdout, "%s mode ends. Mode time is over.\n", STR_MODE_cC);
290                 break;
291             }
292         }
293
294         diff_current = fabs(*act_current - nom_current);
295
296         //when actual current is higher than nominal current: sink
297         if(*act_current > (nom_current + THRESCURR_TOL))
298             sign = -1;
299         //when actual current is lower than nominal current: rise
300         else if(*act_current < (nom_current - THRESCURR_TOL))
301             sign = +1;
302         else
303             continue;
304
305         //regulation condition cases (regulation parameters are an experimental found)
306         if(diff_current >  CURRENT_DIFF_FAC_4 * THRESCURR_TOL)
307             bca->dacVoltage += sign * STEP_FAC_4 * VOLTAGE_STEP;
308         else if(diff_current > CURRENT_DIFF_FAC_3 * THRESCURR_TOL)
309             bca->dacVoltage += sign * STEP_FAC_3 * VOLTAGE_STEP;
310         else if(diff_current > CURRENT_DIFF_FAC_2 * THRESCURR_TOL)
311             bca->dacVoltage += sign * STEP_FAC_2 * VOLTAGE_STEP;
312         else if(diff_current > CURRENT_DIFF_FAC_1 * THRESCURR_TOL)
313             bca->dacVoltage += sign * STEP_FAC_1 * VOLTAGE_STEP;
314         retval = BCAP_sendVoltage(sock_fd[SERV_DAC], bca->dacVoltage);
315         if(retval < 0)
316             break;
317
318         nanosleep(&bcm->period, NULL);      //non busy waiting
319     }
```

```
320
321     //restore actual current in the bcm struct
322     retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
323
324     //if something wrong, throw error message
325     if(retval < 0) {
326         fprintf(stderr, "Failure occurred in cC regulation: %s\n", strerror(errno));
327         return -1;
328     }
329     fprintf(stdout, "\n");
330     return 1;
331 }
332
333 /** First this function regulates the battery cycle period by an constant
334  *  current (cC) and when threshold voltage is reached by an constant voltage
335  *  (cV). The mode time is only possible for cV regulation. When mode time
336  *  is 0, the timer is disabled.
337  *  @param1: file descriptor of socket to a peered server
338  *  @param2: pointer of battery cycle measurement struct
339  *  @param3: pointer of battery cycle actutor struct
340  *  @param4: pointer of battery cycle parameter struct
341  *  return: if success 1, else -1
342  */
343 int cCcV_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp) {
344     uint32_t cntdown;
345     int *temp1;
346     int retval;
347
348     /*begin with cC mode*/
349     fprintf(stdout, "%s mode begins with an \n", STR_MODE_cCcV);
350     temp1 = &bcp->nominals.mode_time;     //work around to disable mode time for cC_Regulation
351     cntdown = *temp1;
352     *temp1 = 0;     //no countdown for cC mode
353     retval = cC_Regulation(sock_fd, bcm, bca, bcp);
354     if(retval < 0)
355         return -1;
356
357     /*change in cV mode*/
358     uint64_t *tsVolt = &bcm->voltage.timestamp;
359     uint64_t *tsCurr = &bcm->current.timestamp;
360     uint64_t *tsTemp = &bcm->temperature.timestamp;
361     double *act_voltage = &bcm->voltage.value;
362     double *act_current = &bcm->current.value;
363     double *act_temp = &bcm->temperature.value;
364     double approxI;             //approximation current in Â[µA] to calculate current limit
365     double resistor = bcp->hw_shunt + bcp->impedance;
366     time_t interval, intstart, start;
367     bool exit = false;
368     int mutex = 1;          //mutual exclusion flag
369     double threshold = bca->dacVoltage;         //copy actual DAC value to threshold voltage
370     double nom_current = (fabs(bcp->nominals.nom_current) * 0.1);  //take 10% of the nominal current for exit condition in cCcV mode
371
372     nom_current = nom_current < END_CURRENT_CV ? END_CURRENT_CV : nom_current;
373      *temp1 = cntdown;     //activate mode time for cV_Regulation
374
375     fprintf(stdout, "switching in %s mode...\n", STR_MODE_cV);
376     fprintf(stdout, "   %s:\t%.3lf [V]\n   %s:\t%.3lf [uA]\n   %s:\t\t%d [s]\n", STR_THRESVOLT, bcp->nominals.thres_voltage,
377             STR_NOMCURR, nom_current * 1e6, STR_MODETIME, bcp->nominals.mode_time);
378
379     start = time(&intstart);      //get start timestamp in sec, copy to intstart
380     while(runFlag) {
381         BCMP_getContinuousValue(sock_fd[SERV_TEMP], tsTemp, act_temp);
382         retval |= BCMP_requestValue(sock_fd[SERV_VOLT], tsVolt, act_voltage);
383         retval |= BCMP_requestValue(sock_fd[SERV_CURR], tsCurr, act_current);
384         if(retval < 0)
385             break;
386
387         //when current lower than nominal_current and mode_time is over: exit function
388         if(fabs(*act_current) <= nom_current) {
389             if((time(NULL) - start) > WARMUP_TIME) {    //miss the start current
390                 if(!exit) {         //reinit start time for mode_time countdown
391                     start = time(NULL);
392                     exit = true;
393                 }
394                 if((time(NULL) - start) >= bcp->nominals.mode_time) {
395                     retval = 1;
396                     fprintf(stdout, "%s mode ends. Battery cell is in threshold range now.\n", STR_MODE_cV);
397                     break;
398                 }
399             }
400         }
401
402         //when actual current is higher than maximal limits: sink current
403         if((*act_current > bcp->limits.max_Ipos) || (*act_current < bcp->limits.max_Ineg)) {
404             resistor -= resistor * 0.1;         //reduce resistor value by 10%
```

```
405             mutex = 1;
406         }
407         else {
408             //when interval is over, readjust voltage to threshold: time interval
409             interval = time(NULL) - intstart;
410             if(interval >= INTERVAL_CV) {
411                 intstart = time(NULL);
412                 resistor = bcp->hw_shunt + bcp->impedance;  //reset resistance to readjust voltage level
413                 #ifdef DEBUG
414                 fprintf(stdout, "readjusting voltage level...\n");
415                 #endif
416                 mutex = 1;
417             }
418         }
419
420         //if actual voltage not inside threshold tolerance band: regulate
421         if(*act_voltage < bcp->nominals.thres_voltage - THRESVOLT_TOL) {
422             threshold += VOLTAGE_STEP;
423             mutex = 1;
424         }
425         else if(*act_voltage > bcp->nominals.thres_voltage + THRESVOLT_TOL) {
426             threshold -= VOLTAGE_STEP;
427             mutex = 1;
428         }
429
430         if(mutex) {
431             approxI = ((threshold - *act_voltage) / resistor);  //get approximated current
432             if(approxI > bcp->limits.max_Ipos)
433                 bca->dacVoltage = *act_voltage + bcp->limits.max_Ipos * resistor;
434             else if(approxI < bcp->limits.max_Ineg)
435                 bca->dacVoltage = *act_voltage + bcp->limits.max_Ineg * resistor;
436             else
437                 bca->dacVoltage = threshold;
438             mutex = 0;
439             #ifdef DEBUG
440             fprintf(stdout, "output voltage regulation: %f [V]\n", bca->dacVoltage);
441             #endif
442
443             //send new voltage to VoltageController
444             retval = BCAP_sendVoltage(sock_fd[SERV_DAC], bca->dacVoltage);
445             if(retval < 0)
446                 break;
447         }
448
449         nanosleep(&bcm->period, NULL);  //non busy waiting
450     }
451
452     //if something wrong, throw error message
453     if(retval < 0) {
454         #ifdef DEBUG
455         fprintf(stderr, "Failure occurred in cV regulation: %s\n", strerror(errno));
456         #endif
457         return -1;
458     }
459     fprintf(stdout, "\n");
460     return 1;
461 }
```

C.12.5. BC_Regulation.h

```
1  /*************************************************************
2   * Client process to control the measurement and actuator processes
3   * from modular batcycle process family
4   *
5   * File:    BC_Regulation.h
6   * Version: 0.2
7   * Date:    2016-04-29
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.2: NR added DAC and battery voltageAlignment function
11  *          0.1: NR first version
12  *
13  *************************************************************/
14
15 #ifndef BC_REGULATION_H_
16 #define BC_REGULATION_H_
17
18 /* Includes */
19 #include "BC_Client.h"
```

```
20
21  /* sampling periods */
22  #define MEASURE_PERIOD_NS   (100e6)     //100ms quasi equidistant current and voltage sampling period
23  #define TEMP_PERIOD_MS      (10e3)      //10s quasi equidistant temperature sampling period
24
25  /* time constants */
26  #define DELAY_RC_TIMECONST  2           //2 sec delay for RC time constant in charge circuit
27  #define RELAY_TIME_US       (200e3)     //relays need a time to switch
28
29  /* regulation parameters */
30  #define THRESVOLT_TOL       0.001       //upside voltage threshold tolerance in [V]
31  #define THRESCURR_TOL       (0.1e-6)   //Â0.1µA upside current threshold tolerance in [A]
32  #define INTERVAL_CV         60          //readjusting interval for cV_Regulation() in [sec]
33  #define WARMUP_TIME         10          //least time to stay in cV_Regulation at beginning for current swing [sec]
34  #define VOLTAGE_STEP        0.00005     //Â50µV maximal voltage step for VoltageController in [V]
35  #define END_CURRENT_CV      (10e-6)     //minimal current as terminating condition for cCcV mode
36
37  #define CURRENT_DIFF_FAC_4  800
38  #define CURRENT_DIFF_FAC_3  200
39  #define CURRENT_DIFF_FAC_2  20
40  #define CURRENT_DIFF_FAC_1  1
41  #define STEP_FAC_4          400
42  #define STEP_FAC_3          100
43  #define STEP_FAC_2          10
44  #define STEP_FAC_1          1
45
46  //battery cycle measurement value set
47  typedef struct bc_measured_value_set {
48      struct bcm_voltage_set {
49          uint64_t timestamp;
50          double value;
51      } voltage;
52      struct bcm_current_set {
53          uint64_t timestamp;
54          double value;
55      } current;
56      struct bcm_temperature_set {
57          uint64_t timestamp;
58          double value;
59      } temperature;
60      struct timespec period;         //time struct for sample period
61  } BCMValSet_t;
62
63  //battery cycle actuator value set
64  typedef struct bc_actual_value_set {
65      double dacVoltage;
66      float act_RGBlight[MAX_LED];
67      uint8_t relayPattern;
68  } BCAValSet_t;
69
70  int cV_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);
71  int cC_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);
72  int cCcV_Regulation(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);
73
74  int voltageAlignment(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca);
75  int convalescenceTime(int sock_fd[MAX_SERVERS], BCMValSet_t *bcm, BCAValSet_t *bca, BCParamTree_t *bcp);
76
77  #endif /* BC_REGULATION_H_ */
```

C.12.6. bc_config.c

```
1   /*************************************************************
2    * Data structures and functions for config file parsing.
3    *
4    * File:    BC_Config/bc_config.c
5    * Version: 0.1
6    * Date:    2016-03-24
7    * Author:  Nico Rieckmann
8    *
9    * Changes: 0.1: NR first version.
10   *
11   *************************************************************/
12
13  #include "bc_config.h"
14
15  /*****************************************************************************************************************
16   *  This function opens an config file.
17   */
18  FILE* openConfigFile(config_t *cfg, const char* filepath) {
```

```
19      FILE *fptr;
20
21      config_init(cfg);
22      fptr = fopen(filepath, "r");            //open param file as only readable
23      if(fptr == NULL) {
24          fprintf(stderr, "%s is missed: %s\n", filepath, strerror(errno));
25          return fptr;
26      }
27      return fptr;
28  }
29
30  /*****************************************************************************************************************
31   *  This function close an config file.
32   */
33  void closeConfigFile(config_t *cfg, FILE* fptr) {
34      fclose(fptr);
35      config_destroy(cfg);
36  }
37
38  /*****************************************************************************************************************
39   *  This function reads the init file and checks the syntax and if all values are plausible.
40   */
41  int readInitFile(config_t *cfg, BCParamTree_t *data) {
42      config_setting_t *inputGroup;
43      config_setting_t *bchwGroup;
44      int errocc = 0;             //error occurrence counter
45
46      if(data->initfile.fptr == NULL)    //valid file existence
47          return -1;
48
49      //Read the file. If there is an error, report it and exit
50      if(config_read(cfg, data->initfile.fptr) <= CONFIG_FALSE) {
51          fprintf(stderr, "File: %s, line: %d - %s\n", config_error_file(cfg), config_error_line(cfg), config_error_text(cfg));
52          return -1;
53      }
54
55      //check versions
56      float progvers = PROG_VERSION_INIT;
57      float* filevers = &data->initfile.version;              //workaround to write a const float value
58      if(checkFileVersion(cfg, &progvers, filevers) < 0)
59          return -1;
60
61      //get inputgroup elements
62      inputGroup = config_lookup(cfg, STR_INPUTGROUP);
63      if(inputGroup == NULL) {
64          fprintf(stderr, "group: '%s' missed in file: %s\n", STR_INPUTGROUP, data->initfile.filepath);
65          return -1;
66      }
67      else {
68          config_setting_t* Sockets;
69
70          //catch socket paths
71          Sockets = config_setting_get_member(inputGroup, STR_SOCKETS);
72          if(Sockets == NULL) {
73              fprintf(stderr, "group: '%s' missed in group: '%s' from file: %s\n",
74                      STR_SOCKETS, STR_INPUTGROUP, data->initfile.filepath);
75              return -1;
76          }
77          else { //get sockI path
78              if(config_setting_lookup_string(Sockets, STR_SOCKI, (const char**) &(data->sockpaths[SERV_CURR])) <= 0) {
79                  fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
80                          STR_SOCKI, STR_SOCKETS, data->initfile.filepath);
81                  errocc |= -1;
82              }   //get sockV path
83              if(config_setting_lookup_string(Sockets, STR_SOCKV, (const char**) &(data->sockpaths[SERV_VOLT])) <= 0) {
84                  fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
85                          STR_SOCKV, STR_SOCKETS, data->initfile.filepath);
86                  errocc |= -1;
87              }   //get sockT path
88              if(config_setting_lookup_string(Sockets, STR_SOCKT, (const char**) &(data->sockpaths[SERV_TEMP])) <= 0) {
89                  fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
90                          STR_SOCKT, STR_SOCKETS, data->initfile.filepath);
91                  errocc |= -1;
92              }   //get sockD path
93              if(config_setting_lookup_string(Sockets, STR_SOCKD, (const char**) &(data->sockpaths[SERV_DAC])) <= 0) {
94                  fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
95                          STR_SOCKD, STR_SOCKETS, data->initfile.filepath);
96                  errocc |= -1;
97              }   //get sockP path
98              if(config_setting_lookup_string(Sockets, STR_SOCKP, (const char**) &(data->sockpaths[SERV_POTI])) <= 0) {
99                  fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
100                         STR_SOCKP, STR_SOCKETS, data->initfile.filepath);
101                 errocc |= -1;
102             }   //get sockR path
103             if(config_setting_lookup_string(Sockets, STR_SOCKR, (const char**) &(data->sockpaths[SERV_REL])) <= 0) {
```

```
104                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
105                         STR_SOCKR, STR_SOCKETS, data->initfile.filepath);
106                 errocc |= -1;
107             }
108         }
109     }
110
111     //get BCHWgroup elements
112     bchwGroup = config_lookup(cfg, STR_BCHWGROUP);
113     if(bchwGroup == NULL) {
114         fprintf(stderr, "group: '%s' missed in file: %s\n", STR_BCHWGROUP, data->initfile.filepath);
115         return -1;
116     }
117     else {  //get shunt value
118         if(config_setting_lookup_float(bchwGroup, STR_SHUNT, (double*) &data->hw_shunt) <= 0) {
119             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
120                     STR_SHUNT, STR_BCHWGROUP, data->initfile.filepath);
121             errocc |= -1;
122         }
123     }
124
125     //close and return
126     if(errocc != 0)      //if any failure occurred
127         return -1;
128     return 1;
129 }
130
131 /*****************************************************************************************************************
132  *  This function reads the param file and checks the syntax and if all values are plausible.
133  */
134 int readParamFile(config_t *cfg, BCParamTree_t *data, config_setting_t **cycleList) {
135     config_setting_t *batGroup;
136     config_setting_t *limitGroup;
137     config_setting_t *cycleGroup;
138     int errocc = 0;     //error occurrence counter
139
140     if(data->paramfile.fptr == NULL)    //valid file existence
141         return -1;
142
143     //Read the file. If there is an error, report it and exit
144     if(config_read(cfg, data->paramfile.fptr) <= CONFIG_FALSE) {
145         fprintf(stderr, "File: %s, line: %d - %s\n", config_error_file(cfg), config_error_line(cfg), config_error_text(cfg));
146         return -1;
147     }
148
149     //check versions
150     float progvers = PROG_VERSION_PARAM;
151     float* filevers = &data->paramfile.version;             //workaround to write a const float value
152     if(checkFileVersion(cfg, &progvers, filevers) < 0) {
153         return -1;
154     }
155
156     //get batterygroup
157     batGroup = config_lookup(cfg, STR_BATGROUP);
158     if(batGroup == NULL) {
159         fprintf(stderr, "group: '%s' missed in file: %s\n", STR_BATGROUP, data->paramfile.filepath);
160         return -1;
161     }
162     else {
163         //get battery type
164         if(config_setting_lookup_string(batGroup, STR_BATTYPE, &data->bat_type) <= 0) {
165             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
166                     STR_BATTYPE, STR_BATGROUP, data->paramfile.filepath);
167             errocc |= -1;
168         }
169         if(config_setting_lookup_float(batGroup, STR_BATRES, (double*) &data->impedance) <= 0) {
170             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
171                     STR_BATRES, STR_BATGROUP, data->initfile.filepath);
172             errocc |= -1;
173         }
174     }
175
176     //get limitgroup
177     limitGroup = config_lookup(cfg, STR_LIMITGROUP);
178     if(limitGroup == NULL) {
179         fprintf(stderr, "group: '%s' missed in file: %s\n", STR_LIMITGROUP, data->paramfile.filepath);
180         return -1;
181     }
182     else {
183         double* dummyDouble;
184
185         //get maxVolt value
186         if(config_setting_lookup_float(limitGroup, STR_MAXVOLT, (double*) &data->limits.max_Volt) <= 0) {
187             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
188                     STR_MAXVOLT, STR_LIMITGROUP, data->paramfile.filepath);
```

```
189             errocc |= -1;
190         }
191         //get minVolt value
192         if(config_setting_lookup_float(limitGroup, STR_MINVOLT, (double*) &data->limits.min_Volt) <= 0) {
193             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
194                     STR_MINVOLT, STR_LIMITGROUP, data->paramfile.filepath);
195             errocc |= -1;
196         }
197         //get maxTemp value
198         if(config_setting_lookup_float(limitGroup, STR_MAXTEMP, (double*) &data->limits.max_Temp) <= 0) {
199             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
200                     STR_MAXTEMP, STR_LIMITGROUP, data->paramfile.filepath);
201             errocc |= -1;
202         }
203         //get minTemp value
204         if(config_setting_lookup_float(limitGroup, STR_MINTEMP, (double*) &data->limits.min_Temp) <= 0) {
205             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
206                     STR_MINTEMP, STR_LIMITGROUP, data->paramfile.filepath);
207             errocc |= -1;
208         }
209         //get maxChargeCurrent value
210         if(config_setting_lookup_float(limitGroup, STR_MAXCURRCHAR, (double*) &data->limits.max_Ipos) <= 0) {
211             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
212                     STR_MAXCURRCHAR, STR_LIMITGROUP, data->paramfile.filepath);
213             errocc |= -1;
214         }
215         //get maxDischargeCurrent value
216         if(config_setting_lookup_float(limitGroup, STR_MAXCURRDIS, (double*) &data->limits.max_Ineg) <= 0) {
217             fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
218                     STR_MAXCURRDIS, STR_LIMITGROUP, data->paramfile.filepath);
219             errocc |= -1;
220         }
221         dummyDouble = &data->limits.max_Ineg;     //negate discharge current
222         *dummyDouble = - fabs((*dummyDouble)) / FAC_MICRO;  //convert discharge max current from ÂµA in A
223
224         dummyDouble = &data->limits.max_Ipos;
225         *dummyDouble = fabs((*dummyDouble)) / FAC_MICRO;   //convert charge max current from ÂµA in A
226     }
227
228     //get cycleGroup
229     cycleGroup = config_lookup(cfg, STR_CYCLEGROUP);
230     if(cycleGroup == NULL) {
231         fprintf(stderr, "group: '%s' missed in group: '%s' from file: %s\n",
232                 STR_CYCLEGROUP, STR_LIMITGROUP, data->paramfile.filepath);
233         return -1;
234     }
235     else {  //get cycleGroup elements
236         config_setting_t *startPhase;
237         config_setting_t *cyclePhase;
238
239         //get startphase element
240         startPhase = config_setting_get_member(cycleGroup, STR_STARTPHASE);
241         if(startPhase == NULL) {
242             fprintf(stderr, "group: '%s' missed in group: '%s' from file: %s\n",
243                     STR_STARTPHASE, STR_CYCLEGROUP, data->paramfile.filepath);
244             return -1;
245         }
246         else {
247             double *dummyDouble;
248
249             //get threshold_voltage value
250             if(config_setting_lookup_float(startPhase, STR_THRESVOLT, (double*) &data->nominals.thres_voltage) <= 0) {
251                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
252                         STR_THRESVOLT, STR_STARTPHASE, data->paramfile.filepath);
253                 errocc |= -1;
254             }   //get nominal_current value
255             if(config_setting_lookup_float(startPhase, STR_NOMCURR, (double*) &data->nominals.nom_current) <= 0) {
256                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
257                         STR_NOMCURR, STR_STARTPHASE, data->paramfile.filepath);
258                 errocc |= -1;
259             }
260             dummyDouble = &data->nominals.nom_current;     //convert nominal current from ÂµA in A
261             *dummyDouble = (*dummyDouble) / FAC_MICRO;
262             //get convalescence_time value
263             if(config_setting_lookup_int(startPhase, STR_CONVTIME, (int*) &data->nominals.conv_time) <= 0) {
264                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
265                         STR_CONVTIME, STR_STARTPHASE, data->paramfile.filepath);
266                 errocc |= -1;
267             }   //get led_R value
268             if(config_setting_lookup_float(startPhase, STR_LED_R, (double*) &data->nominals.nom_RGBlight[0]) <= 0) {
269                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
270                         STR_LED_R, STR_STARTPHASE, data->paramfile.filepath);
271                 errocc |= -1;
272             }   //get led_G value
273             if(config_setting_lookup_float(startPhase, STR_LED_G, (double*) &data->nominals.nom_RGBlight[1]) <= 0) {
```

```
274                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
275                         STR_LED_G, STR_STARTPHASE, data->paramfile.filepath);
276                 errocc |= -1;
277             }   //get led_B value
278             if(config_setting_lookup_float(startPhase, STR_LED_B, (double*) &data->nominals.nom_RGBlight[2]) <= 0) {
279                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
280                         STR_LED_B, STR_STARTPHASE, data->paramfile.filepath);
281                 errocc |= -1;
282             }   //get mode string
283             if(config_setting_lookup_string(startPhase, STR_MODE, (const char**) &data->nominals.charge_mode) <= 0) {
284                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
285                         STR_MODE, STR_STARTPHASE, data->paramfile.filepath);
286                 errocc |= -1;
287             }   //get mode_time value
288             if(config_setting_lookup_int(startPhase, STR_MODETIME, (int*) &data->nominals.mode_time) <= 0) {
289                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
290                         STR_MODETIME, STR_STARTPHASE, data->paramfile.filepath);
291                 errocc |= -1;
292             }
293
294             //close and return
295             if(errocc != 0)         //if any failure occurred
296                 return -1;
297
298             //plausibility check
299             errocc = checkCycleBlockLimits(data, STR_STARTPHASE, 1, data->nominals.thres_voltage,
300                                            data->nominals.nom_current, data->nominals.conv_time,
301                                            (double*) data->nominals.nom_RGBlight,
302                                            (char*) data->nominals.charge_mode, data->nominals.mode_time);
303         }
304
305         //get cyclePhase group
306         cyclePhase = config_setting_get_member(cycleGroup, STR_CYCLEPHASE);
307         if(cyclePhase == NULL) {
308             fprintf(stderr, "group: '%s' missed in group: '%s' from file: %s\n",
309                     STR_CYCLEPHASE, STR_CYCLEGROUP, data->paramfile.filepath);
310             return -1;
311         }
312         else {
313             int cnt = 0;
314
315             //get 'cycles' element
316             if(config_setting_lookup_int(cyclePhase, STR_CYCLES, (int*) &data->cycles) <= 0) {
317                 fprintf(stderr, "field: '%s' missed in group: '%s' from file: %s\n",
318                         STR_CYCLES, STR_CYCLEPHASE, data->paramfile.filepath);
319                 errocc |= -1;
320             }
321
322             //get cycle list
323             *cycleList = config_setting_get_member(cyclePhase, STR_CYCLELIST);
324             if(*cycleList == NULL) {
325                 fprintf(stderr, "list: '%s' missed in group: '%s' from file: %s\n",
326                         STR_CYCLELIST, STR_CYCLEPHASE, data->paramfile.filepath);
327                 return -1;
328             }
329
330             cnt = config_setting_length(*cycleList);
331             if(cnt < 2) {
332                 fprintf(stderr, "number of '%s' members must be least %d.\n", STR_CYCLELIST, 2);        //TODO 2 as define
333                 return -1;
334             }
335             else  {
336                 //only check the full cyclelist for completeness, cause later errors would be fatal
337                 double thres_volt, nom_current, ledRGB[3];       //dummy variables
338                 int conv_time, mode_time;
339                 char* mode;
340                 int i;
341
342                 for(i = 0; i < cnt; ++i) {
343                     config_setting_t *valset = config_setting_get_elem(*cycleList, i);
344
345                     //check if threshold_voltage value exists
346                     if(config_setting_lookup_float(valset, STR_THRESVOLT, (double*) &thres_volt) <= 0) {
347                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
348                                 STR_THRESVOLT, STR_CYCLELIST, data->paramfile.filepath);
349                         errocc |= -1;
350                     }   //check if nominal_current value exists
351                     if(config_setting_lookup_float(valset, STR_NOMCURR, (double*) &nom_current) <= 0) {
352                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
353                                 STR_NOMCURR, STR_CYCLELIST, data->paramfile.filepath);
354                         errocc |= -1;
355                     }
356                     nom_current /= FAC_MICRO;       //convert nominal current from ÂµA in A
357                     //check if convalescence value exists
358                     if(config_setting_lookup_int(valset, STR_CONVTIME, (int*) &conv_time) <= 0) {
```

```
359                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
360                                 STR_CONVTIME, STR_CYCLELIST, data->paramfile.filepath);
361                         errocc |= -1;
362                     }   //check if led_R value exists
363                     if(config_setting_lookup_float(valset, STR_LED_R, (double*) &ledRGB[0]) <= 0) {
364                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
365                                 STR_LED_R, STR_CYCLELIST, data->paramfile.filepath);
366                         errocc |= -1;
367                     }   //check if led_G value exists
368                     if(config_setting_lookup_float(valset, STR_LED_G, (double*) &ledRGB[1]) <= 0) {
369                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
370                                 STR_LED_G, STR_CYCLELIST, data->paramfile.filepath);
371                         errocc |= -1;
372                     }   //check if led_B value exists
373                     if(config_setting_lookup_float(valset, STR_LED_B, (double*) &ledRGB[2]) <= 0) {
374                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
375                                 STR_LED_B, STR_CYCLELIST, data->paramfile.filepath);
376                         errocc |= -1;
377                     }   //check if mode string exists
378                     if(config_setting_lookup_string(valset, STR_MODE, (const char**) &mode) <= 0) {
379                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
380                                 STR_MODE, STR_CYCLELIST, data->paramfile.filepath);
381                         errocc |= -1;
382                     }   //check if mode_time value exists
383                     if(config_setting_lookup_int(valset, STR_MODETIME, (int*) &mode_time) <= 0) {
384                         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
385                                 STR_MODETIME, STR_CYCLELIST, data->paramfile.filepath);
386                         errocc |= -1;
387                     }
388
389                     //exit point
390                     if(errocc != 0)      //if any failure occurred
391                         return -1;
392
393
394                     //check plausibility
395                     errocc = checkCycleBlockLimits(data, STR_CYCLELIST, i + 1, thres_volt, nom_current,
396                                                    conv_time, ledRGB, mode, mode_time);
397                 }
398             }
399         }
400     }
401
402     //exit point
403     if(errocc != 0)      //if any failure occurred
404         return -1;
405     return 1;
406 }
407
408 /*****************************************************************************************************************
409  *  This function fetches the next parameter set from cyclelist in param file.
410  *  Return value is the amount of pending sets.
411  */
412 int getNextParamSet(config_setting_t *cycleList, BCParamTree_t *data, int reset) {
413     static int cnt;        //counter to remember previous cyclelist element
414     static int max_elem;
415     config_setting_t *valset;
416     double *dummyDouble;
417
418     if(reset < 0)       //condition to reset listpointer at begin
419         cnt = 0;
420     max_elem = config_setting_length(cycleList);     //get length of cyclelist
421     if(max_elem < 2) {
422         fprintf(stderr, "number of '%s' members must be least %d.\n", STR_CYCLELIST, 2);
423         return -1;
424     }
425
426     if(cnt >= max_elem)     //exit condition
427        return 0;
428
429     valset = config_setting_get_elem(cycleList, cnt);
430     cnt++;
431
432     //get threshold_voltage value
433     if(config_setting_lookup_float(valset, STR_THRESVOLT, (double*) &data->nominals.thres_voltage) <= 0) {
434         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
435                 STR_THRESVOLT, STR_CYCLELIST, data->paramfile.filepath);
436         return -1;
437     }   //get nominal_current value
438     if(config_setting_lookup_float(valset, STR_NOMCURR, (double*) &data->nominals.nom_current) <= 0) {
439         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
440                 STR_NOMCURR, STR_CYCLELIST, data->paramfile.filepath);
441         return -1;
442     }
443     dummyDouble = &data->nominals.nom_current;     //convert nominal current from ÂµA in A
```

```
444     *dummyDouble = (*dummyDouble) / FAC_MICRO;
445     //get convalescence time value
446     if(config_setting_lookup_int(valset, STR_CONVTIME, (int*) &data->nominals.conv_time) <= 0) {
447         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
448                 STR_CONVTIME, STR_CYCLELIST, data->paramfile.filepath);
449         return -1;
450     }   //get led_R value
451     if(config_setting_lookup_float(valset, STR_LED_R, (double*) &data->nominals.nom_RGBlight[LED_R]) <= 0) {
452         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
453                 STR_LED_R, STR_CYCLELIST, data->paramfile.filepath);
454         return -1;
455     }   //get led_G value
456     if(config_setting_lookup_float(valset, STR_LED_G, (double*) &data->nominals.nom_RGBlight[LED_G]) <= 0) {
457         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
458                 STR_LED_G, STR_CYCLELIST, data->paramfile.filepath);
459         return -1;
460     }   //get led_B value
461     if(config_setting_lookup_float(valset, STR_LED_B, (double*) &data->nominals.nom_RGBlight[LED_B]) <= 0) {
462         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
463                 STR_LED_B, STR_CYCLELIST, data->paramfile.filepath);
464         return -1;
465     }   //get mode string
466     if(config_setting_lookup_string(valset, STR_MODE, (const char**) &data->nominals.charge_mode) <= 0) {
467         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
468                 STR_MODE, STR_CYCLELIST, data->paramfile.filepath);
469         return -1;
470     }   //get mode_time value
471     if(config_setting_lookup_int(valset, STR_MODETIME, (int*) &data->nominals.mode_time) <= 0) {
472         fprintf(stderr, "field: '%s' missed in list: '%s' from file: %s\n",
473                 STR_MODETIME, STR_CYCLELIST, data->paramfile.filepath);
474         return -1;
475     }
476
477     return max_elem - cnt;      //return the amount of future sets
478 }
479
480 /*****************************************************************************************************************
481  *  This function checks the versions from init and param file against the program software version.
482  */
483 int checkFileVersion(config_t *cfg, float *progVersion, float *fileVersion) {
484     double dummyDouble;
485
486     if(config_lookup_float(cfg, STR_VERSION, &dummyDouble) <= CONFIG_FALSE) {
487         fprintf(stderr, "field: '%s' missed from file: %s\n", STR_VERSION, cfg->filenames[0]);
488         return -1;
489     }
490     *fileVersion = (float) dummyDouble;         //double to float cast
491
492     //when file version is higher than program version: corruption
493     if(*fileVersion > *progVersion) {
494         fprintf(stderr, "file version %1.1f is higher than program can read. \nLeast compatible version"
495                 " of %s must be v%1.1f \n", *fileVersion, cfg->filenames[0], *progVersion);
496         return -1;
497     }
498     return 1;
499 }
500
501 /*****************************************************************************************************************
502  *  This function checks if cycle parameters within limit ranges.
503  */
504 int checkCycleBlockLimits(BCParamTree_t *data, char* group, int index, double thres_volt, double nom_curr,
505                           int conv_time, double ledRGB[MAX_LED], char* mode, int mode_time) {
506     int errocc = 0;
507
508     if(thres_volt > data->limits.max_Volt || thres_volt < data->limits.min_Volt) {
509     fprintf(stderr, "'%s' in group: '%s[%d]' is not inside limit range of '%s' and '%s'\n",
510             STR_THRESVOLT, group, index, STR_MINVOLT, STR_MAXVOLT);
511     errocc |= -1;
512     }
513     if(nom_curr > data->limits.max_Ipos || nom_curr < data->limits.max_Ineg) {
514         fprintf(stderr, "'%s' in group: '%s'[%d] is not inside limit range of '%s' and '%s'\n",
515                 STR_NOMCURR, group, index, STR_MAXCURRDIS, STR_MAXCURRCHAR);
516         errocc |= -1;
517     }
518     if(conv_time < 0) {
519         fprintf(stderr, "'%s' in group: '%s'[%d] is negative\n", STR_CONVTIME, group, index);
520         errocc |= -1;
521     }
522     if(ledRGB[LED_R] < 0.0 || ledRGB[LED_R] > 100.0) {  //led value reaches from 0% to 100%
523         fprintf(stderr, "'%s' in group: '%s'[%d] is not inside limit range of %.1f %% and %3.1f %%\n",
524                 STR_LED_R, group, index, 0.0, 100.0);
525         errocc |= -1;
526     }
527     if(ledRGB[LED_G] < 0.0 || ledRGB[LED_G] > 100.0) {  //led value reaches from 0% to 100%
528         fprintf(stderr, "'%s' in group: '%s'[%d] is not inside limit range of %.1f %% and %3.1f %%\n",
```

```
529                 STR_LED_G, group, index, 0.0, 100.0);
530         errocc |= -1;
531     }
532     if(ledRGB[LED_B] < 0.0 || ledRGB[LED_B] > 100.0) {  //led value reaches from 0% to 100%
533         fprintf(stderr, "'%s' in group: '%s'[%d] is not inside limit range of %.1f %% and %3.1f %%\n",
534                 STR_LED_B, group, index, 0.0, 100.0);
535         errocc |= -1;
536     }
537     if(strcmp(mode, STR_MODE_cCcV) && strcmp(mode, STR_MODE_cC) && strcmp(mode, STR_MODE_cV)) {
538         fprintf(stderr, "'%s' in group: '%s'[%d] is not \"%s\", \"%s\" or \"%s\"\n",
539                 STR_MODE, group, index, STR_MODE_cCcV, STR_MODE_cC, STR_MODE_cV);
540         errocc |= -1;
541     }
542     if(mode_time < 0) {
543         fprintf(stderr, "'%s' in group: '%s'[%d] is negative\n", STR_MODETIME, group, index);
544         errocc |= -1;
545     }
546
547     if(errocc != 0)    //if any failure occurred
548         return -1;
549     return 0;
550 }
```

C.12.7. bc_config.h

```
1  /*************************************************************
2   * Data structures and functions for config file parsing.
3   *
4   * File:    BC_Config/bc_config.h
5   * Version: 0.1
6   * Date:    2016-03-24
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version.
10  *
11  *************************************************************/
12
13 #ifndef BC_CONFIG_H_
14 #define BC_CONFIG_H_
15
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include <unistd.h>
19 #include <stdint.h>
20 #include <errno.h>
21 #include <string.h>
22 #include <math.h>
23 #include "conlib/libconfig.h"  //for config file parser
24
25 /* defines */
26 #define MAX_SERVERS 6
27 #define SERV_CURR   0   //array index for amperementer server
28 #define SERV_VOLT   1   //array index for voltmenter server
29 #define SERV_TEMP   2   //array index for temperature measurement server
30 #define SERV_DAC    3   //array index for voltage controller server
31 #define SERV_POTI   4   //array index for potentiometer server
32 #define SERV_REL    5   //array index for relay controller server
33
34 #define MAX_LED     3
35 #define LED_R       0   //array index for red led
36 #define LED_G       1   //array index for green led
37 #define LED_B       2   //array index for blue led
38
39 #define FAC_MILLI   1E3     //conversion factor from [mA] to [A]
40 #define FAC_MICRO   1E6     //conversion factor from [µA] to [A]
41 #define FAC_NANO    1E9     //conversion factor from [nA] to [A]
42
43 /* Data types */
44 struct bc_file {            //struct for input and output files
45     const char* filepath;
46     FILE* fptr;
47     const float version;
48 } BCFile_s;
49
50 //struct for nominal cycling parameters
51 struct bc_nominal_set {
52     const double        thres_voltage;          //threshold voltage of battery cell in [V]
53     const double        nom_current;            //nominal set current in [µA]
54     const uint32_t      conv_time;              //convalescence time in [s]
```

```
55      const double        nom_RGBlight[MAX_LED];  //red, green, blue led
56      const char*         charge_mode;            //charging mode
57      const uint32_t      mode_time;              //countdown for mode end
58  } BCNominalSet_s;
59
60  //struct for limit values of battery cell
61  struct bc_limit_set {
62      const double  max_Volt;         //upper voltage limit in [V]
63      const double  min_Volt;         //lower voltage limit in [V]
64      const double  max_Temp;         //upper temperature limit in °[C]
65      const double  min_Temp;         //lower temperature limit in °[C]
66      const double  max_Ipos;         //charging current limit in [µA]
67      const double  max_Ineg;         //discharging current limit in [µA]
68  } BCLimitSet_s;
69
70  //bundled struct type of all input parameters
71  typedef struct bc_parameter_tree {
72      struct bc_file          initfile;
73      struct bc_file          paramfile;
74      const char*             sockpaths[MAX_SERVERS];
75      const char*             bat_type;
76      const double            impedance;
77      const double            hw_shunt;
78      struct bc_limit_set     limits;
79      struct bc_nominal_set   nominals;
80      const uint32_t          cycles;
81  } BCParamTree_t;
82
83  /* prototypes */
84  FILE* openConfigFile(config_t *cfg, const char* filepath);
85  void closeConfigFile(config_t *cfg, FILE* fptr);
86  int getNextParamSet(config_setting_t *cycleList, BCParamTree_t *data, int reset);
87  int checkFileVersion(config_t *cfg, float *progVersion, float *fileVersion);
88  int checkCycleBlockLimits(BCParamTree_t *data, char* group, int index, double thres_volt, double nom_curr,
89                            int conv_time, double ledRGB[MAX_LED], char* mode, int mode_time);
90
91  /* defines for reading fields from iniFile */
92  #define PROG_VERSION_INIT   0.2                     //reading program version
93
94  #define STR_VERSION         "version"               //1. field in iniFile
95
96  #define STR_INPUTGROUP      "inputgroup"            //1. group block in inifile
97  #define STR_SOCKETS         "sockets"
98  #define STR_SOCKI           "sockCurrent"
99  #define STR_SOCKV           "sockVoltage"
100 #define STR_SOCKT           "sockTemp"
101 #define STR_SOCKD           "sockDAC"
102 #define STR_SOCKP           "sockPoti"
103 #define STR_SOCKR           "sockRelay"
104
105 #define STR_BCHWGROUP       "BCHWgroup"             //2. group block in inifile
106 #define STR_SHUNT           "current_shunt"
107 int readInitFile(config_t *cfg, BCParamTree_t *data);
108
109 /* defines for reading fields from paramFile */
110 #define PROG_VERSION_PARAM  0.3                    //reading program version
111
112 #define STR_VERSION         "version"               //1. field in paramfile
113
114 #define STR_BATGROUP        "batterygroup"          //1. group block in paramfile
115 #define STR_BATTYPE         "type"
116 #define STR_BATRES          "impedance"
117
118 #define STR_LIMITGROUP      "limitgroup"            //2. group block in paramfile
119 #define STR_MAXVOLT         "max_voltage"
120 #define STR_MINVOLT         "min_voltage"
121 #define STR_MAXTEMP         "max_temp"
122 #define STR_MINTEMP         "min_temp"
123 #define STR_MAXCURRCHAR     "max_current_charge"
124 #define STR_MAXCURRDIS      "max_current_discharge"
125
126 #define STR_CYCLEGROUP      "cyclegroup"            //3. group block in paramfile
127 #define STR_STARTPHASE      "startphase"
128 #define STR_THRESVOLT       "threshold_voltage"
129 #define STR_NOMCURR         "nominal_current"
130 #define STR_CONVTIME        "convalescence"
131 #define STR_LED_R           "led_R"
132 #define STR_LED_G           "led_G"
133 #define STR_LED_B           "led_B"
134 #define STR_MODE            "mode"
135 #define STR_MODETIME        "mode_time"
136 #define STR_MODE_cCcV       "cCcV"
137 #define STR_MODE_cC         "cC"
138 #define STR_MODE_cV         "cV"
139 #define STR_CYCLEPHASE      "cyclephase"
```

```
140 #define STR_CYCLELIST       "cyclelist"
141 #define STR_CYCLES          "cycles"
142 int readParamFile(config_t *cfg, BCParamTree_t *data, config_setting_t **cycleList);
143
144 #endif /* BC_CONFIG_H_ */
```

C.13. BC_DataLogger_CP

C.13.1. BCMP_Logger.c

```
1  /*************************************************************
2   * BatCycle Measurement Protocol Logger
3   *
4   * File:    BCMP_Logger.c
5   * Version: 0.2
6   * Date:    2016-04-18
7   * Author:  Jan Griessbach
8   *
9   * Changes: 0.2: NR added equidistant mode
10  *          0.1: JG first version
11  *
12  *************************************************************/
13
14 #include "BCMP_Logger.h"
15 #include <stdlib.h>
16 #include <stdio.h>
17 #include <unistd.h>
18 #include <string.h>
19 #include <signal.h>
20 #include <sys/socket.h>
21 #include <sys/un.h>
22 #include <netinet/in.h>
23 #include <arpa/inet.h>
24 #include <sys/select.h>
25 #include <errno.h>
26 #include "BC_Protocol/BC_MeasurementProtocol.h"
27
28 static volatile sig_atomic_t BREAKFLAG = 0;
29 static volatile sig_atomic_t ALARMFLAG = 1;
30
31 int main (int argc, char **argv) {
32     int          i, maxfd = -1;
33     fd_set       readset, tempset;
34     bcmpserver_t *servers;
35     uint64_t     timestamp, prev[argc-1];
36
37     if(argc < 2) {
38         usage(argv[0]);
39         return EXIT_FAILURE;
40     }
41
42     errno = 0;
43     servers = (bcmpserver_t*) malloc((argc-1) * sizeof(bcmpserver_t));
44     if(servers == NULL) {
45         fprintf(stderr, "Out of memory: %s\n", strerror(errno));
46         return EXIT_FAILURE;
47     }
48
49     FD_ZERO(&readset);
50     memset(prev, 0x00, (argc-1) * sizeof(uint64_t));    //init prev array for timestamp calculation
51
52     /* parse commandline arguments */
53     for(i=0; i<argc-1; i++) {
54         char *fileStr, *modeStr, *intStr;
55         int mode = MODE_0;
56         int interval = -1;
57
58         fileStr = strchr(argv[i+1],':');    //return first occurrence of ':'
59         if(fileStr == NULL) {
60             usage(argv[0]);
61             return EXIT_FAILURE;
62         }
63
64         intStr = strrchr(argv[i+1],':');    //return last occurrence of ':'
65         if( intStr != fileStr ) {  //get interval
```

```
66              *intStr = '\0';
67              interval = atoi(intStr+1);
68          }
69
70          modeStr = strrchr(argv[i+1],':');   //return last occurrence of ':'
71          if( modeStr != fileStr ) { //standalone or equivalent mode
72              *modeStr = '\0';
73              mode = atoi(modeStr+1);
74          }
75
76          if(mode > 0 && interval <= 0) {
77              usage(argv[0]);
78              return EXIT_FAILURE;
79          }
80
81          *fileStr = '\0';
82          (servers+i)->sockfile = argv[i+1];
83          (servers+i)->logfile  = fileStr+1;
84          (servers+i)->mode     = mode;
85          (servers+i)->interval = interval;
86      }
87
88      /* initialization */
89      for(i=0; i<argc-1; i++) {
90          /* create sockets */
91          if( ((servers+i)->sock_fd = socket(PF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0)) < 0 ) {
92              fprintf(stderr, "Can't create socket. %s\n", strerror(errno));
93              exit(EXIT_FAILURE);
94          }
95          (servers+i)->address.sun_family = AF_LOCAL;
96          strcpy((servers+i)->address.sun_path, (servers+i)->sockfile);
97
98          /* connect sockets */
99          if(connect((servers+i)->sock_fd, (struct sockaddr*) &((servers+i)->address), sizeof ((servers+i)->address)) < 0) {
100             fprintf(stderr, "Can't connect socket. %s\n", strerror(errno));
101             exit(EXIT_FAILURE);
102         }
103
104         /* build up socket descriptor set*/
105         FD_SET((servers+i)->sock_fd, &readset);
106         maxfd = ((servers+i)->sock_fd > maxfd) ? (servers+i)->sock_fd : maxfd;
107
108         /* open logfile */
109         if( ((servers+i)->log_fp = fopen((servers+i)->logfile, "a")) == NULL ) {
110             fprintf(stderr, "Can't open logfile. %s\n", strerror(errno));
111             return EXIT_FAILURE;
112         }
113
114         /* output caption */
115         fprintf((servers+i)->log_fp, "# Timestamp\tValue\n");
116
117
118         /* get control of and send continuous command to standalone servers */
119         if( (servers+i)->mode == MODE_1 && (servers+i)->interval > 0) {
120             MPFrame_t f;
121
122             BCMP_initializeFrame(&f, BCMP_REQ_GETCONTROL);
123             BC_sendFrame((servers+i)->sock_fd, &f);
124
125             BCMP_initializeFrame(&f, BCMP_REQ_STARTCONTINUOUS);
126             BC_setArgumentAsUint32(&f, BCMP_REQ_STARTCONTINUOUS_OFFSET_INTERVAL, (servers+i)->interval);
127             BC_sendFrame((servers+i)->sock_fd, &f);
128         }
129
130     }
131
132     fprintf(stdout, "Timestamp and value logging...\n");
133
134     /* register signal handler for SIGINT signal */
135     signal(SIGINT, sigint_handler);
136
137     while(!BREAKFLAG) {
138         int sec, msec, tol = EQUIDISTANT_TOL_MS;    //50ms tolerance
139         double   value;
140
141         tempset = readset;
142         if (select(maxfd+1, &tempset, NULL, NULL, NULL) >= 0) {
143             for(i=0; i<argc-1; i++) {
144
145                 if(FD_ISSET((servers+i)->sock_fd, &tempset)) {
146                     MPFrame_t f;
147
148                     if(BC_recvFrame((servers+i)->sock_fd, &f) >= 0 && f.fields.command == BCMP_RES_VALUE) {
149                         timestamp = BC_getArgumentAsUint64(&f, BCMP_RES_VALUE_OFFSET_TIMESTAMP);
150
```

```
151                         if(((servers+i)->mode == MODE_2) && ((timestamp - prev[i]) <= ((servers+i)->interval - tol))) {
152                             #ifdef DEBUG
153                             printf("now: %"PRIu64"\tprev: %"PRIu64"\tdiff: %"PRIi64"\n", timestamp, prev[i], timestamp - prev[i]);
154                             #endif
155                             continue;
156                         }
157                         sec       = timestamp / 1000;
158                         msec      = timestamp % 1000;
159                         value     = BC_getArgumentAsDouble(&f, BCMP_RES_VALUE_OFFSET_VALUE);
160                         fprintf((servers+i)->log_fp, "%d.%03d\t%E\n", sec, msec, value);
161                         fflush((servers+i)->log_fp);
162                         prev[i] = timestamp;
163                         #ifdef DEBUG
164                         printf("%"PRIu64"\tprev: %"PRIu64"\tdiff: %"PRIi64"\n", timestamp, prev[i], timestamp - prev[i]);
165                         #else
166                         fprintf(stdout, " %d: value: %.7lf\ttimestamp: %"PRIu64"\ton %s\n"
167                                 , i+1, value, timestamp, servers[i].sockfile);
168                         #endif
169                     }
170                 }
171             }
172         }
173     }
174
175     /* clean up */
176     for(i=0; i<argc-1; i++) {
177         /* stop continuous mode and release control of standalone servers */
178         if((servers+i)->mode == MODE_1 && (servers+i)->interval > 0) {
179             MPFrame_t f;
180
181             /* stop continuous mode */
182             BCMP_initializeFrame(&f, BCMP_REQ_STOPCONTINUOUS);
183             BC_sendFrame((servers+i)->sock_fd, &f);
184             /* return control */
185             BCMP_initializeFrame(&f, BCMP_REQ_RETCONTROL);
186             BC_sendFrame((servers+i)->sock_fd, &f);
187
188             sleep(1);
189         }
190
191         /* close socket connection */
192         shutdown((servers+i)->sock_fd, SHUT_RDWR);
193         close((servers+i)->sock_fd);
194         fclose((servers+i)->log_fp);
195     }
196
197     /* deallocate memory */
198     free(servers);
199     return EXIT_SUCCESS;
200 }
201
202 void sigint_handler(int signal)
203 {
204     BREAKFLAG = 1;
205 }
206
207 void sigalarm_handler(int signal)
208 {
209     ALARMFLAG = 1;
210 }
211
212 void usage(char *progname)
213 {
214     fprintf(stderr, "\n\tUSAGE:\t %s sockfile:logfile[:mode:interval]\n"
215                     "\t\t\t\t   [sockfile:logfile[:mode:interval]]\n"
216                     "\t\t\t\t   [...]\n"
217                     "\n\tOPTIONS: sockfile: existing path to any BCMP-socket.\n"
218                     "\t\t logfile:  name of logfile that will be created. If file\n"
219                     "\t\t\t   exists, it will be appended.\n\n"
220                     "\t\t mode:\t0: default mode; DataLogger logs every value\n"
221                     "\t\t\t   and timestamp on socket.\n"
222                     "\t\t\t1: standalone mode; DataLogger becomes master client \n"
223                     "\t\t\t   on BCMP-server. Server will send values in\n"
224                     "\t\t\t   continuous mode.\n"
225                     "\t\t\t2: equidistant mode; DataLogger reads values from\n"
226                     "\t\t\t   socket in quasi equidistant time, which is delivered\n"
227                     "\t\t\t   in interval.\n"
228                     "\t\t interval: time in msec in combination with an mode argument\n"
229                     "\t\t\t   greater than 0.\n"
230                     "\n\tDESCRIPTION:\t   BatCycle Measurement Protocol Logger.\n\n", progname);
231 }
```

C.13.2. BCMP_Logger.h

```
/*************************************************************
 * BatCycle Measurement Protocol Logger
 *
 * File:    BCMP_Logger.h
 * Version: 0.2
 * Date:    2016-04-18
 * Author:  Jan Griessbach
 *
 * Changes: 0.2: NR added equidistant mode
 *          0.1: JG first version
 *
 *************************************************************/

#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <inttypes.h>

/* defines */
#define MODE_0      0       //default mode (read every value on socket)
#define MODE_1      1       //server interval mode
#define MODE_2      2       //client interval mode

#define EQUIDISTANT_TOL_MS  50      //tolerance of equidistance in MODE_2

/* structs */
typedef struct struct_bcmpserver {
    char *sockfile;
    char *logfile;
    int  sock_fd;
    FILE *log_fp;
    int  mode;
    int  interval;
    struct sockaddr_un address;
} bcmpserver_t;

/* prototypes */
void sigint_handler(int signal);
void sigalarm_handler(int signal);
void usage(char *progname);
```

C.14. start_all.sh

```
# This skript starts the batcycle programs and kill camera process when cycling is finished

# User definable variables
CELL="MR23Z2P05"
STARTDATE=$(date  +"%m-%d-%Y") #string variables to change by user
USBDRIVE="/media/65b55418-38c6-4990-b8ba-a1cf2dee2462/Nico/BatCycle"
FOLDER=$USBDRIVE/$CELL'_'$STARTDATE

# Fixed variables

mkdir $FOLDER       #create new folder on HDD
mkdir $FOLDER/Images

sockFileI="/tmp/Isock.uds"      #socket file of Amperemeter
sockFileV="/tmp/Vsock.uds"      #socket file of Voltmeter
sockFileT="/tmp/Tsock.uds"      #socket file of Temperature
sockFileD="/tmp/Dsock.uds"      #socket file of VoltageController
sockFileR="/tmp/Rsock.uds"      #socket file of RelayController

initFile="in/bc_init.cfg"       #input files
paramFile="in/bc_param.cfg"

logFileI=$FOLDER'/'$CELL'_'$STARTDATE'_logI.txt'      #output files on HDD
logFileV=$FOLDER'/'$CELL'_'$STARTDATE'_logV.txt'
logFileT=$FOLDER'/'$CELL'_'$STARTDATE'_logT.txt'
#logFileI=out/logI.txt
#logFileV=out/logV.txt
#logFileT=out/logT.txt
PICS=$FOLDER/Images/'%s'.png

#Amperemeter
lxterminal --command="./BC_DMM4020Reader_SP -d /dev/ttyUSB2 -p $sockFileI -m 2" --title="DMM4020 Amperemeter" --geometry=74x20
```

```
33 #lxterminal --command="sudo ./BC_Amperemeter_SP -p $sockFileI -n 3" --title="BC_Amperemeter_SP" --geometry=74x20
34
35 #Voltmeter
36 lxterminal --command="./BC_DMM4020Reader_SP -d /dev/ttyUSB1 -p $sockFileV -m 0" --title="DMM4020 Voltmeter" --geometry=74x20
37 #lxterminal --command="sudo ./BC_Voltmeter_SP -p $sockFileV -n 3" --title="BC_Voltmeter_SP" --geometry=74x20
38
39 #Temperature
40 #lxterminal --command="sudo ./BC_Temperature_SP -p $sockFileT -n 3" --title="BC_Temperature_SP" --geometry=74x20
41
42 #controllers
43 lxterminal --command="sudo ./BC_VoltageController_SP -p $sockFileD -n 3" --title="BC_VoltageController_SP" --geometry=74x20
44 lxterminal --command="sudo ./BC_RelayController_SP -p $sockFileR -n 3" --title="BC_RelayController_SP" --geometry=74x20
45
46 sleep 8             #wait for servers
47
48 #DataLogger
49 #lxterminal --command="./BC_DataLogger_CP $sockFileI:$logFileI:2:1000 $sockFileV:$logFileV:2:1000 $sockFileT:$logFileT:0:0" --title="
       BC_DataLogger_CP" --geometry=74x20
50 lxterminal --command="./BC_DataLogger_CP $sockFileI:$logFileI:2:1000 $sockFileV:$logFileV:2:1000" --title="BC_DataLogger_CP" --geometry
       =74x20
51
52 #MainController
53 lxterminal --command="./BC_MainController_CP -i $initFile -p $paramFile" --title="BC_MainController_CP" --geometry=74x20 &
54 PID_MAIN=$!
55
56 #start camera tool fswebcam in background
57 #lxterminal --comand='fswebcam -d /dev/video0 --loop 10 -F5  --png 0 --scale 640x480 --set "White Balance Temperature, Auto=False" --
       set "Backlight Compensation=0" --save $PICS' --title="fswebcam" --geometry=74x20
58 fswebcam -d /dev/video0 --loop 10 -F5  --png 0 --scale 640x480 --set "White Balance Temperature, Auto=False" --set "Backlight
       Compensation"=0 --save $PICS &
59 PID_CAM=$!
60
61 sleep 1
62
63 echo "kill fswebcam"
64 #loop checks if MainController has finished, for kill fswebcam
65 while [ true ]; do
66     var='ps a | grep BC_MainController_CP | wc -l'      #filter prozess view for BC_MainController_CP and count results
67
68     if test $var -le 1; then        #if results less than 1, MainController has finished
69         echo "MainController is finished"
70         sudo kill -SIGINT $PID_CAM          #kill fswebcam process
71         break
72     fi
73
74     sleep 10
75 done
```

D. Test Programme

D.1. MP_FrameDebugger

D.1.1. mp_debug.c

```
1  /************************************************************
2   * Debugging tool for measurement server process
3   *
4   * File:    TestTools/MP_FrameDebugger/mp_debug.c
5   * Version: 0.1
6   * Date:    2016-02-12
7   * Author:  Jan Griessbach, Nico Rieckmann
8   *
9   * Changes: 0.1: JG, NR first version
10  *
11  ************************************************************/
12 /* uds_client.c */
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <unistd.h>
16 //for toupper fkt
17 #include <ctype.h>
18 //for sockets
19 #include <sys/socket.h>
20 #include <sys/un.h>
21 #include <netinet/in.h>
22 #include <arpa/inet.h>
23 //for printf
24 #include <inttypes.h>
25 //for signal handler
26 #include <signal.h>
27 //for boolean
28 #include <stdbool.h>
29 //for client server communication
30 #include "BC_Protocol/BC_MeasurementProtocol.h"
31
32 #define OPT_STRING  "p:h"
33 //#define UDS_FILE    ""      //kann weg
34
35 void clear(FILE *file);
36 void printProtocol(void);
37 void usage(char *progname);
38
39 volatile sig_atomic_t runFlag = true;   //atomic Flag for Interrupthandler
40
41 void sigEnd(int sign) {
42     runFlag = false;
43 }
44
45 int main (int argc, char **argv) {
46     int sock_fd;                    //filedescriptor of corresponding socket
47     struct sockaddr_un address;     //address struct
48     BCFrame_t inframe, outframe;
49     int command, len;
50     int retval;
51     char c;
52     int i;
53     sig_atomic_t intervalMode = false;
54     char *sockFilePath;
55     int mandatoryOptions = 1;
56
57     printf("#BC_FrameDebug_clientProc start\n\n");
58
59
```

```
60
61      //evaluate commandline options
62      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
63          switch(retval) {
64          case 'p':       //path of the socketfile
65              sockFilePath = optarg;
66              mandatoryOptions--;
67              #ifdef DEBUG
68              printf("p argument: %s", optarg);
69              #endif
70              break;
71          case 'h':
72          default:
73              usage(argv[0]);
74              exit(EXIT_FAILURE);
75          }
76      }
77
78      //limit check
79      if(mandatoryOptions > 0) {
80          usage(argv[0]);
81          exit(EXIT_FAILURE);
82      }
83
84      //create socket discriptor
85      sock_fd = socket(PF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0); //use unix socket in stream mode, nonblock
86      if(sock_fd <= 0) {          //if an error occurred
87          perror("Client: Failure at socket() call");
88          close(sock_fd);
89          exit(EXIT_FAILURE);
90      }
91
92      //init structs
93      address.sun_family = AF_LOCAL;          //make them to unix sockets
94      strcpy(address.sun_path, sockFilePath);
95
96      //connect to socket file
97      #ifdef DEBUG
98      printf(".try to connect with socket...\n");
99      #endif
100     if(connect(sock_fd, (struct sockaddr*) &address, sizeof (address)) == -1) {
101         perror("Client: Failure at connect to socket");
102         close(sock_fd);
103         exit(EXIT_FAILURE);
104     }
105     #ifdef DEBUG
106     printf(".Client is connected\n");
107     #endif // DEBUG
108
109     printProtocol();
110
111     //signal(SIGINT, sigEnd);         //register signalhandler for SIGINT
112     //memset(outframe.buffer, 0x00, BC_MAX_FRAME_SIZE);     //initialize outframe.buffer with zeros
113     //*(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;         //init output frame
114     while(runFlag) {    //false when SIGINT
115         printf("[n]ew frame,   [s]end frame,   [r]ead frame,   [v]iew protocol,   [q]uit\n");
116
117         c = fgetc(stdin);
118         clear(stdin);       //clear stdin buffer
119         c = (char) tolower(c);
120         switch(c) {
121         case 'n':       //create new frame
122             printf(".create new frame\n");
123             //set command
124             printf("command: 0x");
125             scanf("%x", &command);
126             clear(stdin);
127
128             //set argument length
129             printf("Arglen: 0x");
130             scanf("%x", &len);
131             clear(stdin);
132
133             //set endchar of frame
134             BC_initializeFrame(&outframe, (uint8_t) command, (uint8_t) len);
135
136             //set arguments
137             if(outframe.fields.argLen > 0) {
138                 unsigned int temp;
139
140                 printf("Argument[%d]: \n", outframe.fields.argLen);
141                 for(i=0; i<outframe.fields.argLen; i++) {
142                     printf("[%d]: 0x", i);
143                     scanf("%2x", &temp);
144                     clear(stdin);
```

```
145                     outframe.fields.argBuffer[i] = (uint8_t) temp;
146                 }
147             }
148             //printf("%x und %d\n", *outframe.fields.argBuffer, outframe.fields.argBuffer);
149             BC_dumpFrameData(&outframe);
150             printf("\n");
151             *(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;
152             break;
153         case 's':       //send frame
154             printf(".send frame to server\n");
155             retval = BC_sendFrame(sock_fd, &outframe);
156             if(retval < 0) {
157                 perror("Error occurred at BC_sendFrame()");
158                 runFlag = false;
159             }
160             BC_dumpFrameData(&outframe);
161             printf("\n");
162             //for Continiuous mode set a flag
163             if(outframe.fields.command == BCMP_REQ_STARTCONTINUOUS)
164                 intervalMode = true;
165             if(outframe.fields.command == BCMP_REQ_STOPCONTINUOUS)
166                 intervalMode = false;
167             break;
168         case 'r':       //receive frame
169             printf(".receive frame from server\n");
170             do {
171                 retval = BC_recvFrame(sock_fd, &inframe);
172                 if(retval < 0) {
173                     if(retval == -1) ;
174                         //perror("Error occurred at BC_recvFrame()");
175                     else
176                         fprintf(stderr, "Error %d occurred at BC_recvFrame()\n", retval);
177                 }
178                 else if (retval == 0) {
179                     #ifdef DEBUG
180                     printf(".no more messages in socket\n");
181                     #endif
182                 }
183                 else if(retval > 0) {
184                     BC_dumpFrameData(&inframe);
185                     printf(".Timestamp as Int: %"PRIu64"\n", BC_getArgumentAsUint64(&inframe, BCMP_RES_VALUE_OFFSET_TIMESTAMP));
186                     printf(".Measure value as Int: %"PRIu64"\n", BC_getArgumentAsUint64(&inframe, BCMP_RES_VALUE_OFFSET_VALUE));
187                     #ifdef DEBUG
188                     printf(".Measure value as Double: %lf\n", BC_getArgumentAsDouble(&inframe, BCMP_RES_VALUE_OFFSET_VALUE));
189                     #endif
190                     printf("\n");
191                 }
192             } while(retval > 0 || intervalMode == true);
193             break;
194         case 'v':       //view protcoll
195             printProtocol();
196             break;
197         case 'q':       //quit program
198             runFlag = false;
199             break;
200         default:        //else
201             printf(".wrong input\n");
202         }
203     }
204     close(sock_fd);
205     printf("#BC_FrameDebug_clientProc end\n");
206     exit(EXIT_SUCCESS);
207 }
208
209 void clear(FILE *file) {    //flushs a file buffer
210     char c;
211
212     while(((c = fgetc(file)) != '\n') && (c != EOF));
213 }
214
215 void printProtocol(void) {
216     printf("This Tool is a frame debugger for the measurement protocol.\n"
217            "Requests:\n"
218            "         |  Commands (uint8)  |  Arglen (uint8)  |    Arguments (uint8[20])    |\n"
219            "GetValue |               0x01 |                0 |                           - |\n"
220            "StartCon |               0x02 |                4 |     Interval [ms] (uint32) |\n"
221            "StopCon  |               0x03 |                0 |                           - |\n"
222            "GetCtl   |               0x04 |                0 |                           - |\n"
223            "RetCtl   |               0x05 |                0 |                           - |\n"
224            "Quit     |               0x7F |                0 |                           - |\n"
225            "\n"
226            "Response:\n"
227            "         |  Commands (uint8)  |  Arglen (uint8)  |    Arguments (uint8[20])    |\n"
228            "Val      |               0x81 |               12 | time (uint32), val (double) |\n"
229            "Ctl      |               0x84 |                1 | 0: denial, 1: grant, 2: Ret |\n"
```

```
230             "\n");
231 }
232
233
234
235 void usage(char *progname) {
236     fprintf(stderr, "\n\tUSAGE:\t %s -p sock [-h help]\n"
237                     "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
238                     "\t\t\tNo default\n"
239                     "\t\t -h\tHelp; view help text\n"
240                     "\n\tDESCRIPTION:\tDebug process for the measurement protocol\n\n"
241                     , progname);
242 }
```

D.2. AP_FrameDebugger

D.2.1. ap_debug.c

```
1  /*************************************************************
2   * Debugging tool for BC_VoltageController_SP and BC_Potentiometer_SP
3   *
4   * File:    TestTools/AP_FrameDebugger/ap_debug.c
5   * Version: 0.1
6   * Date:    2016-03-10
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version
10  *
11  *************************************************************/
12 /* uds_client.c */
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <unistd.h>
16 //for toupper fkt
17 #include <ctype.h>
18 //for sockets
19 #include <sys/socket.h>
20 #include <sys/un.h>
21 #include <netinet/in.h>
22 #include <arpa/inet.h>
23 //for printf
24 #include <inttypes.h>
25 //for signal handler
26 #include <signal.h>
27 //for boolean
28 #include <stdbool.h>
29 //for client server communication
30 #include "BC_Protocol/BC_ActuatorProtocol.h"
31
32 //#define DEBUG       1     //kann weg
33 #define OPT_STRING  "p:h"
34 #define UDS_FILE    "/tmp/Dsock.uds"      //default
35
36 void clear(FILE *file);
37 void printProtocol(void);
38 void usage(char *progname);
39
40 volatile sig_atomic_t runFlag = true;   //atomic Flag for Interrupthandler
41
42 void sigEnd(int sign) {
43     runFlag = false;
44 }
45
46 int main (int argc, char **argv) {
47     int sock_fd;                    //filedescriptor of corresponding socket
48     struct sockaddr_un address;     //address struct
49     APFrame_t outframe;
50     int command, len;
51     int retval;
52     char c;
53     //int i;
54     char *sockFilePath;
55     //int mandatoryOptions = 1;
56
57     printf("#CP_ActuatorDebugger start\n\n");
```

```
58
59      //evaluate commandline options
60      sockFilePath = UDS_FILE;        //set default path
61      while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
62          switch(retval) {
63          case 'p':       //path of the socketfile
64              sockFilePath = optarg;
65              //mandatoryOptions--;
66              #ifdef DEBUG
67              printf("p argument: %s", optarg);
68              #endif
69              break;
70          case 'h':
71          default:
72              usage(argv[0]);
73              exit(EXIT_FAILURE);
74          }
75      }
76
77      /*//limit check
78      if(mandatoryOptions > 0) {
79          usage(argv[0]);
80          exit(EXIT_FAILURE);
81      }*/
82
83      //create socket discriptor
84      sock_fd = socket(PF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0); //use unix socket in stream mode, nonblock
85      if(sock_fd <= 0) {          //if an error occurred
86          perror("Client: Failure at socket() call");
87          close(sock_fd);
88          exit(EXIT_FAILURE);
89      }
90
91      //init structs
92      address.sun_family = AF_LOCAL;          //make them to unix sockets
93      strcpy(address.sun_path, sockFilePath);
94
95      //connect to socket file
96      #ifdef DEBUG
97      printf(".try to connect with socket...\n");
98      #endif
99      if(connect(sock_fd, (struct sockaddr*) &address, sizeof (address)) == -1) {
100         perror("Client: Failure at connect to socket");
101         close(sock_fd);
102         exit(EXIT_FAILURE);
103     }
104     #ifdef DEBUG
105     printf(".Client is connected\n");
106     #endif // DEBUG
107
108     printProtocol();
109
110     //signal(SIGINT, sigEnd);         //register signalhandler for SIGINT
111     //memset(outframe.buffer, 0x00, BC_MAX_FRAME_SIZE);     //initialize outframe.buffer with zeros
112     //*(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;         //init output frame
113     while(runFlag) {    //false when SIGINT
114         printf("[n]ew frame,   [s]end frame,   [v]iew protocol,   [q]uit\n");
115
116         c = fgetc(stdin);
117         clear(stdin);       //clear stdin buffer
118         c = (char) tolower(c);
119         switch(c) {
120         case 'n':       //create new frame
121             printf(".create new frame\n");
122             //set command
123             printf("command: 0x");
124             scanf("%x", &command);
125             clear(stdin);
126
127             //set argument length
128             printf("Arglen: 0x");
129             scanf("%x", &len);
130             clear(stdin);
131
132             //set endchar of frame
133             //BC_initializeFrame(&outframe, (uint8_t) command, (uint8_t) len);      //kann weg
134             BCAP_initializeFrame(&outframe, (uint8_t) command);
135
136             //set arguments
137             if(outframe.fields.argLen > 0) {
138
139                 if(command == BCAP_REQ_SETDAC) {
140                     double temp;
141
142                     printf("Argument[%d]: \n", outframe.fields.argLen);
```

```
143                     scanf("%lf", &temp);
144                     clear(stdin);
145                     *((double*) outframe.fields.argBuffer) = temp;
146                 }
147                 else if(command == BCAP_REQ_SETLED) {
148                     float temp[3];
149
150                     printf("Argument 1 [%d]: \n", sizeof(float));
151                     scanf("%f", &temp[0]);
152                     printf("Argument 2 [%d]: \n", sizeof(float));
153                     scanf("%f", &temp[1]);
154                     printf("Argument 3 [%d]: \n", sizeof(float));
155                     scanf("%f", &temp[2]);
156                     clear(stdin);
157                     *((float*) (outframe.fields.argBuffer + BCAP_REQ_LED_RED_OFFSET_VALUE)) = temp[0];
158                     *((float*) (outframe.fields.argBuffer + BCAP_REQ_LED_GREEN_OFFSET_VALUE)) = temp[1];
159                     *((float*) (outframe.fields.argBuffer + BCAP_REQ_LED_BLUE_OFFSET_VALUE)) = temp[2];
160                 }
161                 else {
162                     unsigned int temp, i;
163
164                     printf("Argument[%d]: \n", outframe.fields.argLen);
165                     for(i=0; i<outframe.fields.argLen; i++) {
166                         printf("[%d]: 0x", i);
167                         scanf("%2x", &temp);
168                         clear(stdin);
169                         outframe.fields.argBuffer[i] = (uint8_t) temp;
170                     }
171                 }
172             }
173             //printf("%x und %d\n", *outframe.fields.argBuffer, outframe.fields.argBuffer);
174             BC_dumpFrameData(&outframe);
175             printf("\n");
176             *(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;
177             break;
178         case 's':       //send frame
179             printf(".send frame to server\n");
180             retval = BC_sendFrame(sock_fd, &outframe);
181             if(retval < 0) {
182                 perror("Error occurred at BC_sendFrame()");
183                 runFlag = false;
184             }
185             BC_dumpFrameData(&outframe);
186             printf("\n");
187             break;
188         case 'v':       //view protcoll
189             printProtocol();
190             break;
191         case 'q':       //quit program
192             runFlag = false;
193             break;
194         default:        //else
195             printf(".wrong input\n");
196         }
197     }
198     close(sock_fd);
199     printf("#CP_ActuatorDebugger end\n");
200     exit(EXIT_SUCCESS);
201 }
202
203 void clear(FILE *file) {    //flushs a file buffer
204     char c;
205
206     while(((c = fgetc(file)) != '\n') && (c != EOF));
207 }
208
209 void printProtocol(void) {
210     printf("This Tool is a frame debugger for the actuator protocol.\n"
211            "Requests:\n"
212            "         |  Commands (uint8)  |  Arglen (uint8)  |  Arguments (uint8[20])  |\n"
213            "SetDAC   |              0x01 |                8 |          Value (double) |\n"
214            "SetLED   |              0x02 |               12 |        Value[3] (float) |\n"
215            "Quit     |              0x7F |                0 |                       - |\n"
216            "\n");
217 }
218
219 void usage(char *progname) {
220     fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-h help]\n"
221                     "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
222                     "\t\t\tDefault is %s\n"
223                     "\t\t -h\tHelp; view help text\n"
224                     "\n\tDESCRIPTION:\tDebug process for the actuator protocol\n\n"
225                     , progname, UDS_FILE);
```

```
226 }
```

D.3. RP_FrameDebugger

D.3.1. rp_debug.c

```
1  /************************************************************
2   * Debugging tool for relay server process
3   *
4   * File:    TestTools/RP_FrameDebugger/rp_debug.c
5   * Version: 0.1
6   * Date:    2016-02-28
7   * Author:  Nico Rieckmann
8   *
9   * Changes: 0.1: NR first version
10  *
11  ************************************************************/
12 /* uds_client.c */
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <unistd.h>
16 //for toupper fkt
17 #include <ctype.h>
18 //for sockets
19 #include <sys/socket.h>
20 #include <sys/un.h>
21 #include <netinet/in.h>
22 #include <arpa/inet.h>
23 //for printf
24 #include <inttypes.h>
25 //for signal handler
26 #include <signal.h>
27 //for boolean
28 #include <stdbool.h>
29 //for client server communication
30 #include "BC_Protocol/BC_RelayProtocol.h"
31
32 //#define DEBUG       1     //kann weg
33 #define OPT_STRING  "p:h"
34 #define UDS_FILE    "/tmp/Rsock.uds"      //default
35
36 void clear(FILE *file);
37 void printProtocol(void);
38 void usage(char *progname);
39
40 volatile sig_atomic_t runFlag = true;   //atomic Flag for Interrupthandler
41
42 void sigEnd(int sign) {
43     runFlag = false;
44 }
45
46 int main (int argc, char **argv) {
47     int sock_fd;                    //filedescriptor of corresponding socket
48     struct sockaddr_un address;     //address struct
49     BCFrame_t inframe, outframe;
50     int command, len;
51     int retval;
52     char c;
53     int i;
54     sig_atomic_t intervalMode = false;
55     char *sockFilePath;
56     //int mandatoryOptions = 1;
57
58     printf("#CP_RelayControl_Debugger start\n\n");
59
60     //evaluate commandline options
61     sockFilePath = UDS_FILE;        //set default path
62     while((retval = getopt(argc, argv, OPT_STRING)) != -1 ) {
63         switch(retval) {
64         case 'p':       //path of the socketfile
65             sockFilePath = optarg;
66             //mandatoryOptions--;
67             #ifdef DEBUG
68             printf("p argument: %s", optarg);
69             #endif
```

```
70              break;
71          case 'h':
72          default:
73              usage(argv[0]);
74              exit(EXIT_FAILURE);
75          }
76      }
77
78      /*//limit check
79      if(mandatoryOptions > 0) {
80          usage(argv[0]);
81          exit(EXIT_FAILURE);
82      }*/
83
84      //create socket discriptor
85      sock_fd = socket(PF_LOCAL, SOCK_STREAM | SOCK_NONBLOCK, 0); //use unix socket in stream mode, nonblock
86      if(sock_fd <= 0) {          //if an error occurred
87          perror("Client: Failure at socket() call");
88          close(sock_fd);
89          exit(EXIT_FAILURE);
90      }
91
92      //init structs
93      address.sun_family = AF_LOCAL;          //make them to unix sockets
94      strcpy(address.sun_path, sockFilePath);
95
96      //connect to socket file
97      #ifdef DEBUG
98      printf(".try to connect with socket...\n");
99      #endif
100     if(connect(sock_fd, (struct sockaddr*) &address, sizeof (address)) == -1) {
101         perror("Client: Failure at connect to socket");
102         close(sock_fd);
103         exit(EXIT_FAILURE);
104     }
105     #ifdef DEBUG
106     printf(".Client is connected\n");
107     #endif // DEBUG
108
109     printProtocol();
110
111     //signal(SIGINT, sigEnd);          //register signalhandler for SIGINT
112     //memset(outframe.buffer, 0x00, BC_MAX_FRAME_SIZE);     //initialize outframe.buffer with zeros
113     //*(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;          //init output frame
114     while(runFlag) {    //false when SIGINT
115         printf("[n]ew frame,   [s]end frame,   [r]ead frame,   [v]iew protocol,   [q]uit\n");
116
117         c = fgetc(stdin);
118         clear(stdin);       //clear stdin buffer
119         c = (char) tolower(c);
120         switch(c) {
121         case 'n':       //create new frame
122             printf(".create new frame\n");
123             //set command
124             printf("command: 0x");
125             scanf("%x", &command);
126             clear(stdin);
127
128             //set argument length
129             printf("Arglen: 0x");
130             scanf("%x", &len);
131             clear(stdin);
132
133             //set endchar of frame
134             BC_initializeFrame(&outframe, (uint8_t) command, (uint8_t) len);
135
136             //set arguments
137             if(outframe.fields.argLen > 0) {
138                 unsigned int temp;
139
140                 printf("Argument[%d]: \n", outframe.fields.argLen);
141                 for(i=0; i<outframe.fields.argLen; i++) {
142                     printf("[%d]: 0x", i);
143                     scanf("%2x", &temp);
144                     clear(stdin);
145                     outframe.fields.argBuffer[i] = (uint8_t) temp;
146                 }
147             }
148             //printf("%x und %d\n", *outframe.fields.argBuffer, outframe.fields.argBuffer);
149             BC_dumpFrameData(&outframe);
150             printf("\n");
151             *(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;
152             break;
153         case 's':       //send frame
154             printf(".send frame to server\n");
```

```
155             retval = BC_sendFrame(sock_fd, &outframe);
156             if(retval < 0) {
157                 perror("Error occurred at BC_sendFrame()");
158                 runFlag = false;
159             }
160             BC_dumpFrameData(&outframe);
161             printf("\n");
162         case 'r':       //receive frame
163             printf(".receive frame from server\n");
164             do {
165                 retval = BC_recvFrame(sock_fd, &inframe);
166                 if(retval < 0) {
167                     if(retval == -1) ;
168                         //perror("Error occurred at BC_recvFrame()");
169                     else
170                         fprintf(stderr, "Error %d occurred at BC_recvFrame()\n", retval);
171                 }
172                 else if (retval == 0) {
173                     #ifdef DEBUG
174                     printf(".no more messages in socket\n");
175                     #endif
176                 }
177                 else if(retval > 0) {
178                     BC_dumpFrameData(&inframe);
179                     printf("\n");
180                 }
181             } while(retval > 0 || intervalMode == true);
182             break;
183         case 'v':       //view protcoll
184             printProtocol();
185             break;
186         case 'q':       //quit program
187             runFlag = false;
188             break;
189         default:        //else
190             printf(".wrong input\n");
191         }
192     }
193     close(sock_fd);
194     printf("#CP_RelayControl_Debugger end\n");
195     exit(EXIT_SUCCESS);
196 }
197
198 void clear(FILE *file) {    //flushs a file buffer
199     char c;
200
201     while(((c = fgetc(file)) != '\n') && (c != EOF));
202 }
203
204 void printProtocol(void) {
205     printf("This Tool is a frame debugger for the relay protocol.\n"          "Requests:\n"
206            "         |  Commands (uint8)  |  Arglen (uint8)  |  Arguments (uint8[20])  |\n"
207            "SetReg   |              0x01 |                1 |         Pattern (uint8) |\n"
208            "CloseRel |              0x02 |                1 |         Relay_X (uint8) |\n"
209            "OpenRel  |              0x03 |                1 |         Relay_X (uint8) |\n"
210            "TogRel   |              0x04 |                1 |         Relay_X (uint8) |\n"
211            "Quit     |              0x7F |                0 |                       - |\n"
212            "\n"
213            "Arguments:\n"
214            "           Relay_VN  =  0x1 \n"
215            "           Relay_VP  =  0x2 \n"
216            "           Relay_I   =  0x4 \n"
217            "\n"
218            "           Pattern   =  Relay_VN | Relay_VP | Relay_I \n"
219            "\n");
220 }
221
222 void usage(char *progname) {
223     fprintf(stderr, "\n\tUSAGE:\t %s [-p sock] [-h help]\n"
224                     "\n\tOPTIONS: -p\tSocketfile; will be overwritten if it exists.\n"
225                     "\t\t\tDefault is %s\n"
226                     "\t\t -h\tHelp; view help text\n"
227                     "\n\tDESCRIPTION:\tDebug process for the relay protocol\n\n"
228                     , progname, UDS_FILE);
229 }
```

D.4. LinearityTest

D.4.1. lin_test.c

```
1  /*************************************************************
2   * Tool to drive up linearly the DAC, ADC_V and ADC_I and log the
3   * characteristic line
4   *
5   * File:    TestTools/LinearityTest/lin_test.c
6   * Version: 0.1
7   * Date:    2016-03-12
8   * Author:  Nico Rieckmann
9   *
10  * Changes: 0.1: NR first version
11  *
12  *************************************************************/
13 /* uds_client.c */
14 #include <stdlib.h>
15 #include <stdio.h>
16 #include <unistd.h>
17 //for toupper fkt
18 #include <ctype.h>
19 //for sockets
20 #include <sys/socket.h>
21 #include <sys/un.h>
22 #include <netinet/in.h>
23 #include <arpa/inet.h>
24 //for printf
25 #include <inttypes.h>
26 //for signal handler
27 #include <signal.h>
28 //for boolean
29 #include <stdbool.h>
30 //for client server communication
31 #include "BC_Protocol/BC_MeasurementProtocol.h"
32 #include "BC_HW_Lib/BC_HW_DAC.h"
33 #include "BC_HW_Lib/BC_HW_ADC.h"
34
35 /* defines */
36 #define UDS_FILE_DAC_VOLTAGE    "/tmp/DMM4020VDACsock.uds" //default socket for external measurement
37 #define UDS_FILE_VOLTAGE        "/tmp/DMM4020Vsock.uds" //default socket for external measurement
38 #define UDS_FILE_CURRENT        "/tmp/DMM4020Isock.uds" //default socket for external measurement
39 #define UDS_FILE_DAC            "/tmp/Dsock.uds"        //default socket for Device under Test
40 #define UDS_FILE_ADC_I          "/tmp/Isock.uds"        //default socket for Device under Test
41 #define UDS_FILE_ADC_V          "/tmp/Vsock.uds"        //default socket for Device under Test
42 #define FILE_PATH_DAC           "linTest_DAC.csv"
43 #define FILE_PATH_ADC_I         "linTest_ADC_I"
44 #define FILE_PATH_ADC_V         "linTest_ADC_V"
45 #define REDO                    10
46
47 struct params {
48     double startVal;
49     double endVal;
50     double stepSize;
51 };
52
53 void clear(FILE *file);
54 void printInfo();       //TODO anpassen
55 void usage(char *progname); //TODO anpassen
56 int connectToSock(int *sock_fd, const char *sockFilePath);
57 int requestToMasterClient(int sock_fd);
58 int requestValue(int sock_fd, uint64_t *timestamp, double *value);
59 void getParams(struct params *dac);
60
61 volatile sig_atomic_t runFlag = true;   //atomic Flag for Interrupthandler
62
63 void sigEnd(int sign) {
64     runFlag = false;
65 }
66
67 int main (int argc, char **argv) {
68     char *sock1_path, *sock2_path;      //socket paths for external measurements
69     int sock1_fd, sock2_fd;             //filedescriptor of corresponding socket
70     int retval;
71     char c;
72     int i;
73     int mandatoryOptions = 1;
74     FILE *logfile = NULL;
75
76     printf("#LinearityTest start\n\n");
77
```

```
78      sock1_path = UDS_FILE_DAC_VOLTAGE;
79      sock1_path = NULL;
80      //evaluate commandline options
81      while((retval = getopt(argc, argv, "1:2:h")) != -1 ) {
82          switch(retval) {
83          case '1':       //path of the socketfile
84              sock1_path = optarg;
85              mandatoryOptions--;
86              break;
87          case '2':
88              sock2_path = optarg;
89              break;
90          case 'h':
91          default:
92              usage(argv[0]);
93              exit(EXIT_FAILURE);
94          }
95      }
96      //limit check
97      if(mandatoryOptions > 0) {
98          usage(argv[0]);
99          exit(EXIT_FAILURE);
100     }
101
102     //initialize hardware
103     retval = BCHW_initializeDACgpio(CS_PIN_DAC);
104     retval |= BCHW_initializeADCgpio(CS_PIN_ADC_I, ADC_CONV_PIN);
105     retval |= BCHW_initializeADCgpio(CS_PIN_ADC_V, ADC_CONV_PIN);
106     if(retval < 0) {
107         perror("Failure occurred in initialize hardware");
108         exit(EXIT_FAILURE);
109     }
110     //get initial values from adc's and throw away
111     BCHW_readFromADC(CS_PIN_ADC_I, ADC_CONV_PIN);
112     BCHW_readFromADC(CS_PIN_ADC_V, ADC_CONV_PIN);
113
114     printInfo();
115
116     if(connectToSock(&sock1_fd, sock1_path) < 0)
117         exit(EXIT_FAILURE);
118     usleep(500000);   //wait 5ms for servers registration
119     if(requestToMasterClient(sock1_fd) < 0)
120         exit(EXIT_FAILURE);
121     //signal(SIGINT, sigEnd);         //register signalhandler for SIGINT
122     //memset(outframe.buffer, 0x00, BC_MAX_FRAME_SIZE);     //initialize outframe.buffer with zeros
123     //*(outframe.fields.argBuffer + outframe.fields.argLen) = BC_FRAME_END_CHAR;         //init output frame
124     while(runFlag) {    //false when SIGINT
125         struct params dac;
126         double sweepV = 0, dac_voltage = 0;
127         int sweepInt = 0;
128         uint64_t timestamp = 0;
129
130         BCHW_sendToDAC(CS_PIN_DAC, 0);      //initialize DAC output to 0V
131         printf("[D]AC ,   ADC_[I],   ADC_[V],   [q]uit\n");
132
133         c = fgetc(stdin);
134         clear(stdin);       //clear stdin buffer
135         c = (char) tolower(c);
136         switch(c) {
137         case 'd':       //test the DAC unit
138             printf("step up DAC\n");
139
140             logfile = fopen(FILE_PATH_DAC, "a+");
141             if(logfile < 0) {
142                 perror("can not open logfile");
143                 break;
144             }
145             getParams(&dac);
146             fprintf(logfile, "Linearity test for DAC\n"
147                              "DAC input nominal [V]; DAC input decimal; DAC output measured [V]; timestamp\n");
148
149             //for loop to sweep the DAC steps
150             for(sweepV=dac.startVal; sweepV<=dac.endVal; sweepV+=dac.stepSize) {
151                 sweepInt = BCHW_parseToDACvalue(sweepV);
152                 BCHW_sendToDAC(CS_PIN_DAC, sweepInt);
153
154                 sleep(2);
155                 for(i=0; i<REDO; i++) {
156                     retval = requestValue(sock1_fd, &timestamp, &dac_voltage);
157                     usleep(500000);     //wait 0.5 sec
158                     //sleep(8);
159                     if(i==0) {
160                         printf("DAC input nominal: %lf[V], DAC input decimal: %d, DAC output measured: %lf[V]\n", sweepV, sweepInt,
      dac_voltage);
161                         fprintf(logfile, "%lf; %d; %lf; %"PRIu64" \n", sweepV, sweepInt, dac_voltage, timestamp);
```

```
162                     }
163                     fprintf(logfile, " ; ; %lf; %"PRIu64" \n", dac_voltage, timestamp);
164                 }
165                 if(dac.stepSize < DAC_ULSB)     //prevent an infinite for-loop
166                     break;
167             }
168             fclose(logfile);
169             break;
170         case 'i':       //test the ADC_I unit
171             printf("step up ADC_I with DAC\n");
172             if(connectToSock(&sock2_fd, sock2_path) < 0) {
173                 runFlag = false;
174                 break;
175             }
176             usleep(500000);   //wait 5ms for servers registration
177             if(requestToMasterClient(sock2_fd) < 0) {
178                 runFlag = false;
179                 break;
180             }
181
182             logfile = fopen(FILE_PATH_ADC_I, "a+");
183             if(logfile < 0) {
184                 perror("can not open logfile");
185                 break;
186             }
187             getParams(&dac);
188             fprintf(logfile, "Linearity test for ADC_I\n"
189                              "DAC input nominal [V]; DAC input decimal; DAC output measured [V]; timestamp; "
190                              "ADC_I output decimal; ADC_I output value [A]; ADC_I input measured [A]\n");
191             //for loop to sweep the DAC steps
192             for(sweepV=dac.startVal; sweepV<=dac.endVal; sweepV+=dac.stepSize) {
193                 int int_ADC_I = 0;              //decimal val from ADC_I
194                 double doub_ADC_I = 0, current = 0;     //current val from ADC_I, DMM4020 current val
195                 uint64_t timestamp = 0;
196
197                 sweepInt = BCHW_parseToDACvalue(sweepV);
198                 BCHW_sendToDAC(CS_PIN_DAC, sweepInt);
199                 sleep(2);
200                 for(i=0; i<REDO; i++) {
201                     retval = requestValue(sock1_fd, &timestamp, &dac_voltage);
202                     retval |= requestValue(sock2_fd, &timestamp, &current);
203                     if(retval < 0)
204                         perror("Failure occurred in requestValue()");
205                     int_ADC_I = BCHW_readFromADC(CS_PIN_ADC_I, ADC_CONV_PIN);
206                     doub_ADC_I = BCHW_parseToCurrent(int_ADC_I);
207                     usleep(500000); //wait 0.5 sec
208
209                     if(i==0) {
210                         printf("DAC output measured: %lf[V], ADC_I output decimal: %d, ADC_I output value: %lf[uA], ADC_I input
        measured: %lf[A] \n", dac_voltage, int_ADC_I, doub_ADC_I, current);
211                         fprintf(logfile, "%lf; %d; %lf; %"PRIu64"; %d; %lf; %lf \n", sweepV, sweepInt, dac_voltage, timestamp,
        int_ADC_I, doub_ADC_I, current);
212                     }
213                     fprintf(logfile, " ; ; %lf; %"PRIu64"; %d; %lf; %lf \n", dac_voltage, timestamp, int_ADC_I, doub_ADC_I, current);
214                 }
215                 if(dac.stepSize < DAC_ULSB)     //prevent an infinite for-loop
216                     break;
217             }
218             fclose(logfile);
219             close(sock2_fd);
220             break;
221         case 'v':       //test the ADC_V unit
222             printf("step up ADC_V with DAC\n");
223             if(connectToSock(&sock2_fd, sock2_path) < 0) {
224                 runFlag = false;
225                 break;
226             }
227             usleep(500000);   //wait 5ms for servers registration
228             if(requestToMasterClient(sock2_fd) < 0) {
229                 runFlag = false;
230                 break;
231             }
232
233             logfile = fopen(FILE_PATH_ADC_V, "a+");
234             if(logfile < 0) {
235                 perror("can not open logfile");
236                 break;
237             }
238             getParams(&dac);
239             fprintf(logfile, "Linearity test for ADC_V\n"
240                              "DAC input nominal [V]; DAC input decimal; DAC output measured [V]; Timestamp; "
241                              "ADC_V output decimal; ADC_V output value [V]; ADC_V input measured [V]\n");
242             //for loop to sweep the DAC steps
243             for(sweepV=dac.startVal; sweepV<=dac.endVal; sweepV+=dac.stepSize) {
244                 int int_ADC_V = 0;              //decimal val from ADC_I
```

```
245                 double doub_ADC_V = 0, voltage = 0;     //voltage val from ADC_V, DMM4020 voltage val
246                 uint64_t timestamp = 0;
247
248                 sweepInt = BCHW_parseToDACvalue(sweepV);
249                 BCHW_sendToDAC(CS_PIN_DAC, sweepInt);
250                 sleep(2);
251                 for(i=0; i<REDO; i++) {
252                     retval = requestValue(sock1_fd, &timestamp, &dac_voltage);
253                     retval |= requestValue(sock2_fd, &timestamp, &voltage);
254                     if(retval < 0)
255                         perror("Failure occurred in requestValue()");
256                     int_ADC_V = BCHW_readFromADC(CS_PIN_ADC_V, ADC_CONV_PIN);
257                     doub_ADC_V = BCHW_parseToVoltage(int_ADC_V);
258                     usleep(500000); //wait 0.5 sec
259
260                     if(i==0) {
261                         printf("DAC output measured: %lf[V], ADC_V output decimal: %d, ADC_V output value: %lf[V], ADC_V input measured
        : %lf[V] \n", dac_voltage, int_ADC_V, doub_ADC_V, voltage);
262                         fprintf(logfile, "%lf; %d; %lf; %"PRIu64"; %d; %lf; %lf \n", sweepV, sweepInt, dac_voltage, timestamp,
        int_ADC_V, doub_ADC_V, voltage);
263                     }
264                     fprintf(logfile, " ; ; %lf; %"PRIu64"; %d; %lf; %lf \n", dac_voltage, timestamp, int_ADC_V, doub_ADC_V, voltage);
265                 }
266                 if(dac.stepSize < DAC_ULSB)     //prevent an infinite for-loop
267                     break;
268             }
269             fclose(logfile);
270             close(sock2_fd);
271             break;
272         case 'q':       //quit program
273             runFlag = false;
274             break;
275         default:        //else
276             printf(".wrong input\n");
277         }
278     }
279     retval = BCHW_releaseDACgpio();
280     retval |= BCHW_releaseADCgpio();
281     retval |= BCHW_releaseADCgpio();
282     close(sock1_fd);
283     close(sock2_fd);
284     if(retval < 0)
285         fprintf(stderr, "Error occurred in BCHW_releaseAll()\n");
286     printf("#LinearityTest end\n");
287     exit(EXIT_SUCCESS);
288 }
289
290 void clear(FILE *file) {    //flushs a file buffer
291     char c;
292
293     while(((c = fgetc(file)) != '\n') && (c != EOF));
294 }
295
296 void printInfo() {
297     printf("This is a test program to drive up linearly the DAC, \n"
298            "ADC_I or ADC_V range and measure their diviations like\n"
299            "gain error, offset error and compare their results against\n"
300            "an professional high resoloution device.\n\n");
301 }
302
303 void usage(char *progname) {
304     fprintf(stderr, "\n\tUSAGE:\t %s [-1 sockFileVDAC] [-2 sockFile] [-h help]\n"
305                     "\n\tOPTIONS: -1\tSocketfile of external device to measure \n"
306                     "\t\t\tDAC voltage; will be overwritten if it exists.\n"
307                     "\t\t\tDefault is '%s'.\n"
308                     "\t\t -2\tSocketfile of external device to compare ADC_I \n"
309                     "\t\t\tor ADC_V;\n"
310                     "\t\t -h\tHelp; view help text.\n"
311                     "\n\tDESCRIPTION:\tTest program to drive up linearly the DAC, \n"
312                     "\t\t\tADC_I or ADC_V range and measure their diviations \n"
313                     "\t\t\tlike gain error, offset error and compare their \n"
314                     "\t\t\tresults against an external professional high \n"
315                     "\t\t\tresoloution device. \n\n"
316                     , progname, UDS_FILE_DAC_VOLTAGE);
317 }
318
319 int connectToSock(int *sock_fd, const char *sockFilePath) {
320     struct sockaddr_un address;     //address struct
321
322     if(sockFilePath == NULL) {
323         printf("No sockFilePath string\n");
324         return -1;
325     }
326
327     //create socket discriptor
```

```
328     *sock_fd = socket(PF_LOCAL, SOCK_STREAM, 0);    //use unix socket in stream mode, nonblock
329     if(*sock_fd <= 0) {         //if an error occurred
330         perror("Client: Failure at socket() call");
331         close(*sock_fd);
332         return -1;
333     }
334
335     //init structs
336     address.sun_family = AF_LOCAL;          //make them to unix sockets
337     strcpy(address.sun_path, sockFilePath);
338
339     //connect to socket file
340     #ifdef DEBUG
341     printf(".try to connect with socket...\n");
342     #endif
343     if(connect(*sock_fd, (struct sockaddr*) &address, sizeof (address)) == -1) {
344         perror("Client: Failure at connect to socket");
345         close(*sock_fd);
346         return -1;
347     }
348     #ifdef DEBUG
349     printf(".Client is connected\n");
350     #endif // DEBUG
351     return 1;
352 }
353
354 int requestToMasterClient(int sock_fd) {
355     MPFrame_t frame;
356     int retval;
357
358     //set endchar of frame
359     BC_initializeFrame(&frame, BCMP_REQ_GETCONTROL, BCMP_getDefaultArgLen(BCMP_REQ_GETCONTROL));
360     retval = BC_sendFrame(sock_fd, &frame);
361     if(retval < 0) {
362         close(sock_fd);
363         perror("Error occurred at BC_sendFrame()");
364         return -1;
365     }
366     //sleep(3);
367     retval = BC_recvFrame(sock_fd, &frame);
368     if(retval <= 0) {
369         close(sock_fd);
370         perror("Error occurred in BC_recvFrame()");
371         fprintf(stderr, "%d, errno %d\n", retval, errno);         //TODO muss wieder weg
372         return -1;
373     }
374     retval = BC_getArgumentAsUint8(&frame, BCMP_RES_CONTROL_OFFSET_CONTROL);
375     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_GRANT)
376         return 1;
377     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_DENIAL)
378         perror("Server response was an denial frame");
379     if(retval == BCMP_RES_CONTROL_ARGVAL_CONTROL_RELEASE)
380         perror("Server answered on a release frame");
381     return -1;
382 }
383
384 int requestValue(int sock_fd, uint64_t *timestamp, double *value) {
385     MPFrame_t frame;
386     int retval;
387
388     BC_initializeFrame(&frame, BCMP_REQ_GETVALUE, BCMP_getDefaultArgLen(BCMP_REQ_GETVALUE));
389     retval = BC_sendFrame(sock_fd, &frame);
390     if(retval < 0) {
391         perror("Error occurred at BC_sendFrame()");
392         return -1;
393     }
394     //usleep(500000);   //TODO muss für release Version verkürzt werden
395     //sleep(3);
396     retval = BC_recvFrame(sock_fd, &frame);
397     if(retval < 0)
398         return -1;
399     *timestamp = BC_getArgumentAsUint64(&frame, BCMP_RES_VALUE_OFFSET_TIMESTAMP);
400     *value = BC_getArgumentAsDouble(&frame, BCMP_RES_VALUE_OFFSET_VALUE);
401     return 1;
402 }
403
404 void getParams(struct params *dac) {
405     double startVal, endVal, stepSize;
406
407     memset(dac, 0x00, sizeof(dac)); //init struct with zeros
408     //get start value, end value and step size parameters
409     printf("start value (voltage range %1.3f V - %1.3f V): \n", DAC_LOW_SCALE*DAC_ULSB, DAC_FULL_SCALE*DAC_ULSB);
410     scanf("%lf", &startVal);
411     clear(stdin);
412     if((startVal < DAC_LOW_SCALE) || (startVal > DAC_FULL_SCALE)) {
```

```
413             printf("value is out of permissible range\n");
414             return;
415         }
416         printf("end value (voltage range %1.3f V - %1.3f V): \n", startVal, DAC_GAIN*DAC_VREF);
417         scanf("%lf", &endVal);
418         clear(stdin);
419         if((endVal < startVal) || (endVal > DAC_FULL_SCALE)) {
420             printf("value is out of permissible range\n");
421             return;
422         }
423         printf("step size (%1.3f V - %1.3f V): \n", 0.0, endVal-startVal);
424         scanf("%lf", &stepSize);
425         clear(stdin);
426         if((stepSize < 0) || (stepSize > (endVal - startVal))) {
427             printf("value is out of permissible range\n");
428             return;
429         }
430         printf("\n");
431         dac->startVal = startVal;
432         dac->endVal = endVal;
433         dac->stepSize = stepSize;
434     }
```

E. 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

22. März 2016

Bachelorthesis Nico Rieckmann

Entwicklung einer Regelsoftware auf Linux-Basis für eine optische Batterie-Analyse

Motivation

Im Rahmen der Forschungsaktiviäten der Arbeitsgruppe Batteriesensorik soll der Zustand von Batterien durch neuartige Verfahren erfasst werden.
Eine genaue Kenntnis des Batteriezustandes ist für Lithiumbatterien besonders wünschenswert, weil kritische Betriebszustände sicher vermieden werden sollen. Neben der Messung von elektrischen Größen soll in einem Experimentalaufbau eine optische Beobachtung des Elektrodenmaterial erfolgen.

Aufgabe

Herr Nico Riekmann erhält die Aufgabe, eine Messzelle mit Mikroskopkamera mit einem kompakten Einplatinen-Computer (Raspberry Pi) mit einem Linux-Betriebsystem zu verbinden. Das System hat vier Hauptaufgaben, welche synchron auszuführen sind:

1. die Steuerung des Lade/Entladebetriebes (Zyklier-Ablaufsteuerung)
2. die Erfassung von Spannung, Strom und Temperatur
3. die Bilderfassung und Aufzeichung
4. die Steuerung der Beleuchtungs-LED für die Mikroskopkamera (optionale Aufgabe)

Hierzu gibt es aus Vorarbeiten eine Zusatzplatine mit Steuersoftware, die jedoch vollständig erprobt und ggf. teilweise überarbeitet werden soll. Die Steuerung des Lade/Entladebetriebes soll eine Regelung von Spannung bzw. Strom für die Messzelle mit umfassen. Ergänzend zur Erfassung der Strom- und Spannungswerte sollen auch Spannungs- und Strommessungen mit Präzisionsmessgeräten unterstützt werden
Kern der Arbeit stellt die Software dar, die den Bilderfassungs-, Zyklier- und Messablauf steuert. Diese Software soll modular und flexibel aufgebaut werden und die Prozesskommunikation und weitere Möglichkeiten des Betriebssystems nutzen. Diese Software soll die flexible Gestaltung der beabsichtigen Messreihen ermöglichen. Durch systematische Erprobungen und Messreihen ist die Funktion zu überprüfen.
Für die Abschlussarbeit sind die folgenden Arbeitspakete geplant:

1. Recherche
    - Recherche der relevanten Literatur und Abschlussarbeiten,
    - Einordnung in die Projektarbeit der BATSEN-Arbeitsgruppe
    - Diskussion und Fixierung der benötigten Spezifikationen
    - Einarbeitung in den Batteriemessplatz und die Linux-Konfiguration

2. Inbetriebnahme des Batteriemessplatzes und Fehleranalyse

    - Inbetriebnahme und Erprobung des Zyklierungsbetriebes
    - Bewertung und Fehleranalyse der bestehenden Hard- und Software (Errata sheet)
    - Durchführung und Dokumentation der Korrekturmaßnahmen
    - Bewertung und Entscheidung von Lösungsalternativen (z.B. Anschluß externer Präzisions-Messinstrumente oder Änderung der ADC-Schaltung)

3. Konzeption der Software des Batteriemessplatzes

    - Erarbeiten von Software-Konzepten
    - Entscheidungsfindung für eine monolithische oder modulare Softwarestruktur
    - Spezifikation Definition
    - Erarbeitung eines Sicherungskonzeptes (Messdatenintegrität und Schutz der Messzellen)
    - optional: Softwareinbindung einer Temperaturerfassung mit SPI-Sensor für die Messzelle
    - optional: Entwurf einer LED-Steuerung mit weißen und mehrfarbigen Leuchtdioden

4. Entwicklung der Steuer- und Regelsoftware

    - Schrittweise Implementation einer detaillierten Softwarestruktur (Flowcharts, Prozessdiagram, Protokollspezifikationen)
    - Kommunikationsprotokoll zwischen Datenaufnahme und Zykliersteuerung (BCMP)
    - Implementation von Clienten (Regelprozess) und Server (Mess- und Steuerprozesse)
    - Entwicklung von Test Units für die Einzelkomponenten
    - Modultest und ggf. Korrektur

5. Erprobung und Tests des Gesamtsystems

    - Definition eines Testplan mit geeigneten Testzyklen
    - Erprobung im Langzeitbetrieb
    - Bewertung der erreichten Messergebnisse (Genauigkeit ADC der Erweiterungsplatine vs. externe Messgeräte, Genauigkeit des DAC, Zeitverhalten, u.a.)
    - Visualisierung und Darstellung der Ergebnisse (Bilddarstellungen und Messwert-Kurven)

6. Ergebnisse, Bewertung und Fazit

    - Dokumentation der Software für spätere Erweiterung und als Bedienungsanleitung
    - Bewertung der gewählten Lösungen und Ergebnisse
    - Diskussion von offenen Punkten und möglichen Verbesserungen

Dokumentation

Die entwickelte Software, die gewählten Lösungen und die Funktionsweise sind gut nachvollziehbar und für die zukünftige Nutzung zu dokumentieren. Die Erprobungsergebnisse sind in aussagefähigem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch einordnend zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

F. Inhalt des Datenträgers

Auf dem Datenträger befinden sich der gesamte Quellcode der BatCycle Programme und kleine Testprogramme, sowie ein vollständiger Schaltplan der Messplatine. Für den Raspberry Pi 2 ist ein Systemabbild als Imagedatei beigelegt und ein Remote Control Tool, um diesen zu bedienen. Der Datenträgersatz ist beim Erst- und Zweitprüfer hinterlegt. Der Inhalt der DVD ist wie folgt aufgelistet:

- PDF-Datei dieser Bachelorarbeit.
- Version 2 von dem Schaltplan der Messplatine.
- Datenblätter der Platinenbauteile.
- Fotos des Gesamtsystems.
- Matlab Skripte für eine Auswertung der Testzyklen.
- Matlab Skripte für eine Auswertung von LinTest.
- Quellcode aller BatCycle Programme.
- Test Quellcodes.
- Systemabbild des aktuellen Raspberry Pi 2.
- Flash-Tool um das Image auf eine SD-Karte zu spielen.
- Remote Control Tool um über Ethernet auf den Raspberry Pi 2 zugreifen zu können.

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, 22. Juni 2016	
Ort, Datum	Unterschrift